Hardware acceleration of post-quantum cryptography for embedded systems

Guilhèm Assael

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : MSTII - Mathématiques, Sciences et technologies de l'information, Informatique
Spécialité : Mathématiques
Unité de recherche : Institut Fourier

Accélération matérielle de cryptographie post-quantique pour les systèmes embarqués

Hardware acceleration of post-quantum cryptography for embedded systems

Présentée par :

Guilhèm ASSAEL

Direction de thèse :

Philippe ELBAZ-VINCENT PROFESSEUR DES UNIVERSITES, UNIVERSITE GRENOBLE ALPES	Directeur de thèse

Rapporteurs :

EMMANUEL PROUFF
SENIOR SCIENTIST, SORBONNE UNIVERSITE
ARNAUD TISSERAND
DIRECTEUR DE RECHERCHE, CNRS BRETAGNE ET PAYS DE LA LOIRE

Thèse soutenue publiquement le **17 décembre 2024**, devant le jury composé de :

JEAN-GUILLAUME DUMAS, PROFESSEUR DES UNIVERSITES, UNIVERSITE GRENOBLE ALPES	Président
PHILIPPE ELBAZ-VINCENT, PROFESSEUR DES UNIVERSITES, UNIVERSITE GRENOBLE ALPES	Directeur de thèse
ARNAUD TISSERAND, DIRECTEUR DE RECHERCHE, CNRS BRETAGNE ET PAYS DE LA LOIRE	Rapporteur
PETER SCHWABE, PROFESSEUR, MAX PLANCK INSTITUTE	Examinateur
FRANÇOIS-XAVIER STANDAERT, PROFESSEUR, UNIVERSITE CATHOLIQUE DE LOUVAIN	Examinateur
ALINE GOUGET, DOCTEURE EN SCIENCES, ANSSI	Examinatrice

Invités :

RUGGERO SUSELLA
SENIOR SCIENTIST, STMicroelectronics
GUILLAUME REYMOND
INGENIEUR, STMicroelectronics

Remerciements

Je me dois d'ouvrir ce document, fruit de plus de trois ans de travail, en remerciant ceux qui l'ont rendu possible et qui ont participé à son aboutissement. Je remercie d'abord sincèrement les membres de mon jury d'avoir accepté cette fonction. Je leur suis d'autant plus reconnaissant que tous étaient déjà très occupés, et qu'au moins l'un d'entre eux a choisi de maintenir sa participation malgré un imprévu qui l'aurait très légitimement délié de cette charge. Je suis particulièrement obligé vis-à-vis des rapporteurs, qui ont bien voulu accepter la responsabilité supplémentaire de la relecture de mon manuscrit. En plus de ce rôle, M. Emmanuel Prouff a également participé au comité annuel de suivi de ma thèse, et je luis sais gré des conseils avisés qu'il m'a donnés et de sa bienveillance.

J'ai bénéficié durant ma thèse du financement de l'entreprise STMicroelectronics, avec la participation de l'ANRT. J'ai également profité du financement ANR-15-IDEX-02 par l'intermédiaire du CyberAlps de Grenoble.

Pour reprendre maintenant l'ordre chronologique, il me faut remonter à l'époque de mon stage de fin d'études d'ingénieur au sein de STMicroelectronics. C'est durant ce stage qu'est née l'idée de cette thèse : non pas de moi, mais de Silvia, ma tutrice de stage, et du chef de l'équipe de R&D en sécurité embarquée, Bernard. C'est sous leur impulsion que je me suis lancé dans ce travail passionnant, et je leur en suis reconnaissant.

Mes remerciements vont à Philippe, qui a suivi et orienté mes travaux par son jugement sûr, encourageant mes succès et m'aidant à surmonter les difficultés. Il a aussi su, ce qui n'est pas rien, supporter patiemment mon incompétence administrative et les menus mais toujours urgents désagréments qui s'en sont suivis.

Au jour le jour, ce sont principalement Guillaume et, une fois Silvia et Bernard partis, Ruggero, qui ont accompagné mes recherches. Je leur suis redevable de tout le temps qu'il m'ont consacré, tous les conseils techniques et pratiques qu'ils m'ont prodigué, mais également de la confiance qu'ils m'ont accordée.

Je tiens également à remercier toute l'équipe sécurité de ST. Tous ont participé à rendre ma thèse plus fructueuse, et surtout plus sereine. Je leur suis reconnaissant pour l'ambiance d'entraide et d'amitié qui règne au sein de l'équipe, pour l'attention que chacun porte aux autres. Ils sont nombreux et variés, ces édifices auxquels chacun a apporté sa pierre, qu'il s'agisse de mener à bien des projets importants, ou de fêter quelque événement autour d'un bon repas et de quelques bonnes bouteilles ! Je suis heureux de pouvoir continuer ma carrière dans cette équipe sympathique et efficace.

J'étendrai ces remerciements à ma famille et mes amis, qui m'ont soutenu dans cette course de fond par leurs pensées et leurs prières, par leurs conseils et par leur compréhension. J'aime à croire qu'avec l'achèvement de ma thèse, je serai maintenant un peu plus régulièrement, ou du moins un peu plus prévisiblement disponible.

Abstract

Most current digital communications are secured using cryptography relying on the hardness of integer factorization and of computing discrete logarithms. However, it has been known for thirty years that a hypothetical *quantum computer* of sufficient size could solve both problems efficiently, making this cryptography insecure. Ample research in the field has now produced quantum computers of ever-increasing size and quality, which threaten current (so-called *classical*) cryptography for the immediate future.

New cryptographic constructions, called *post-quantum* for their presumed resilience against quantum cryptanalysis, have thus been studied. Notably, the US National Institute of Standards and Technology (NIST) standardized three post-quantum algorithms in August 2024: two digital signature algorithms and a key-encapsulation mechanism.

Despite extensive research, implementing these algorithms securely remains difficult, mainly due to the concern of *implementation attacks*. Two vivid security threats are *side-channel attacks*, through which secret information can be determined by monitoring the activity of the secure device, and *fault-injection attacks*, that try to disturb the device to make it insecure. *Embedded devices* are particularly exposed to these attacks, but few studies have concentrated on effective countermeasures bearing a sufficiently low cost to be accommodated by the limited computational resources of such devices.

This thesis, focused on the hardware implementation of post-quantum cryptography for embedded devices, aims at filling this gap. We study two of the NIST standards: the key-encapsulation mechanism Kyber (named ML-KEM by NIST), and the digital signature algorithm Dilithium (ML-DSA), both based on *module lattices*. Their similar construction helps implement both of them efficiently in hardware.

We start by presenting lattice-based cryptography in general, and the two schemes under study in particular, underlining the keys of implementing them efficiently and securely in hardware. We then review the state of the art of side-channel attacks and countermeasures against the two schemes and related constructions. Our review includes software implementations, since they have attracted much more attention, and attacks and countermeasures are mostly transferable between software and hardware.

Building upon a line of existing side-channel attacks against the *number-theoretic transform* used in Kyber, we adapt them to deal with recent, strongly optimized software implementations of the scheme. We show that these optimizations do not hamper the success of our attack, even in noisy conditions. As a second contribution, we propose an efficient implementation of masked arithmetic addition over Boolean shares, which is an essential component of side-channel countermeasures for lattice-based cryptography and other constructions. Our proposal has native support for modular addition, which helps protect the two schemes of interest with adequate performance but very low hardware area requirements. Three more specialized side-channel protections follow, the first of which is a new method to mask Kyber ciphertext compression, with an asymptotic complexity improvement over previous works. We also show how to mask the small-norm uniform sampling operation of Dilithium, for which no protection complying with the standard seems to have been previously proposed. Finally, we study an efficient shuffling countermeasure with qualitative advantages over previous solutions, that should strengthen it against shuffling-aware attacks. Overall, combining our contributions with the state of the art can bring security and efficiency gains for the hardware implementation of post-quantum cryptography.

Keywords Post-quantum cryptography, lattice-based cryptography, ML-KEM, ML-DSA, side-channel attacks and countermeasures

Résumé

La plupart des communications numériques actuelles sont protégées par des méthodes cryptographiques basées sur le problème de la factorisation des entiers ou celui du logarithme discret. Cependant, nous savons depuis trente ans qu'un éventuel *ordinateur quantique* de taille suffisante pourrait résoudre ces problèmes efficacement, rendant cette cryptographie obsolète. Or, des ordinateurs quantiques de plus en plus conséquents ont récemment vu le jour, menaçant la cryptographie dite *classique* à brève échéance.

De nouvelles constructions cryptographiques, appelées *post-quantiques*, ont donc été étudiées pour leur résistance escomptée à la cryptanalyse quantique. Le *National Institute of Standards and Technology* (NIST) américan a notamment standardisé trois de ces algorithmes en août 2024 : deux signatures numériques et un mécanisme d'encapsulation de clef.

Malgré des années de recherche, implémenter ces algorithmes de manière sûre demeure difficile, notamment face aux *attaques physiques*. Parmi elles, les *attaques par canaux cachés* visent à déterminer des données secrètes en analysant l'activité du dispositif cryptographique, tandis que les *attaques par injection de fautes* perturbent activement ce dispositif pour le rendre vulnérable. Les *systèmes embarqués* y sont particulièrement exposés, mais peu de contremesures efficaces ont un coût suffisamment faible pour être toléré par ces dispositifs aux resources de calcul limitées.

Cette thèse, portant sur l'implémentation matérielle de la cryptographie post-quantique pour les systèmes embarqués, vise à combler ce vide. Deux standards du NIST y sont étudiés : le mécanisme d'encapsulation Kyber (nommé ML-KEM par le NIST) et l'algorithme de signature Dilithium (ML-DSA), tous deux basés sur les *réseaux modules*. Leur construction similaire facilite leur implémentation matérielle conjointe.

Ce mémoire s'ouvre sur une présentation de la cryptographie basée sur les réseaux, et en particulier des deux schémas étudiés, soulignant leurs aspects de sécurité et de performance. Nous rappelons l'état de l'art des attaques physiques et contremesures relatives à ces schémas ou à d'autres similaires. Nous y incluons les implémentations logicielles, bien plus présentes dans la littérature, sachant que les attaques et contremesures sont similaires dans les contextes logiciel et matériel.

Continuant une lignée de travaux qui attaque la *transformée en nombres entiers* de Kyber, nous montrons comment s'adapter aux implémentations logicielles optimisées de Kyber. Ces optimisations n'entravent pas le succès de notre attaque, même dans des conditions de mesure bruitées. Notre deuxième contribution est une architecture économique pour l'addition masquée sur *shares* booléennes, composant essentiel de contremesures aux attaques par canaux cachés, pour la cryptographie basée sur les réseaux comme pour d'autres constructions. Notre proposition, supportant nativement l'addition modulaire, aide à conserver une performance satisfaisante pour les deux schémas à l'étude, malgré une occupation de surface très faible. Nous ajoutons à cela trois contremesures spécialisées, à commencer par une solution de masquage pour l'opération de compression du chiffré de Kyber, de meilleure complexité que l'art antérieur. Nous masquons également l'opération d'échantillonnage uniforme pour la clef privée de Dilithium : aucune des protections précédemment publiées pour cette opération n'était conforme au standard. Enfin, nous étudions une solution de permutation aléatoire des opérations, qui par des avantages qualitatifs sur la littérature, est susceptible de mieux résister aux attaques. En conclusion, nos contributions combinées à l'état de l'art promettent une implémentation matérielle plus efficace et plus sûre de la cryptographie post-quantique pour les systèmes embarqués.

Mots-clefs Cryptographie post-quantique, cryptographie basée sur les réseaux, ML-KEM, ML-DSA, attaques par canaux cachés et contremesures

Contents

1 Introduction

Cryptography is concerned with protecting information against adversarial parties, and with providing convincing evidence that this objective has been successfully satisfied. This protection is traditionally split into the following properties.

Confidentiality is satisfied when the relevant information, once appropriately protected, can only be understood by its intended recipient, while other parties see opaque data from which they can extract to meaningful knowledge.

Authenticity and non repudiation consist in proving that a piece of data originates from a specific entity, and in the inability for this entity to convincingly deny this statement.

Integrity is satisfied when a party can determine with certainty that a given piece of information has not been altered after issuance, in particular in relation with the property of authenticity.

Public-key cryptography pursues these objectives in situations in which there is an intrinsic *asymmetry* between communicating parties (hence its other name of *asymmetric cryptography*). It typically allows for the communication of arbitrary individuals with a single entity (let us call her Alice), which singularly possesses the ability to issue or receive secured information. The two most illustrative applications of public-key cryptography are public-key encryption and digital signature. Thanks to public-key encryption, anyone can encrypt messages intended for Alice, but only Alice can decrypt them and make sense of the information they carry. Digital signature is, in some sense, the opposite: it allows Alice to send authenticated messages to anyone, and any recipient of the message to check that it indeed originates from Alice and has not been altered by any other party.

Cryptography employs mathematical constructions that translate the computational hardness of some problems into the desired security properties. This computational hardness is always based on some assumptions that, along the history of cryptography, have been constantly put to the test. On one hand, the gradual evolution of computing capabilities, and incremental improvements to algorithms solving these problems, progressively weaken older cryptographic constructions, to the point that they no longer hold their security unless they are adjusted for this evolution. On the other hand, major mathematical and technical breakthroughs can render some constructions entirely insecure, when the problems they are based upon are found to be easy to solve.

1.1 Quantum computing and quantum cryptanalysis

The early 1980s saw the introduction of an entirely new algorithmic concept, that of quantum algorithms. It progressively became clearer that these algorithms, able to leverage the intrinsic properties of quantum mechanics, would have vastly different

capabilities from those of classical algorithms. Notably, some mathematical problems that are inherently hard to solve classically would become easy to solve on a hypothetical quantum computer. We refer the reader to appendix A for more details on quantum computation, and will only recall here its relevance to cryptography.

What might be the most widely known quantum algorithm, *Shor's algorithm* [Sho94], was a major breakthrough in the field of cryptanalysis. While the prime-factoring problem is classically hard, requiring super-polynomial running time, Peter Shor devised a quantum algorithm that could solve it in polynomial time. He also showed how to solve the discrete-logarithm problem in quantum polynomial time, while this other task takes exponential effort (for the elliptic-curve discrete logarithm) with classical cryptanalysis. This latter algorithm was refined by Proos and Zalcka [PZ03] with several optimizations. The hardness of these two problems being at the basis of the security of most constructions of public-key cryptography, a direct consequence was that, should a working and large enough quantum computer ever be build, current public-key cryptography would become obsolete.

At the time of Shor's works, no quantum computers were in existence, and his findings were purely theoretical. However, as practical quantum computers started being built, the threat started to become material, though distant. Research in quantum computing theory and technology has benefited from extensive funding in recent years, thereby allowing for a very quick progress in the field. In particular, significant milestones have been reached recently in terms of quantum error correction (see e.g. [GC24, RAC+24]), which is a prerequisite for a cryptographically relevant quantum computer, i.e., one that has sufficient size and fidelity to break cryptosystems that were secure against classical computers.

1.2 Post-quantum cryptography and its standardization by NIST

Considering this major threat to the cryptographic algorithms that are currently deployed, various initiatives were started during the 2010s to replace these algorithms with new constructions that resist to quantum computers. One of the most widely publicized of these initiatives has been the *post-quantum standardization process* initiated by the US National Institute of Standards and Technology (NIST) at the end of 2016 [Nat16]. This process, similarly to previous cryptographic contests run by the standardization agency, would evaluate proposals from the cryptographic community, and select the best out of them for standardization.

The first round of this process, spanning over year 2018, received 82 submissions, classified among key-encapsulation mechanisms (KEMs) and public-key encryption mechanisms (PKEs) on one hand, and digital signature algorithms on the other hand. Out of these, 69 candidates met the submission requirements, and 26 (including some mergers) progressed to the second round of the process. The second round proceeded likewise, from 2019 to 2020, and forwarded 15 candidates to the third round, which was concluded in July 2022 [AAC+22]. The decisions at the end of the third round, summarized in

table 1.1, were the following. The key-encapsulation mechanism Kyber[1] and the digital signatures Dilithium, Falcon and SPHINCS+ were selected for immediate standardization, while three other candidate KEMs were kept for a fourth round of evaluation, with the aim of diversifying the KEM portfolio.

Indeed, during PQC standardization process, several devastating attacks were found against schemes that had been previously considered secure, in particular GeMSS [TPD21] and SIKE [CD23], which are signaled as *broken* in the table since a key component in the design of the schemes has been shown insecure. While the scheme is not strictly broken, an attack against Rainbow has been found which strongly reduces its security, and would be unreasonably costly to protect against [Beu21, Beu22].

In August 2024, NIST published the first three standards accrued from this process: ML-KEM (FIPS 203) [FIP24a] from Kyber, ML-DSA (FIPS 204) [FIP24b] from Dilithium, and SLH-DSA (FIPS 205) [FIP24c] from SPHINCS+. The first two of these will be discussed in chapter 2.

Table 1.1: Conclusions of round 3 of NIST post-quantum standardization process

(a) Key-encapsulation mechanisms

Candidate	Underlying construct	Decision
Kyber	Structured lattices	Standardized: ML-KEM
Classic McEliece	Error-correcting codes	4th round
BIKE	Quasi-cyclic error-correcting codes	4th round
HQC	Quasi-cyclic error-correcting codes	4th round
NTRU	Structured lattices	Rejected (better alternatives)
SABER	Structured lattices	Rejected (better alternatives)
NTRU Prime	Structured lattices	Rejected (better alternatives)
FrodoKEM	Plain lattices	Rejected (inefficient)
SIKE	Elliptic-curve isogenies	4th round, then broken

(b) Digital signature algorithms

Candidate	Underlying construct	Decision
Dilithium	Structured lattices	Standardized: ML-DSA
Falcon	Structured lattices	Will be standardized: FN-DSA
SPHINCS+	Stateless hash-based	Standardized: SLH-DSA
Picnic	Symmetric-based zero-knowledge proof	Rejected (insufficient trust)
Rainbow	Multivariate	Rejected (insecure)
GeMSS	Multivariate	Rejected (broken)

[1]For consistency within this document, we will not respect the various typographic styles chosen by the submitters for the names of their candidate schemes, and for conciseness we remove the CRYSTALS- prefix from Kyber and Dilithium.

1.2.1 Position of French ANSSI

The French national cybersecurity agency, *Agence Nationale de la Sécurité des Systèmes d'Information* (ANSSI), has been providing recommendations on the transition toward quantum-resistant cryptography for some time. It notably issued two successive position papers highlighting its view on this matter (with no normative role yet), in 2022 and 2023 [ANS22, ANS23].

Their first position paper acknowledges the seriousness of the threat of quantum computers against asymmetric cryptography, and the urgency of protecting against them. For this purpose, quantum key distribution[2] is not seen as a valid solution except for niche applications, in particular due to strong technical and logistic barriers against its use. Instead, they view post-quantum cryptography as the way forward, since it essentially provides the same functionality as pre-quantum asymmetric constructions, while being secure against quantum-enabled adversaries [ANS22].

The need for hybridization ANSSI highlighted that post-quantum algorithms were not mature, and would stay so for some time, but were nonetheless needed in the short term: hence their recommendation to quickly start transitioning to *hybrid* mechanisms that combine pre-quantum and post-quantum constructions, in such a way that breaking the overall construction requires breaking both constituents. This allows to quickly benefit from the presumed quantum security of post-quantum algorithms, while almost certainly[3] not losing any security with respect to the current status, even in case the new algorithms are in fact vulnerable[4]. Since the size of the cryptographic material in post-quantum schemes is anywhere between a few times and several orders of magnitude larger than its equivalent in pre-quantum schemes, ANSSI believes that the additional cost of hybrid mechanisms on top of post-quantum–only ones is negligible [ANS22].

Transition roadmap ANSSI advocates for a three-phase transition roadmap: in the first one, maintaining pre-quantum security will be mandatory, but adding post-quantum *defense-in-depth* (i.e., additional layers that can only increase the overall security of the system) will be optional. In this phase, no quantum resistance can be claimed since the post-quantum algorithms are still immature.

[2] Quantum key distribution aims to perform secure key establishment between two parties, using quantum-physical properties as the sole way to achieve security. The two parties need to share a quantum communication channel, as well as a classical communication channel on which they have no expectation of privacy but which allows for authentication. By transfering specially prepared quantum states over the quantum channel, and exchanging information about the process over the classical channel, the two parties can theoretically agree on a secret, and simultaneously check that no eavesdropper intercepted their communication over the quantum channel. While the security of this concept only depends on fundamental properties of quantum states, its implementation poses innumerable challenges, including the possibility for imperfect hardware to introduce vulnerabilities not accounted for by the model.

[3] Design or implementation mistakes in the hybridization layer could render the whole construction insecure [Bec21]; however, the surface for error at this level is relatively small.

[4] As exemplified by the above-mentioned devastating attacks on Rainbow during the third round of NIST PQC standardization, and on SIKE after the conclusion of the third round [Beu22, CD23].

In the second phase, the scope of hybridization will be strengthened: quantum resistance can be claimed, in which case it is explicitly evaluated in the delivery of security certifications. Post-quantum security may become mandatory in the contexts in which long-term security is targeted; still, when post-quantum mechanisms are used, hybridization with vetted pre-quantum algorithms remains compulsory.

When enough time has passed for post-quantum algorithms to be confidently considered as secure, the third phase will begin: hybridization with pre-quantum algorithms can optionally be stopped, and post-quantum algorithms can be used as a standalone replacement of current cryptography.

ANSSI insists the first phase be started as soon as possible. The switch to the second phase was initially planned no earlier than 2025 [ANS22], but then anticipated to 2024–2025 [ANS23]. The third phase is expected to start no earlier than 2030 [ANS22].

Year 2030 is also considered as the time when the sole protection against classical adversaries becomes insufficient: ANSSI states that the systems meant to provide cryptographic protection until 2030 (in particular, those that may still be used after 2030 without updates) must switch to phase 1 immediately [ANS22, ANS23].

Choice of the algorithm and security level ANSSI does not define an a-priori list of endorsed algorithms, and does not mandate using one of those standardized by NIST: the main criterion, instead, is to use a algorithm with a stable specification that has been thoroughly scrutinized by the community [ANS22]. As an example of this, they mention FrodoKEM [NAB+20]: while no longer being considered in NIST standardization, it has been studied during the three rounds of the process, and is being standardized by ISO.

They thus recommend to use ML-KEM or FrodoKEM for key encapsulation, and either ML-DSA, Falcon (to be standardized as FN-DSA), LMS, XMSS or SLH-DSA for signature. They recall that LMS and XMSS, being stateful hash-based signatures, are only suited for very specific applications such as software updates, and that state management is of a critical importance. They also highlight that Falcon is ill-suited to applications in which side-channel attacks are a concern [ANS23].

Their recommendation in terms of the choice of security level is to go for the highest one that is possible for the application, preferably equivalent to NIST level 5 or level 3, except for Falcon, for which they only mention level 5 since it has no level-3 parameter set [ANS23].

On hash-based signature schemes Since hash-based signature schemes ground their security on much simpler assumptions than other post-quantum schemes, they are considered by ANSSI as sufficiently reliable for standalone use without hybridization [ANS22, ANS23].

1.2.2 Position of German BSI

Similarly, the German federal office for Information Security (BSI) updated in early 2024 its recommendations on cryptographic mechanisms [BSI24], including a discussion of

post-quantum cryptography. No explicit transition timeline is given, except for bounding the validity of the technical recommendation to no later than 2030.

The need for hybridization Like ANSSI, BSI highlights that post-quantum mechanisms being still very recent, it is difficult to be as confident in their security as for pre-quantum schemes. Their recommendation is thus to rely, whenever possible, on hybridization mechanism combining pre- and post-quantum cryptography, so that the resulting cryptograms are at least as strong as either of the cryptographic schemes in use. This direction holds both for key-encapsulation mechanisms and for digital signature algorithms. They state the same exception as ANSSI: due to their more conservative security assumptions, hash-based signatures do not need hybridization.

Algorithms and security levels For key-encapsulation mechanisms, three post-quantum constructions are recommended by BSI: FrodoKEM [NAB+20], which is undergoing standardization by ISO but has been dismissed by NIST for being inefficient; Classic McEliece [ABC+22], also being standardized by ISO, and likely by NIST; and, ML-KEM [FIP24a], which is the NIST standard issued from Kyber. For all three algorithms, the parameter sets they recommend correspond to NIST security levels 3 and 5.

For digital signature, three types of schemes are recommended: SLH-DSA [FIP24c], which is the NIST standard pertaining to SPHINCS+; ML-DSA [FIP24b], standardized by NIST from Dilithium; and, Merkle signatures [CAD+20]. The last type, comprised of LMS, XMSS, HSS, and XMSS^{MT}, describes *stateful* hash-based signatures, which have stringent requirements in terms of key management, to ensure that no reuse of one-time signature keys ever happens, since this event would break their security.

1.3 Hardware design for embedded systems

Embedded systems are computing devices dedicated to a particular application relating, for instance, to the control of electromechanical systems, to the acquisition or sensor data, or to the management of electronic communication equipment. Owing to the characteristics of the devices they are embedded in, they often have limited computational resources, both for cost and for power-consumption reasons.

These embedded systems always involve some kind of communication equipment, either as their main function, or as a means to accomplish this function. They thus need to secure these communications through cryptographic methods. Since they are physically included inside the application they control, they can often be physically accessed by anyone, including malicious actors. Consequently, they are bound to process sensitive cryptographic material while being in a potentially hostile environment. This raises two very important concerns for the security of the cryptography they implement: that of side-channel attacks, and that of fault-injection attacks.

1.3.1 Side-channel attacks

Cryptographic schemes are designed for security in the black-box model, in which it is assumed that adversaries only have access to the non-sensitive inputs and outputs of the algorithms, and have no information on any of the intermediate processing steps. However, this assumption is rarely satisfied for their *implementations*: for instance, the time it takes for a cryptographic algorithm to complete can be determined by an adversary in a wide range of circumstances. Consequently, this information should not reveal any sensitive data: otherwise, timing attacks can be exerted, potentially leaking secrets to an attacker [Koc96, BB03, JFB+22, RCDB23].

An adversary can thus acquire information on what is processed by the algorithm through other means than the formally defined inputs and outputs of this algorithm. Such attacks are called *side-channel attacks*.

Especially in the case of embedded systems, timing attacks are very far from being the only side-channel attacks. By monitoring the power consumption of the device, an attacker can get noisy but fine-grained information on intermediate values successively processed by the algorithm. While power side-channel attacks only give global measurements of the device activity at each time sample, it is actually possible to get more localized measurements through electromagnetic attacks. These can pick up the electromagnetic emissions of a specific part of the device, to focus on the sole activity of this part: for instance, a whole cryptographic module, or some smaller part of it (see, e.g., [GMO01, HMH+12]).

Protecting cryptographic implementations against these threats is of prime importance. Slightly less than three decades of research in this field have brought several countermeasures against these attacks, but the side-channel security of cryptography very much remains a race between implementers and attackers, particularly since countermeasures can easily raise the cost of cryptographic implementations beyond what can be tolerated by applications.

1.3.2 Fault-injection attacks

Side-channel attackers are passive adversaries that merely monitor the activity of the cryptographic device to steal sensitive data. However, active adversaries are also a concern. By perturbing the power supply of the device, its clocking, its temperature, by subjecting it to electromagnetic emissions, or through many other disturbances of its environment, it is possible to alter its behavior in such a way as to inject vulnerabilities in its cryptographic operations. Again, protecting against this type of attack is no easy task, and needs dedicated attention during the implementation of cryptographic algorithms [Cla88, BDL97, BS97, RCDB23].

1.4 Purpose and organization of this work

The objective of this PhD thesis is to study the implementation of post-quantum cryptography for embedded devices. Specifically, the focus is put on two algorithms of

lattice-based cryptography, the above-mentioned Kyber and Dilithium candidates to the NIST post-quantum standardization process. As alluded to, the security of the implementations against side-channel attacks, being determining in an embedded context, is put as a priority. However, given the cost-sensitiveness of embedded devices, keeping side-channel countermeasures as inexpensive as possible is key.

Outline This document starts by setting up some foundational notions in chapter 2. In particular, the notations that will be employed are defined, and the principles of lattice-based cryptography are introduced. An in-depth overview of the two cryptographic schemes of interest, Kyber and Dilithium, is then provided.

In chapter 3, we carry out a review of the side-channel attacks and fault-injection attacks against lattice-based cryptography, with a focus on their applicability to Kyber and Dilithium. We then examine the state of the art of protections against these attacks.

The exposition of our contributions starts in chapter 4 with the relation of our improved implementation of an existing side-channel attack against an arithmetic component of the Kyber scheme. This chapter corresponds to the work published in [AEVR23].

We then move on to our proposal, published in [AEV24], for a generic operation protected against side-channel attacks: secure modular addition over Boolean shares. Chapter 5 describes this protected operation in depth, proves its security, and evaluates its performance with respect to state-of-the-art equivalents.

Finally, chapter 6 develops our three additional protections against side-channel attacks, which are more intrinsically linked with the two schemes under study. We expose an invention of ours, realizing the protection of the ciphertext-compression operation, together with its formal proof of security. We then explain how to mask the secret-key sampling operation for Dilithium: from the available literature, it seems that this operation had never been protected in compliance with the specification. Lastly, we propose a new implementation of the shuffling countermeasure, that aims at correcting the flaws of previous implementations of the countermeasure. While incomplete, the theoretical and experimental analysis of this solution open promising perspectives.

We conclude on this work in chapter 7, giving an outlook on how our contributions (of which we provide a detailed list in section 7.2) fit within the state of the art, and suggesting followup research directions.

2 Foundations

This chapter establishes the foundational concepts that will inform the rest of this work. It begins with section 2.1 that introduces the main notations. Section 2.2 then discusses lattice-based cryptography and is followed by a description of the two cryptographic schemes under study, Kyber and Dilithium, in sections 2.3 and 2.4.

2.1 Notations

This section introduces the main notations used in this work. Additional notations specific to each chapter will be defined as needed throughout the text.

Prime-order finite fields and cyclic groups We will use the standard notation $\mathbb{F}_q$ to refer to the finite field of order q, a prime or prime power. For arbitrary m, we will also work with the cyclic group of order m, with the usual notation $\mathbb{Z}/m\mathbb{Z}$. Elements of $\mathbb{F}_q$ or $\mathbb{Z}/m\mathbb{Z}$ will be transparently identified with the integer from $\mathbb{Z}$ corresponding to their canonical representative.

Intervals For real numbers $a < b$, we use notations $[a, b]$ and $]a, b[$ respectively for the closed and open real intervals from a to b. $]a, b]$ and $[a, b[$ are respectively left- and right-open. If a and b are integers, we denote by $[\![a, b]\!] = [a, b] \cap \mathbb{Z}$ the set of integers between a and b (inclusively).

Modular arithmetic For $a, b \in \mathbb{R}$ and a positive real $q > 0$ (all three will usually be integers), we write $a \equiv b \pmod{q}$ if and only if there exists some $k \in \mathbb{Z}$ such that $a = b + kq$. We extend this notation to any set $\mathcal{S}$, writing $a \in \mathcal{S} \pmod{q}$ when there exists some $x \in \mathcal{S}$ such that $a \equiv x \pmod{q}$. We furthermore define the modular-reduction *operator* mod, such that $r = a \bmod q$ is the unique element of $[0, q[$ satisfying $r \equiv a \pmod{q}$. This operator has intermediate precedence between addition and multiplication, i.e. $a + bc \bmod d = a + ((bc) \bmod d)$.

Bit and byte strings We use a specific notation for the set of bits, $\mathbb{b} = \{0, 1\}$, and for the set of 8-bit bytes, $\mathbb{B} = [\![0, 2^8 - 1]\!]$. No specific arithmetic structure is attributed to these sets since it will be specified in each context. Byte strings are identified with bit strings of eight times their length, in little-endian order. On both objects, we use symbol $\|$ for concatenation. When concatenation involves integers, subscripting them with a power of $\mathbb{b}$ or $\mathbb{B}$ disambiguates their length, e.g. $i_{\mathbb{B}^2} \,\|\, j_{\mathbb{B}}$ is a 3-byte string consisting of two bytes for i in little-endian order, followed by one byte for j.

Bitwise logic We will regularly deal with bitwise operations, for which we use the following notations: $\oplus$ denotes bitwise exclusive OR, $\odot$ stands for bitwise AND, and $|$ represents bitwise OR. k-bit left shift $\ll k$ and right shift $\gg k$ assume a binary representation of integers, and respectively multiply and divide by 2^k the affected integer. How negative integers are affected will be mentioned on the spot.

Rounding We denote by $\lfloor\cdot\rfloor$, $\lfloor\cdot\rceil$, and $\lceil\cdot\rceil$, respectively, the flooring, rounding (half-way up), and ceiling functions, such that for any $x \in \mathbb{R}$, $\lfloor x\rfloor$, $\lceil x\rceil$ and $\lfloor x\rceil$ are the only integers satisfying $x - 1 < \lfloor x\rfloor \leqslant x \leqslant \lceil x\rceil < x + 1$ and $x - \frac{1}{2} < \lfloor x\rceil \leqslant x + \frac{1}{2}$.

Polynomial rings Since Kyber and Dilithium perform arithmetic on similar rings, we designate both rings by $R_q = \mathbb{F}_q[X]/(X^{256}+1)$, where the modulus $q \in \{q_\mathrm{K}, q_\mathrm{D}\}$ depends on which scheme is discussed. We also extend this notation to arbitrary m (not necessarily prime) with $R_m = (\mathbb{Z}/m\mathbb{Z})[X]/(X^{256} + 1)$.

We will denote by $\hat{R}_q$ the canonical (as specified by the scheme) Number Theoretic Transform (NTT) domain associated with R_q, and whenever a polynomial $a \in R_q$ is discussed, its NTT-domain counterpart will be denoted by $\hat{a} \in \hat{R}_q$.

Matrices and vectors We use bold uppercase letters, e.g. $\mathbf{A}$, to denote matrices of polynomials, that is, elements of $R_q^{k\times\ell}$ where k and ℓ are the dimensions of the matrix. Similarly, bold lowercase letters, e.g. $\mathbf{a}$, are used for vectors of polynomials, that is, elements of R_q^k for some integer k. Functions defined on integer coefficients are implicitly extended to whole polynomials and to polynomial vectors, by coefficient-wise evaluation.

The component of $\mathbf{A}$ with index (i, j), $i \in [\![0, k-1]\!]$ being the row index and $j \in [\![0, \ell-1]\!]$ being the column index, is denoted by $\mathbf{A}_{i,j}$. Vectors are indexed with a similar notation but a single index.

Algorithm listings In algorithm listings, $=$ denotes variable assignment or equality check depending on the context. An assignment to comma-separated variables is equivalent to several simultaneous assignments, e.g. $a, b = b, a$ exchanges the values of a and b. For any finite set $\mathcal{S}$, we denote by $x \xleftarrow{\$} \mathcal{S}$ sampling the value of x uniformly at random from $\mathcal{S}$. Similarly, if $\mathcal{D}$ is a probability distribution, $x \xleftarrow{\$} \mathcal{D}$ samples the value of x according to $\mathcal{D}$.

Masking For x a masked value over d shares, its shares are denoted by $x^{\mathbf{0}}, \ldots, x^{\boldsymbol{d-1}}$. The set of all shares is denoted by $x^\star$. The symbol without any share index, x, stands for the unmasked value: for Boolean sharing, $x = \bigoplus_{\boldsymbol{i}} x^{\boldsymbol{i}}$, while for arithmetic masking modulo m, $x = (\sum_{\boldsymbol{i}} x^{\boldsymbol{i}}) \bmod m$.

For some integer w and a set $\mathcal{S} \subset \mathbb{F}_2^w$, we denote by $\mathcal{B}^{d+1}(\mathcal{S}) = \{(x^{\mathbf{0}}, \ldots, x^{\boldsymbol{d}}) \in (\mathbb{F}_2^w)^{d+1} \mid \bigoplus_{\boldsymbol{i}} x^{\boldsymbol{i}} \in \mathcal{S}\}$ the set of order-d Boolean sharings of elements of $\mathcal{S}$. Likewise, the set of order-d arithmetic sharings modulo q of elements of $\mathcal{S}$ is denoted by $\mathcal{A}_q^{d+1}(\mathcal{S})$. We will omit the masking order d when it does not matter.

Note that integer exponentiation (seldom used in this work) will never use a boldface exponent, so whether superscripts represent share indexes or exponents will always be clear.

2.2 Lattice-based cryptography

This section introduces lattices as a mathematical object, and presents the main principles of lattice-based cryptography. It also recalls some more general cryptographic notions, in particular in terms of security analysis.

2.2.1 Background on Euclidean lattices

Euclidean lattices are mathematical objects that can be thought of as regular arrangements of points (grids) in space. More formally, we say that a lattice is a discrete additive subgroup of a d-dimensional Euclidean vector space. For an in-depth introduction to the topic, we refer the reader to [Esp19].

We can construct a lattice from a set of n linearly independent vectors $(b_1, \ldots, b_n)$ of $\mathbb{R}^d$, which we will refer to as a *basis* of the lattice. Equation (2.1) gives the set of points of Λ defined by this basis.

$$\Lambda = \left\{ \sum_{i=1}^{n} x_i b_i \;\middle|\; (x_i) \in \mathbb{Z}^n \right\}. \tag{2.1}$$

A lattice has many equivalent bases, all of which have the same number of vectors. These bases are all related through unimodular integer matrices — matrices with integer coefficients and determinant ± 1. We call d the *dimension* of the lattice, and n its *rank*, that corresponds to the dimension of the real span of the lattice, or the number of its basis vectors. Typically, lattices will be full-rank, that is, $d = n$, so that the real span of the lattice is the whole Euclidean space $\mathbb{R}^d$.

2.2.2 Successive minima

Since a lattice is discrete by definition, it contains a shortest nonzero vector[1]. Using the $\|\cdot\|$ notation for the norm (usually Euclidean or infinity norm), we denote the length of this vector by

$$\lambda_1(\Lambda) = \min\{\|x\|,\ x \in \Lambda \setminus \{0\}\}. \tag{2.2}$$

More generally, for $1 \leqslant i \leqslant d$ we define the i^{th} successive minimum of the lattice as the smallest length for which there exist i linearly independent vectors smaller than this length, according to eq. (2.3).

$$\lambda_i(\Lambda) = \min\left\{ \lambda > 0 \;\middle|\; \exists\, v_1, \ldots, v_i \in \Lambda : \operatorname{rank}(v_1, \ldots, v_i) = i \text{ and } \max_j \|v_j\| = \lambda \right\} \tag{2.3}$$

Note that, for lattices of rank 5 or more, the family $v_1, \ldots, v_d$ above is not necessarily a basis of Λ [NS01].

Although Minkowski's theorems [Min10] allow to determine bounds on the successive minima of a lattice from the sole knowledge of one of its bases, finding shortest vectors

[1] Actually, at least two, and possibly more.

in a lattice and determining the exact values of these successive minima are hard problems [Ajt96, Ajt98]: this will be discussed in section 2.2.3. Some algorithms can find relatively short vectors in polynomial time in the dimension of the lattice, however the approximation factor (the ratio of the length of the returned vector to that of the actual shortest vector) in this case grows exponentially with the dimension.

2.2.3 Problems over lattices

The study of lattices gives rise to a wide range of computational problems, some of which have particular importance in cryptography, among other fields. After a brief introduction to the subject of algorithmic problem reduction, we present four lattice problems that are highly relevant to cryptography: the *shortest vector problem* (SVP), the *short integer solution* problem (SIS), as well as the *learning with errors* (LWE) and *learning with rounding* (LWR) problems.

2.2.3.1 Problem reductions

To prove that a given computational problem is intractable, that is, impossible to solve with any physically conceivable amount of computing resources, it is customary to compare it with a reference problem, whose hardness is proved, or at least very widely accepted. A reduction of the reference problem $\mathcal{R}$ to the problem $\mathcal{P}$ under study is an algorithm that can solve $\mathcal{R}$ with the help of an *oracle* for solving $\mathcal{P}$. If the algorithm used in the reduction is efficient (generally defined as running in polynomial time), then we have proved that $\mathcal{P}$ is at least as hard as $\mathcal{R}$. Given that we know or strongly believe $\mathcal{R}$ to be intractable, the intractability of $\mathcal{P}$ is proved by the reduction.

We must note that a classical polynomial-time reduction is a stronger argument than a quantum one, because the former means that $\mathcal{P}$ has at least the same classical and quantum hardness as $\mathcal{R}$. A quantum reduction, instead, only proves that $\mathcal{P}$ is as hard to solve with classical or quantum resources as $\mathcal{R}$ with quantum resources, the mere classical hardness of $\mathcal{R}$ giving no guarantee at all. Given that quantum algorithms are still in their infancy, many problems that are presumed intractable may happen to have an efficient quantum solution, thus making proofs by quantum reduction of this problem irrelevant.

Worst-case to average-case reductions An important property of lattice problems is the existence of worst-case to average-case reductions for a number of them [Ajt96, MR04]. In this situation, the reduction is able to solve efficiently all instances of $\mathcal{R}$ using a solver for average-case instances of $\mathcal{P}$. Such a reduction is very interesting, because it means that an average instance of $\mathcal{P}$ is asymptotically at least as hard as the hardest instances of $\mathcal{R}$.

Yet, such a reduction tells nothing about the best-case hardness of the problem under study. In particular, restricting a hard problem to a specific class of instances might make it easy, if all these instances correspond to best-case instances of the original problem. It is thus essential to study the average-case hardness of the restricted problem as well. This concern is particularly relevant in the case of algebraic lattices, described in section 2.2.4.

Tightness and practical significance of algorithmic reductions Hard problems are of particular importance to cryptography, because they are proven (under some reasonable assumptions) to be intractable: it is thus infeasible for an adversary to break the security of cryptographic schemes based on these problems. However, due to the very definition of hardness, only the asymptotic security of the scheme is proven: starting from a certain size, the problem cannot be solved in reasonable time, and each small increase in the size of the problem increases its difficulty dramatically. Yet, asymptotic hardness tells nothing about this minimum size, only that it exists.

Likewise, the algorithmic reductions that prove the hardness hierarchy of problems are asymptotic and tend to hide polynomial time factors. Consequently, even if a problem $\mathcal{P}$ is provably at least as hard as a reference problem $\mathcal{R}$ through polynomial-time reductions, the hardness of $\mathcal{P}$ may be less than that of $\mathcal{R}$ by a very large polynomial factor. This problem is specially highlighted by Chatterjee *et al.* [CKMS17], who discuss the impact of the tightness of reductions on their practical significance, where the tightness roughly quantifies the runtime and success rate of the problem reduction. The potential issue of loose reductions is exemplified through the analysis of Regev's reduction from the Shortest Independent Vectors Problem (SIVP) to the Learning With Errors problem (LWE) [Reg09], that is shown by Chatterjee *et al.* to be extremely loose. For example, this problem reduction applied to LWE with reasonable parameters in dimension $n = 1024$ has a tightness gap $t \approx 2^{504}$, while current estimates for the quantum cost of solving SVP in the same dimension amount to 2^{293} operations. By assuming that SIVP can be solved in comparable time, Chatterjee *et al.* conclude that the reduction only proves that this instantiation of LWE cannot be solved in fewer than about 2^{-200} operations, which does not bring any hardness guaranty for these parameters.

The cited analysis is cautionary only, and does not invalidate Regev's reduction from an asymptotic aspect. In fact, a more fine-grained analysis of the reduction tightness by Fletcher [Gat18] shows that it gives practical security guarantees for an LWE instantiation in dimension $n = 1460$, which is admittedly large, yet perfectly practical. In conclusion, algorithmic reductions give strong theoretical security guarantees for cryptography, and particularly so for worst-case to average-case reductions, but additional analysis is needed to ascertain the practical security of the precise instantiations of the problems.

2.2.3.2 Shortest vector problem (SVP)

One of the main problems to be solved on lattices is known as the Shortest Vector Problem (SVP). The search problem consists in finding the shortest nonzero vector in a lattice given by one of its bases. The decision problem consists, given a lattice basis and some positive integer $\lambda > 0$, in determining whether there exists a nonzero lattice vector shorter than λ.

The search-SVP exists in an approximate form, in which given an *approximation factor* $\gamma(n)$, the goal is to find a somewhat-short vector $\mathbf{v}$ of the lattice such that $\|\mathbf{v}\| \leqslant \gamma\lambda_1(\Lambda)$. This problem was shown by Khot [Kho04] to be NP-hard for any constant approximation factor, and even for subpolynomial factor $2^{(\log n)^{1/2-\varepsilon}}$ under stronger hypotheses.

The most prominent variant of the decision-SVP is known as GapSVP [Pei09]. Given an approximation factor $\gamma(n)$, a basis $\mathbf{B}$ of an n-dimensional lattice Λ and $d > 0$ a real number, such that either $\lambda_1(\Lambda) \leqslant d$ or $\lambda_1(\Lambda) > d\gamma(n)$, GapSVP_γ consists in deciding which of $\lambda_1(\Lambda) \leqslant d$ or $\lambda_1(\Lambda) > d\gamma(n)$ is true.

Polynomial-time reductions from GapSVP are widely used as a method to put lower bounds on the hardness of many other lattice problems.

2.2.3.3 Short integer solution (SIS) problem

The Short Integer Solution[2] problem (SIS) [MR04] is the problem of finding a solution with small integer coefficients to a set of n linear equations over m variables, each equation being considered modulo q. Consider positive integers m, n and q, together with $\beta > 0$. The Shortest Integer Solution problem $\text{SIS}_{q,m,\beta}$ in dimension n is, given a matrix $\mathbf{A} \in \mathbb{F}_q^{n \times m}$, to find a nonzero vector $\mathbf{x} \in \mathbb{Z}^m \setminus \{\mathbf{0}\}$ such that $\mathbf{Ax} \equiv \mathbf{0}^n \pmod q$ and $\|\mathbf{x}\| \leqslant \beta$.

A variant of the problem, the inhomogeneous SIS, consists in finding a solution to $\mathbf{Ax} \equiv \mathbf{t} \pmod q$ instead, where $\mathbf{t}$ is some known vector of $\mathbb{F}_q^n$.

Note the security parameter n does not define the dimension of the vectors, but the number of equations: the more equations there are, the more difficult it is to satisfy them all. From Micciancio and Regev [MR04], the problem has a solution as soon as $\beta \geqslant \sqrt{m} q^{n/m}$ for the Euclidean norm.

2.2.3.4 Learning with errors (LWE) problem

The Learning with Errors problem (LWE) consists in solving a system of noisy linear equations. Consider positive integers n, m and q, and a distribution χ over $\mathbb{Z}$. Given a matrix $\mathbf{A} \in \mathbb{F}_q^{m \times n}$ sampled uniformly at random, and a vector $\mathbf{t}$ defined by

$$\mathbf{t} = \mathbf{As} + \mathbf{e} \in \mathbb{F}_q^m \,, \tag{2.4}$$

where $\mathbf{s} \xleftarrow{\$} \chi^n$ and $\mathbf{e} \xleftarrow{\$} \chi^m$ are sampled with independent and identically distributed coefficients following the distribution χ, the search LWE problem consists in finding $\mathbf{s}$ [DDGR20].

A decisional variant of LWE is also common, where $\mathbf{t}$ may be sampled either as in eq. (2.4), or uniformly at random from $\mathbb{F}_q^m$, and the task is to distinguish between the two cases.

It was shown by Peikert [Pei09] that solving the search version of LWE, for large modulus $q \geqslant 2^{n/2}$ and χ a Gaussian error distribution of deviation αq, is at least as hard as approximating the worst-case instances of GapSVP to within a factor $\tilde{O}(n/\alpha)$, which is strongly believed to be intractable. Regev [Reg05] also previously provided a quantum reduction from GapSVP to search LWE with the same approximation factor but no restriction on q, and proved that for prime q, the decisional version is as hard as the search version.

[2] Sometimes also Shortest or Smallest Integer Solution

Towards proving the classical hardness of LWE with small modulus, Brakerski *et al.* [BLP+13] first showed that LWE with arbitrary modulus and Gaussian noise is at least as hard as LWE with the next-higher power-of-two modulus and noise rate decreased by some factor $\tilde{O}(n)$. They also showed that LWE with binary secret $\mathbf{s} \xleftarrow{\$} \{0,1\}^n$ with dimension larger than $n \log_2 p$ and modulus q is at least as hard as LWE with uniform secret in dimension n and modulus q. This result was expanded upon by Micciancio [Mic18]. Note, however, that only the *secret* may be binary, as the LWE problem with binary (or otherwise very small) error was found to be solvable in subexponential time by Kirchner and Fouque [KF15].

2.2.3.5 Learning with rounding (LWR)

The LWE problem is based on noisy inner products, where the noise is due to the addition of a small random error. In contrast, Banerjee *et al.* proposed the similar Learning With Rounding (LWR) problem [BPR12], in which errors are due to deterministic rounding. The decision-LWR problem was shown to be at least as hard as the decision-LWE problem with the same dimension and modulus, as soon as q is superpolynomial in n. It also seems to be exponentially hard when the amplitude p of the rounding error is polynomial in n and $\sqrt{n} \leqslant q/p \in \mathbb{N}$ [BPR12].

2.2.4 Algebraic or Structured lattices

In the above, we introduced Euclidean lattices, used by *unstructured* lattice-based cryptography schemes. There exist other kinds of lattices, having some additional algebraic structure. They are particularly relevant for cryptography because, while their structure allows for more efficient representation and manipulation of their elements, it does not seem to negatively affect the hardness of lattice problems on them if the structure is sufficiently light.

In particular, ideal lattices have a ring structure, that is, their elements can be represented as polynomials. They have also been known as cyclic lattices, and have been studied for quite a long time [HPS98, Mic02, PM19]. Intermediate between Euclidean and ideal lattices are module lattices, whose elements can be represented as vectors of polynomials, and which have the structure of a module.

Problems over algebraic lattices The lattice problems defined in section 2.2.3 also exist in versions restricted to ideal lattices and module lattices. However, the average-case hardness of these problems does not necessarily hold for these restrictions [Cam17]. For instance, the *decision* version of the SVP, GapSVP_γ, is easy on ideal lattices of dimension n for approximation factor $\gamma \leqslant \sqrt{n}$ [LPR10, Lemma 2.9], while the same problem is presumed to be hard on arbitrary lattices, for any polynomially growing $\gamma(n)$.

However, for reasonable instantiations, it is well established that the *search* LWE problem over rings, the *decision* LWE problem over modules, and the SIS problem over modules are similarly hard to their counterparts on plain lattices [LPR10, AD17, PP19a, LS15, BJRW23, SSTX09].

2.3 CRYSTALS-Kyber key-encapsulation mechanism and ML-KEM

CRYSTALS-Kyber (Kyber for short) is a key-encapsulation mechanism based on the LWE problem, that was submitted to NIST PQC standardization [SAB+20]. After progressing through the first three rounds of the standardization process, it was the only KEM selected for immediate standardization at the end of the third round.

In this section, we will present the scheme in accordance with version 3.02 of its specification [ABD+21], and will briefly mention the changes that NIST incorporated into the standardized algorithm, named ML-KEM (Module Lattice Key Encapsulation Mechanism). However, these changes are mentioned for reference only and will generally not be considered in the rest of this document.

2.3.1 Specification

Being based on the Module LWE problem, Kyber performs most of its arithmetic in a fixed polynomial ring $R_{q_\mathrm{K}} = \mathbb{F}_{q_\mathrm{K}}[X]/(X^n + 1)$, where the degree of the reducing polynomial is $n = 256$ and the integer modulus is prime $q_\mathrm{K} = 3329$. Kyber involves matrices and vectors of elements from this ring; the security level of the scheme is chosen by varying the dimension k of matrices and vectors among 2, 3, and 4. Due to the dimension kn of the resulting lattice, these three security levels of Kyber are called Kyber-512, Kyber-768, and Kyber-1024, which have approximately the same classical and quantum security as AES-128, AES-192 and AES-256 respectively.

2.3.1.1 Polynomial arithmetic

Kyber gets most of its performance from the efficiency of its basic arithmetic. Its scalars are polynomials taken from the ring $R_{q_\mathrm{K}} = \mathbb{F}_{q_\mathrm{K}}[X]/(X^n + 1)$, where $n = 256$. Since n divides $q_\mathrm{K} - 1$, polynomial multiplication can be computed with complexity $O(n \log(n))$ using the Number-Theoretic Transform (NTT), which is the equivalent of the discrete Fourier transform for finite fields [AB74]. Given two polynomials $a, b \in R_{q_\mathrm{K}}$, their product is computed according to eq. (2.5)

$$a \times b = \mathrm{NTT}^{-1}(\mathrm{NTT}(a) \circ \mathrm{NTT}(b)) \,, \tag{2.5}$$

where NTT and NTT^{-1} respectively denote the forward and inverse NTT, and $\circ$ denotes point-wise multiplication, which has linear complexity.

Since n divides $q_\mathrm{K} - 1$ but $2n$ does not, the reducing polynomial of R_{q_K} is not *split* (i.e., it cannot be decomposed into a product of linear terms), so Kyber NTT is not *complete*: it decomposes a polynomial into $n/2$ linear terms instead of n constant terms. Consequently, point-wise multiplication computes the term-by-term product of 128 degree-1 polynomials rather than the product of 256 integers. However, this has little influence on the arithmetic performance of Kyber.

Our generic implementations of forward and inverse NTT with L layers are respectively shown in algorithm 2.1 and algorithm 2.2[3]. These operate in place, which means that the polynomial operand is overwritten with the result of the NTT transformation, without using any additional storage during the operation. In the case of Kyber, the number of layers is $L = 7$. Both of these algorithms rely on a precomputed table of the $(2^L)^{\text{th}}$ roots modulo q of -1, denoted $(\zeta_z)_{1 \leqslant z < 2^L}$ and sometimes called *twiddle factors*. More specifically, for ζ a primitive $(2^L)^{\text{th}}$ root of -1 and $1 \leqslant z < 2^L$, ζ_z is defined by

$$\zeta_z = \zeta^{\text{br}_L(z)} \bmod q\,, \tag{2.6}$$

where function $\text{br}_L(z) = \sum_{i=0}^{L-1} 2^{L-1-i}((z \gg i) \bmod 2)$ bit-reverses an L-bit integer by swapping its bits at positions i and $L-1-i$. At layer λ of the NTT (in order or in reverse depending on the NTT direction), all the coefficients of the polynomial are taken in pairs, each pair involving two indexes that differ by $n/2^{\lambda+1}$. A *butterfly* operation is then applied to the pair, giving out the new value of the pair after a few modular additions, subtractions, and a single multiplication by the appropriate twiddle factor.

Algorithm 2.1: In-place Forward NTT

Parameter: Dimension $n = 2^\nu$ of the polynomial ring R_q, number $L < \nu$ of NTT layers
Parameter: Roots of unity $(\zeta_z)_{1 \leqslant z < 2^L}$
Input and output: Polynomial $a \in R_q$

```
for λ = 0 to L − 1 do
    // Iterate over all coefficient indices whose (ν − 1 − λ)^th bit is cleared
    for i ∈ ⟦0, n − 1⟧ such that (i ≫ (ν − 1 − λ)) ≡ 0 (mod 2) do
        z = 2^λ + (i ≫ (ν − λ))
        j = i + 2^(ν−1−λ)
        a[i], a[j] = (a[i] + a[j]ζ_z) mod q, (a[i] − a[j]ζ_z) mod q
    end
end
```

Kyber point-wise multiplication reuses this table or roots of -1, as shown in algorithm 2.3. The adoption of Karatsuba multiplication allows to efficiently compute the product of two linear polynomials modulo $X^2 \pm \zeta_z$, following Xing and Li [XL21]. This method uses four integer multiplications and five additions (or subtractions), instead of the five multiplications and two additions used by schoolbook multiplication.

2.3.1.2 Random sampling

The LWE framework involves sampling secrets and errors from small distributions centered around zero: the original scheme of Regev [Reg05], as well as the Lyubashevsky-

[3]Most implementations available in the literature, e.g. [POG15, Appendix A.1], use three nested loops for the computation of the NTT: the outer one iterates over NTT layers, the middle one over contiguous groups of coefficients, and the innermost one over coefficients within the groups. We merge the two inner loops for convenience, so that the resulting inner loop has the same number of iterations $(n/2)$ at all iterations of the outer loop.

Algorithm 2.2: In-place Inverse NTT

Parameter: Dimension $n = 2^\nu$ of the polynomial ring R_q, number $L < \nu$ of NTT layers
Parameter: Roots of unity $(\zeta_z)_{1 \leqslant z < 2^L}$
Input and output: Polynomial $a \in R_q$

for $\lambda = L - 1$ to 0 **do**
  // Iterate over all coefficient indices whose $(\nu - 1 - \lambda)^{th}$ bit is cleared
  for $i \in [\![0, n-1]\!]$ such that $(i \gg (\nu - 1 - \lambda)) \equiv 0 \pmod 2$ **do**
    $z = 2^\lambda + ((n - 1 - i) \gg (\nu - \lambda))$
    $j = i + 2^{\nu - 1 - \lambda}$
    $a[i], a[j] = (a[j] + a[i]) \bmod q, (a[j] - a[i])\zeta_z \bmod q$
  end
end
for $i \in [\![0, n-1]\!]$ **do**
  $a[i] = a[i] \times n^{-1} \bmod q$
end

Algorithm 2.3: Kyber point-wise multiplication (operator $\circ$), from [XL21]

Parameter: Dimension $n = 2^\nu$ of the polynomial ring R_q
Parameter: Roots of unity $(\zeta_z)_{1 \leqslant z < 2^L}$
Input: NTT-domain polynomials $a = (\hat{a}[0], \ldots, \hat{a}[n-1]), b = (\hat{b}[0], \ldots, \hat{b}[n-1]) \in \hat{R}_{q_\mathrm{K}}$
Output: NTT-domain polynomial $c = (\hat{c}[0], \ldots, \hat{c}[n-1]) \in \hat{R}_{q_\mathrm{K}}$

for $i \in [\![0, n/2 - 1]\!]$ **do**
  $z = n/4 + \lfloor i/2 \rfloor$
  $x_{00} = a[2i] \times b[2i] \bmod q_\mathrm{K}$
  $x_{11} = a[2i+1] \times b[2i+1] \bmod q_\mathrm{K}$
  $c[2i+1] = ((a[2i] + a[2i+1]) \times (b[2i] + b[2i+1]) - x_{00} - x_{11}) \bmod q_\mathrm{K}$
  $c[2i] = (x_{00} + (-1)^i x_{11} \times \zeta_z) \bmod q_\mathrm{K}$
end

Peikert-Regev (LPR) scheme [LPR10], involved discrete Gaussian distributions. However, these distributions being difficult to sample from, in particular when side-channel resistance is needed, Kyber replaces them with centered binomial sampling. Algorithm 2.4 details how binomial sampling in $[\![-\eta, \eta]\!]$ is done: given a uniformly random string of n elements of 2η bits, the bits of the first and last halves of each element are separately summed; subtracting these two sums then gives a sample with the desired distribution.

Another type of random sampling which is required is uniform sampling in R_{q_K}, which consists in sampling a list of n integers independently and uniformly distributed between 0 and $q_\mathrm{K} - 1$. Since this range is not a power of two, rejection sampling is employed: input randomness is taken 12 bits at a time, and compared with the bound q_K: values strictly smaller than the bound are kept as a valid sample, while larger values are rejected and the corresponding randomness is not used. This procedure is formalized by the Parse function (algorithm 2.5). Note that uniform sampling gives a polynomial directly in the NTT domain to save a later domain conversion.

Algorithm 2.4: Kyber CBD function: centered binomial sampling

Parameter: Half-width η of binomial distribution
Input: n-length sequence of 2η-bit values, $(\beta_i)_{0 \leqslant i < n} \in (\mathbb{b}^{2\eta})^n$
Output: Polynomial $f \in R_{q_\mathrm{K}}$ with binomially distributed coefficients

for $i = 0$ to $n - 1$ **do**
  $p = \sum_{j=0}^{\eta-1} \beta_{2i\eta+j}$
  $n = \sum_{j=0}^{\eta-1} \beta_{(2i+1)\eta+j}$
  $f[i] = p - n$
end
return f

Algorithm 2.5: Kyber Parse function: expansion of public polynomials

Input: Byte stream $(b_i) \in \mathbb{B}^*$
Output: NTT-domain polynomial $\hat{a} = (\hat{a}[0], \ldots, \hat{a}[n-1]) \in \hat{R}_{q_\mathrm{K}}$

$i = 0$
$j = 0$
while $j < n$ **do**
  if $i \equiv 0 \pmod 2$ **then** $d = (b_{3i/2+1} \bmod 2^4 \ll 4) \mid b_{3i}$
  else $d = (b_{3(i-1)/2+2} \ll 4) \mid (b_{3(i-1)/2+1} \gg 4)$
  if $d < q$ **then**
    $\hat{a}[j] = d$
    $j = j + 1$
  end
  $i = i + 1$
end
return $\hat{a}$

All random sampling in Kyber uses as source of randomness the digest of a hash function or extendable-output function (XOF) from the Keccak family [BDPA13]: SHA3-256, SHA3-512, SHAKE128 or SHAKE256[4].

2.3.1.3 Ciphertext compression

To reduce its bandwidth usage, Kyber employs ciphertext compression, whereby the coefficients of the ciphertext, initially comprised between 0 and $q_\mathrm{K} - 1$, are rescaled down to a lower power-of-two range. The two parts $\mathbf{u}$ and v of the ciphertext have their coefficients truncated to $d_\mathrm{u} \in \{10, 11\}$ and $d_\mathrm{v} \in \{4, 5\}$, thereby reducing the size of the ciphertexts by 19 % to 33 % depending on the security level. Practically, the compression of each coefficient to $d \leqslant 11$ bits is accomplished through rounded multiplication by $2^d/q_\mathrm{K}$, as accomplished by the Compress function shown in eq. (2.7). Decompression of a coefficient is likewise performed through rounded multiplication by the inverse ratio, as

[4] A notable consequence of using a pseudorandom XOF instead of true randomness for the expansion of matrix $\mathbf{A}$ is that this operation provably terminates for any value of the seed [BS23]. The proved bound on the number of *Squeeze* operations for Parse, however, is a practically meaningless 3×2^{1600}.

done by Decompress (eq. (2.8)). Note that for decompression, no modular reduction is required after the rounded multiplication since the multiplicative factor is greater than one.

$$\text{Compress}_d(x \in [\![0, q_\text{K} - 1]\!]) = \left\lfloor x \frac{2^d}{q_\text{K}} \right\rceil \bmod 2^d \in [\![0, 2^d - 1]\!]\,, \tag{2.7}$$

$$\text{Decompress}_d(y \in [\![0, 2^d - 1]\!]) = \left\lfloor y \frac{q_\text{K}}{2^d} \right\rceil \qquad \in [\![0, q_\text{K} - 1]\!]\,. \tag{2.8}$$

It is important to note that this compression is *lossy*, which means that in general, $\text{Decompress}_d(\text{Compress}_d(x)) \neq x$. In the other direction, instead, it is easy to check that equality $\text{Compress}_d(\text{Decompress}_d(x)) = x$ holds for all x.

2.3.1.4 Ind-CPA public-key encryption

Kyber is built in two layers to achieve its security properties. Internally, it uses a public-key encryption (PKE) scheme that only has the property of indistinguishability under a chosen-plaintext attack (Ind-CPA); on top of it, an additional layer gives it indistinguishability under a chosen-ciphertext attack (Ind-CCA). Each layer is composed of three functions: key generation, encryption or encapsulation, and decryption or decapsulation.

Ind-CPA key generation is shown in algorithm 2.6. It starts by generating a public matrix, $\hat{\mathbf{A}} \in R_{q_\text{K}}^{k \times k}$, with polynomials having uniformly random coefficients. This matrix is sampled directly in the NTT domain. Then, a secret vector, $\mathbf{s} \in R_{q_\text{K}}^k$ and an error vector $\mathbf{e} \in R_{q_\text{K}}^k$ are sampled with binomially distributed coefficients in the range $[\![-\eta_1, \eta_1]\!]$. After passing these two vectors into the NTT domain, a set of k LWE samples is computed through $\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$. This vector $\hat{\mathbf{t}}$, together with the seed ρ of matrix $\hat{\mathbf{A}}$ that generates them, constitutes the public key of the Ind-CPA and Ind-CCA schemes, while the secret key is made up of vector $\hat{\mathbf{s}}$, stored in the NTT domain.

Ind-CPA encryption of a 256-bit message $m \in R_2$ under public key $(\hat{\mathbf{t}}, \rho)$ can be seen in algorithm 2.7. After expanding the public matrix from its seed, two vectors $\mathbf{r}, \mathbf{e}_1 \in R_{q_\text{K}}^k$ (a one-time secret and an error) are sampled with binomially distributed coefficients in $[\![-\eta_1, \eta_1]\!]$, and an additional noise polynomial, this time with coefficients in $[\![-\eta_2, \eta_2]\!]$. LWE samples are generated from the public matrix, and the noise vectors as $\mathbf{u} = \text{NTT}^{-1}(\hat{\mathbf{A}}^\top \circ \hat{\mathbf{r}}) + \mathbf{e}_1$. Then, a mask polynomial is generated as the dot product of the one-time secret $\mathbf{r}$ with the public LWE samples $\mathbf{t}$, obscured with noise e_2; the result of this operation is used to mask message m, previously inflated so that its bits are mapped to 0 or $\lfloor q/2 \rceil$. This masked message is called v. As a last step, the two parts of the ciphertext, $\mathbf{u}$ and v, are separately compressed before transmission.

Ind-CPA decryption decrypts the received ciphertext $ct = (\mathbf{c}_1, c_2) \in R_{2^{d_u}}^k \times R_{2^{d_v}}$ using the private key, following the steps of algorithm 2.8. Before anything, the two parts of the ciphertext are decompressed, giving $\mathbf{u}'$ and v', which are close to the $\mathbf{u}$ and v computed during encryption, but not necessarily equal to them due to the noise added by ciphertext compression. Then, the dot-product of the secret vector with ciphertext

Algorithm 2.6: Kyber Ind-CPA key generation

Output: Private key $\hat{\mathbf{s}} \in R_{q_\mathrm{K}}^k$, public key $pk = (\hat{\mathbf{t}}, \rho) \in R_{q_\mathrm{K}}^k \times \mathbb{B}^{32}$

$d \xleftarrow{\$} \mathbb{B}^{32}$
$(\rho, \sigma) = \text{SHA3-512}(d)$ // $\rho, \sigma \in \mathbb{B}^{32}$
for $i = 0$ **to** $k - 1$ **do**
  for $j = 0$ **to** $k - 1$ **do**
    $\hat{\mathbf{A}}_{i,j} = \text{Parse}(\text{SHAKE128}(\rho \parallel i_\mathbb{B} \parallel j_\mathbb{B}))$ // *i and j are considered as 8-bit integers*
  end
end
for $i = 0$ **to** $k - 1$ **do**
  $\mathbf{s}_i = \text{CBD}_{\eta_1}(\text{SHAKE256}(\sigma \parallel i_\mathbb{B}))$
end
for $i = 0$ **to** $k - 1$ **do**
  $\mathbf{e}_i = \text{CBD}_{\eta_1}(\text{SHAKE256}(\sigma \parallel (i + k)_\mathbb{B}))$
end
$\hat{\mathbf{s}} = \text{NTT}(\mathbf{s})$
$\hat{\mathbf{e}} = \text{NTT}(\mathbf{e})$
$\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$
$pk = (\hat{\mathbf{t}}, \rho)$
return $(\hat{\mathbf{s}}, pk)$

Algorithm 2.7: Kyber Ind-CPA encryption

Input: Public key $pk = (\hat{\mathbf{t}}, \rho) \in R_{q_\mathrm{K}}^k \times \mathbb{B}^{32}$
Input: Message $m \in R_2$, random coins $r \in \mathbb{B}^{32}$
Output: Ciphertext $ct = (\mathbf{c}_1, c_2) \in R_{2^{d_u}}^k \times R_{2^{d_v}}$

for $i = 0$ **to** $k - 1$ **do**
  for $j = 0$ **to** $k - 1$ **do**
    $\hat{\mathbf{A}}_{i,j} = \text{Parse}(\text{SHAKE128}(\rho \parallel i_\mathbb{B} \parallel j_\mathbb{B}))$
  end
end
for $i = 0$ **to** $k - 1$ **do**
  $\mathbf{r}_i = \text{CBD}_{\eta_1}(\text{SHAKE256}(\sigma \parallel i_\mathbb{B}))$
end
for $i = 0$ **to** $k - 1$ **do**
  $(\mathbf{e}_1)_i = \text{CBD}_{\eta_1}(\text{SHAKE256}(\sigma \parallel (i + k)_\mathbb{B}))$
end
$e_2 = \text{CBD}_{\eta_2}(\text{SHAKE256}(r \parallel (2k)_\mathbb{B}))$
$\hat{\mathbf{r}} = \text{NTT}(\mathbf{r})$
$\mathbf{u} = \text{NTT}^{-1}(\hat{\mathbf{A}}^\top \circ \hat{\mathbf{r}}) + \mathbf{e}_1$
$v = \text{NTT}^{-1}(\hat{\mathbf{t}}^\top \circ \hat{\mathbf{r}}) + e_2 + \text{Decompress}_1(m)$
$\mathbf{c}_1 = \text{Compress}_{d_u}(\mathbf{u})$
$c_2 = \text{Compress}_{d_v}(\mathbf{v})$
return $ct = (c_1, c_2)$

part $\mathbf{u}$ is subtracted from v, giving polynomial M'. This polynomial is decoded to one bit per coefficient through rounded division by $q_{\mathrm{K}}/2$, outputting m', which should be equal to the original plaintext m unless a decryption failure has occurred.

Algorithm 2.8: Kyber Ind-CPA decryption

Input: Private key $\hat{\mathbf{s}} \in R_{q_{\mathrm{K}}}^k$
Input: Ciphertext $ct = (\mathbf{c}_1, c_2) \in R_{2^{d_u}}^k \times R_{2^{d_v}}$
Output: Message $m' \in R_2$
1 $\mathbf{u}' = \mathrm{Decompress}_{d_u}(\mathbf{c}_1)$
2 $v' = \mathrm{Decompress}_{d_v}(c_2)$
3 $M' = v' - \mathrm{NTT}^{-1}(\hat{\mathbf{s}}^{\mathsf{T}} \circ \mathrm{NTT}(\mathbf{u}'))$
4 $m' = \mathrm{Compress}_1(M')$
5 **return** m'

2.3.1.5 Ind-CCA key-encapsulation mechanism

Since indistinguishability under a chosen plaintext attack is not sufficient for security when a given key pair is used several times, the preceding scheme is integrated into a construction akin to the Fujisaki-Okamoto transformation [FO99, FO13], with the improvements by Hoffstein *et al.* [HHK17] that allow for the possibility of decryption failures. The main change introduced by this *Ind-CCA transform* is that encryption derives all of its randomness deterministically from the plaintext, and that decryption is immediately followed by re-encryption of the plaintext using the same randomness, to check whether the ciphertext is genuine. Furthermore, since a key-encapsulation mechanism is described instead of plain public-key encryption, the plaintext shared through the underlying PKE is fed into a key-derivation function to make key material from it.

The Ind-CCA key-generation algorithm, shown in algorithm 2.9, internally performs Ind-CPA key generation. It also samples a uniformly random 256-bit string, z, which is only used to perform implicit rejection in case of a decapsulation failure. It embeds this element into the Ind-CCA private key, which also contains the whole Ind-CPA key pair together with a hash of the public key. The public key, instead, is left unchanged.

Algorithm 2.9: Kyber Ind-CCA key generation

Output: Private key $sk = (\hat{\mathbf{s}}, pk, pkh, z) \in R_q^k \times (R_q^k \times \mathbb{B}^{32}) \times \mathbb{B}^{32} \times \mathbb{B}^{32}$
Output: Public key $pk = (\hat{\mathbf{t}}, \rho) \in R_q^k \times \mathbb{B}^{32}$
1 $z \xleftarrow{\$} \mathbb{B}^{32}$
2 $(\hat{\mathbf{s}}, pk) = \mathrm{KyberCPAKeyGen}()$
3 $sk = (\hat{\mathbf{s}}, pk, \mathrm{SHA3\text{-}256}(pk), z)$
4 **return** (sk, pk)

Encapsulation (algorithm 2.10) is built upon the encryption of a randomly-sampled[5] 256-bit message m, with the random coins r deterministically derived from m through a hash. The same hash call also provides a precursor for the shared secret $\bar{K}$. The ciphertext output by Ind-CPA encryption is sent unchanged to the other party, while a shared key is computed by evaluating a key-derivation function on the secret precursor $\bar{K}$ and the ciphertext c.

Algorithm 2.10: Kyber Ind-CCA encapsulation

Input: Public key $pk = (\hat{\mathbf{t}}, \rho) \in R_q^k \times \mathbb{B}^{32}$
Output: Ciphertext $ct = (\mathbf{c}_1, c_2) \in R_{2^{d_u}}^k \times R_{2^{d_v}}$
Output: Shared key $K \in \mathbb{B}^*$

$m_{\text{pre}} \xleftarrow{\$} \mathbb{B}^{32}$
$m = \text{SHA3-256}(m_{\text{pre}})$ *// Hash the system randomness*
$(\bar{K}, r) = \text{SHA3-512}(m \,\|\, \text{SHA3-256}(pk))$
$ct = \text{KyberCPAEncrypt}(pk, m, r)$
$K = \text{SHAKE256}(\bar{K} \,\|\, \text{SHA3-256}(ct))$
return (ct, K)

The important difference between the Ind-CPA and the Ind-CCA schemes is found in the decapsulation procedure (algorithm 2.11). At a high level, this procedure decrypts the received ciphertext, then re-encrypts it with random coins derived in the same way as during encapsulation. The re-encrypted ciphertext ct' obtained in this way is then compared with the received ciphertext ct: if they match, the shared key is computed as during encapsulation; if, instead, the comparison fails, a pseudorandom shared key is deterministically derived from the received ciphertext ct and part z of the private key. This check and the specific way of deriving the shared key give the scheme its Ind-CCA security [HHK17].

Algorithm 2.11: Kyber Ind-CCA decapsulation

Input: Ciphertext $ct = (\mathbf{c}_1, c_2) \in R_{2^{d_u}}^k \times R_{2^{d_v}}$
Input: Private key $sk = (\hat{\mathbf{s}}, pk, pkh, z) \in R_q^k \times (R_q^k \times \mathbb{B}^{32}) \times \mathbb{B}^{32} \times \mathbb{B}^{32}$
Output: Shared key $K \in \mathbb{B}^*$

$m' = \text{KyberCPADecrypt}(\hat{\mathbf{s}}, ct)$
$(\bar{K}', r') = \text{SHA3-512}(m' \,\|\, pkh)$
$ct' = \text{KyberCPAEncrypt}(pk, m', r')$
if $ct = ct'$ **then** *// N.B. The truth value of the comparison is sensitive*
  $K' = \text{SHAKE256}(\bar{K}' \,\|\, \text{SHA3-256}(ct))$
else
  $K' = \text{SHAKE256}(z \,\|\, \text{SHA3-256}(ct))$
end
return K'

[5] By the designers' specification, m is not directly taken from the system random number generator: it is passed through a hash, to ensure that the system-issued randomness is not revealed.

2.3.1.6 Parameter sets

The parameters of the scheme depending on the security level are presented in table 2.1. The security level of the scheme is selected by changing the rank k of the module, and increasing the amount of noise η_1 for the lowest security level. The size of compressed coefficients also varies: the highest security level keeps one more bit per coefficient, in order to keep the probability of decryption failure negligible.

2.3.1.7 Decryption failures

In order to have an efficient scheme, the designers of Kyber chose parameters that make decryption failures possible, although very rare [ABD+21]. Under the assumption that the failures of individual bits are independent, the authors analyzed the probability of decryption failure, shown as δ in table 2.2. However, this assumption was proven significantly incorrect by D'Anvers *et al.* [DVV19]. Consequently, Fang *et al.* [FWZ22] performed a more thorough analysis of this failure rate by sampling Kyber key pairs and computing the actual probability of decryption failure for each of these keys. They showed that different key pairs were associated with widely different failure probabilities. Their results, also reported in table 2.2, furthermore suggest that δ is an overestimate of the actual failure probability for Kyber-512 and Kyber-768, but possibly underestimated for a significant proportion of Kyber-1024 keys[6].

2.3.2 Changes from Kyber to ML-KEM

As part of the standardization of Kyber into ML-KEM by NIST [FIP24a], several changes have been made to the scheme. We here list these changes and briefly mention their implications. Despite some requests for removing the lowest-security parameter set, Kyber-512, all three security levels are kept for standardization. The naming of the three security levels is analogous to that of Kyber: ML-KEM-512, ML-KEM-768 and ML-KEM-1024, which use exactly the same parameters as Kyber-512, Kyber-768 and Kyber-1024 respectively (table 2.1).

2.3.2.1 No additional hashing of system randomness

The specification of Kyber encapsulation involves encrypting a message that is generated as the hash of system randomness (line 2 of algorithm 2.10). This hashing step is supposed to protect against a specific hypothetical weakness of the PRNG: that getting access to its raw output may leak partial or total information on its state. This can happen with a badly designed, or backdoored PRNG [SF07, CNE+14]. While the opinion of the community on this matter has been split, with proponents on both sides, this provision was deemed unnecessary in ML-KEM since a FIPS-approved PRNG is required: these do not have (publicly known) weaknesses of this kind [BDK+23].

[6] None of these two conclusions are certain since Fang *et al.* are only able to compute bounds for the failure probability, with a multiplicative gap of about 2^{40} between the lower and upper bounds, and since they base this analysis on a small sample of 300 randomly-selected keys, which almost certainly does not contain the extreme cases.

Table 2.1: Recommended parameters for Kyber

Param.	Value			Description
n	256			Dimension of the base ring over $\mathbb{F}_q$, chosen equal to the number of bits of the secret to be encapsulated.
q_{K}	3329			Characteristic of the base ring, chosen as the smallest prime satisfying $n \mid (q_{\mathrm{K}} - 1)$ and allowing for negligible failure probability.
	Value for Kyber-			
	512	768	1024	
k	2	3	4	Rank of the module over the base ring. Set such that the dimension of the resulting lattice gives the desired security level.
η_1	3	2	2	Half-width of the centered binomial distribution used to sample $\mathbf{s}$ and $\mathbf{e}$ in key generation, and $\mathbf{r}$ in encryption.
η_2	2	2	2	Half-width of the centered binomial distribution used to sample $\mathbf{e}_1$ and e_2 in encryption.
d_{u}	10	10	11	Compressed bits per coefficient for ciphertext part $\mathbf{u}$. Minimizes ciphertext size while keeping decryption-failure probability low.
d_{v}	4	4	5	Compressed bits per coefficient for ciphertext part v. Minimizes ciphertext size while keeping decryption-failure probability low.

Table 2.2: Kyber decryption-failure probability

	Kyber-512	Kyber-768	Kyber-1024
Specified failure probability from [ABD+21]			
δ	2^{-139}	2^{-164}	2^{-174}
Failure-probability upper bound over a sample of 300 keys [FWZ22]			
Maximum	2^{-148}	2^{-166}	2^{-142}
Median	2^{-186}	2^{-222}	2^{-232}
Geometric mean	2^{-188}	2^{-233}	2^{-255}
Minimum	2^{-221}	2^{-366}	2^{-592}

2.3.2.2 Allowing public-key expansion to fail when rejection sampling lasts too long

Since Kyber uses rejection sampling to expand public matrix $\mathbf{A}$ from its seed (algorithm 2.5), the duration of this operation is unpredictable, as well as the number of SHAKE128 output blocks requested. While a strict implementation of public-key expansion may loop for a long time with negligible but nonzero probability (except for the meaninglessly large bound discussed in § 2.3.1.2, footnote 4), NIST allows (but strongly recommends against it) to abort the execution of the Parse function if it exceeds a certain number of iterations. The minimum upper bound allowed by NIST corresponds to processing 560 input words of 12 bits, which are sufficient to sample the polynomial except with a probability of 2^{-261} [FIP24a].

2.3.2.3 Domain separation between security levels

During Ind-CPA key generation, the public and private seeds ρ and σ are generated as two parts of the digest of a single 32-byte seed d (line 2 of algorithm 2.6). This can be a security issue if seed d is reused for key pairs corresponding to different security levels of Kyber, since the public matrix $\mathbf{A}$ for security level 3 will be a sub-matrix of the level-5 public matrix, and the level-3 secret vector $\hat{\mathbf{s}}$ will be a sub-vector of the level-5 one. Thus, knowing the key pairs generated from d for both level 3 and level 5 seems to reduce the strength of both to security level 3, as detailed in section 2.A. This does not directly apply from level 1 to levels either 3 or 5, due to the different value of the η parameter that changes the expansion of the secret vector, but the concern is still valid in this case. To avoid this shortcoming, NIST implemented domain separation between the security levels of ML-KEM by replacing line 2 of algorithm 2.6 with $(\rho, \sigma) = \text{SHA3-512}(d \parallel k_{\mathbb{B}})$, where k is the dimension of the vectors specifying the security level, expressed as an 8-bit byte: with this change, the key pairs with different security levels generated from a single seed are completely unrelated [Per24, FIP24a].

2.3.2.4 Changes to the CCA transform

The round-3 specification of Kyber uses a tweaked CCA transform, that loosely corresponds to the $\mathsf{U}^{\not\perp}$ transformation of Hofheinz *et al.* [HHK17], but which derives the shared secret through additional hashing both on the ciphertext side and on the message side. We denote by K_{KYBER} the shared secret derived in Kyber for a successful decapsulation:

$$K_{\text{KYBER}} = \text{SHAKE256}\Big(\text{G}_{256}(m' \parallel \text{SHA3-256}(pk)) \parallel \text{SHA3-256}(ct)\Big)\,,$$

where G_{256} denotes the 256-bit prefix of SHA3-512. In contrast, a direct application of the $\mathsf{U}^{\not\perp}$ transformation would compute a single digest directly from the message and the ciphertext, giving shared secret K_{HHK17} according to

$$K_{\text{HHK17}} = \text{SHAKE256}(m' \parallel ct)\,.$$

This deviation from proved CCA transformations, analyzed by Grubbs *et al.* [GMP22], brings some barriers to tight security proofs, and is not documented to bring notable advantages on other aspects. Maram and Xagawa later gave a tight security proof of the Ind-CCA security of Kyber in the QROM, but with some assumed restrictions on the strength of an attacker against its Ind-CPA component [MX23]. Bernstein however notes that, since the underlying assumptions are not backed by solid results from the literature, this conclusion must be considered with care [Ber23a].

To avoid these shortcomings, ML-KEM no longer includes an additional hashing step on m', and sends it directly to the key derivation function, computing shared secret $K_{\text{ML-KEM}}$ as

$$K_{\text{ML-KEM}} = \text{SHAKE256}(m' \parallel \text{SHA3-256}(pk))\,.$$

To make the key derivation *contributory*, i.e. dependent on data from both communicating parties, the hash of the public key is also used in key derivation, in place of the hash of

the ciphertext. Indeed, while the ciphertext hash was supposed to already provide this property, Grubbs *et al.* [GMP22] note that it may not be the case since there may be some messages that encrypt to the same ciphertext under two different public keys.

2.3.2.5 Fixed size of the shared secret

To support establishing a shared secret of arbitrary length, Kyber specifies its key derivation as an XOF, SHAKE256, with the length of the digest freely chosen by the application. The standardized version, instead, fixes the length of the shared secret to 256 bits, and applications needing a different length must perform an additional key-derivation step to convert this fixed-length secret into the suitable amount of key material. The same XOF is still used, so this change only impacts applications needing long shared secrets, which must be obtained through an additional key-derivation function.

2.3.2.6 Validity check of the public key

ML-KEM introduces input validation for the encapsulation and decapsulation operations. Apart from checking that all received byte strings have exactly the required length, it mandates a modulus check on the public-key component $\mathbf{t}$, which must only contain integers in the range $[\![0, q_{\mathrm{K}} - 1]\!]$.

2.3.3 Security

The security of Kyber is rooted both in a theoretical, asymptotic analysis based on the assumed security of the Module-LWE problem, and an estimation of the effort needed to break it in practice, considering the cost of known attacks. We will here give a brief overview of its security arguments.

2.3.3.1 Reduction from MLWE in the Random Oracle Model

Assuming the hardness of decisional Module-LWE, the Ind-CPA security of the underlying PKE of Kyber is straightforward to prove, since the public keys and ciphertexts are pseudorandom under this assumption [SAB+20]. Then, the designers of Kyber show that the CCA transform of the scheme gives it a tight Ind-CPA security proof, if the hash functions are modeled as classically accessible random oracles. As mentioned in § 2.3.2.4, the CCA transform is not a direct application of the transform of Hofheinz *et al.*, which tends to complicate the proof but still makes it feasible.

2.3.3.2 Reduction from MLWE in the Quantum Random Oracle Model

When modeling the hash functions as quantumly accessible random oracles instead, the reduction from MLWE still holds, but it is no longer tight. It is possible to get a tight reduction in this context, under the additional assumption that a deterministic version of the PKE of Kyber (deriving the randomness used by encryption from the input message) is pseudorandom in the Quantum Random Oracle Model (QROM). This assumption is however not backed by previous art [SAB+20].

2.3.3.3 Security against decryption-failure attacks

The security reductions from the MLWE problem include a term that depends on the probability of a decryption failure. Indeed, the knowledge of ciphertexts that fail to decrypt correctly gives some information on the private key and allows to perform key recovery in the long run. Such attacks exploiting decryption failures have first been demonstrated against NTRU [JJ00, HNP+03, GN07]. While these first works rely on chosen ciphertexts, that can be detected and rejected with a proper Ind-CCA transformation, decryption failures may also happen with well-formed ciphertexts.

For this reason, D'Anvers *et al.* investigate in [DGJ+19] the impact of genuine failing ciphertexts on the security of LWE. They show that an adversary can find, through a brute-force search, *weak* ciphertexts that are more likely than others to lead to decryption failure. By querying for their decryption, actual failures can eventually be found, a few tens or hundreds of which are sufficient for key recovery. The cost of the attack has two main factors: finding weak ciphertexts, which is an offline step that can be sped up using Grover's algorithm, and checking them with the decryption oracle, which can only be accessed classically. In theory, the attack of D'Anvers *et al.* would significantly reduce the security of Kyber (and, similarly, of Saber), by a factor 2^{33} to 2^{70}, assuming that the quadratic speedup of Grover's algorithm is available. However, the attack requires an unreasonably high number of decryption queries: more than 2^{130}, which seems very far from practical[7]. A follow-up work by D'Anvers and other coauthors [DRV20] shows how, once a failing ciphertext is known, additional ones can be found with much reduced effort. However, the authors note that their attack still remains impractical against Kyber and similar schemes, since the number of decryption queries is not unlimited.

While these attacks thus seem irrelevant in practical applications, they still underline the necessity of choosing the parameters of LWE schemes such that decryption failures are essentially nonexistent: hence the choice of the designers of Kyber to ensure a negligible decryption-failure rate (as discussed before, § 2.3.1.7).

2.3.3.4 Practical security

In addition to these asymptotic security proofs, analyzing the complexity of practical attacks is required to set the parameters of the scheme. To this end, the authors of Kyber submission give an in-depth analysis of the attacks against the underlying MLWE problem [SAB+20]. We recall this analysis below.

Their evaluation has two layers. In the first one, they examine the two viable approaches to solve the MLWE instances involved in Kyber: the primal and dual attacks. Both approaches solve the problem through the finding of short vectors in a lattice derived from the public data manipulated by the scheme (public key and/or ciphertext). From this, the authors compute the needed block size to solve the SVP instances using the BKZ algorithm [SE94]. Note that, in the case of Kyber-512 (i.e., the lowest security level only), they rely on the ciphertext compression to add a small amount of rounding noise, that is supposed to increase the security by a few bits.

[7] For reference, NIST required the security of the PQC candidates to hold until at least 2^{64} decryptions.

In the second layer, they convert this block size into an estimate of the cost of running this algorithm, in terms of gate count. On one hand, this gate count is in the RAM model: it assumes that accessing an exponentially large memory has constant cost, thereby underestimating the high memory-access cost of sieving algorithms. On the other hand, lattice-reduction algorithms have seen incremental performance improvements over the years, so small further improvements may be expected, and the cost analysis needs to be conservative about these. Consequently, the designers of Kyber report the attack-cost figures reproduced in table 2.3, of which the classical gate count is always slightly more than the estimated gate count to break the corresponding security level of AES.

Table 2.3: Estimation of the practical security of Kyber, from [SAB+20]

	Kyber-512	Kyber-768	Kyber-1024
at least as hard as	AES-128	AES-192	AES-256
$\log_2$(gate-count cost)	151.5	215.1	287.3
$\log_2$(bits of memory)	93.8	138.5	189.7

2.4 CRYSTALS-Dilithium digital signature and ML-DSA

CRYSTALS-Dilithium, simply designated as Dilithium in the rest of this document, is a digital signature scheme closely related to Kyber in terms of their low-level operations. It was, likewise, submitted to the NIST standardization process, and with the end of the third round of this process, the decision was made to standardize this scheme under the name ML-DSA (Module Lattice Digital Signature Algorithm). As for Kyber, our main focus will be on the latest specification provided by the submitters, version 3.1 [BDK+21], and we will briefly mention the changes introduced in ML-DSA.

2.4.1 Specification

Dilithium shares a certain number of similarities with Kyber, hence their belonging to the CRYSTALS family. Both are indeed based on the Module-LWE problem, instantiated over similar polynomial rings. In the case of Dilithium, this ring is $R_{q_\mathrm{D}} = \mathbb{F}_{q_\mathrm{D}}[X]/(X^n+1)$, where the degree of the reducing polynomial is $n = 256$ as for Kyber, but the order of the underlying finite field is different: $q_\mathrm{D} = 2^{23} - 2^{13} + 1 = 8\,380\,417$, a 23-bit prime.

2.4.1.1 Polynomial arithmetic

As before, polynomial arithmetic heavily relies on NTT-domain multiplication, with the difference that Dilithium NTT is complete contrary to that of Kyber. Thus, when implementing the forward and inverse NTT along algorithm 2.1 and algorithm 2.2, the number of layers is chosen equal to $L = 8$. Multiplication in the NTT domain is then accomplished through coefficient-wise multiplication, denoted by the $\cdot$ symbol.

2.4.1.2 Random sampling

Dilithium relies on four specific types of random sampling (not including the sampling of uniformly and independently distributed bit strings), used to sample the public matrix, the secret key, the mask polynomial, and the challenge.

As usual for LWE-based schemes, the public matrix $\mathbf{A}$ is sampled uniformly at random from $R_{q_\mathrm{D}}^{k\times\ell}$. This is accomplished with uniform rejection sampling of each polynomial separately, as expressed in algorithm 2.12. This procedure, ExpandAi, takes 24 bits of randomness at a time, drops their most significant bit, and compares the integer represented by the remaining 23 bits with q_D. If they are lower than this bound, the corresponding sample is accepted as the next polynomial coefficient, otherwise the sample is rejected and the next chunk of 24 bits is considered.

Algorithm 2.12: Dilithium ExpandAi function for public-matrix expansion

Input: Stream of 24-bit words $(w_i) \in [\![0, 2^{24}-1]\!]^*$
Output: NTT-domain polynomial $\hat{a} = (\hat{a}[0], \ldots, \hat{a}[n-1]) \in \hat{R}_{q_\mathrm{D}}$

```
i = 0
for j = 0 to n − 1 do
    while w_i ⩾ q_D do i = i + 1        // Rejection sampling in F_{q_D}
    â[j] = w_i
end
return â
```

The sampling of the secret key likewise chooses uniformly-distributed coefficients, but in a very small range of the form $[\![-\eta, \eta]\!]$, where $\eta \in \{2, 4\}$. To make the operation efficient in software, this sampling takes as input 4-bit nibbles, irrespective of the value of η, and performs the rejection sampling either in $[\![0, 2\eta]\!] = [\![0, 8]\!]$ when $\eta = 4$, or in $[\![0, 3(2\eta+1)-1]\!] = [\![0, 14]\!]$ when $\eta = 2$. In the latter case, the resulting sample is then reduced modulo $2\eta + 1 = 5$. Then, the obtained integer is subtracted from η to shift the result into the correct centered range. The description of this operation, ExpandSi, can be found in algorithm 2.13.

The mask polynomials, later on referred to as $\mathbf{y}_i$, need no rejection sampling. The input randomness is taken in chunks of $\log_2(\gamma_1)+1 \in \{18, 20\}$ bits at a time, and the obtained integers are subtracted from γ_1 to obtain samples in the range $[\![-\gamma_1+1, \gamma_1]\!] \bmod q_\mathrm{K}$, which are used as the successive coefficients of the considered polynomial of $\mathbf{y}$ following algorithm 2.14.

Finally, a constant-weight *challenge polynomial*, denoted by c, must be computed during signature generation and verification. This polynomial must contain exactly τ coefficients in $\{-1, +1\}$, the rest being zero. This is accomplished with *inside-out* Fisher-Yates shuffling: after choosing at random the signs of the τ nonzero coefficients and putting them in the high-order coefficients of the polynomial, τ iterations of shuffling are used to spread them out uniformly at random over all coefficients. The i^{th} iteration chooses a random coefficient index $j \in [\![0, n-\tau+i]\!]$, and exchanges the corresponding value with that of coefficient $n-\tau+i$. This sampling is formalized in algorithm 2.15.

Algorithm 2.13: Dilithium ExpandSi function for the sampling of secret polynomials

Parameter: Half-width $\eta \in \{2, 4\}$ of the uniform-sampling range
Input: Stream of 4-bit words $(w_i) \in [\![0, 15]\!]^*$
Output: Polynomial $s = \sum_{i=0}^{n-1} s[i]X^i \in R_{q_D}$ with each $s[i]$ uniformly distributed in $[\![-\eta, \eta]\!] \bmod q_D$

$i = 0$
// Rejection-sampling bound: the largest multiple of $2\eta + 1$ strictly smaller than 16
if $\eta = 2$ **then** $bound = 15$ **else** $bound = 9$
for $j = 0$ to $n - 1$ **do**
  while $w_i \geqslant bound$ **do** $i = i + 1$
  $x = w_i$
  $i = i + 1$
  if $\eta = 2$ **then** $x = x \bmod (2\eta + 1)$
  $s[j] = \eta - x$
end
return s

Algorithm 2.14: Dilithium ExpandMaski function: sampling of mask polynomials

Parameter: Scheme parameter $\gamma_1 \in \{2^{17}, 2^{19}\}$
Input: Stream of $(\log_2 \gamma_1 + 1)$-bit words $(w_i) \in [\![0, 2\gamma_1 - 1]\!]^*$
Output: Polynomial $s = \sum_{i=0}^{n-1} s[i]X^i \in R_{q_D}$ with each $s[i]$ uniformly distributed in $[\![-\gamma_1 + 1, \gamma_1]\!] \bmod q_D$

for $j = 0$ to $n - 1$ **do** $a[j] = (\gamma_1 - w_j) \bmod q_D$
return a

Algorithm 2.15: Dilithium SampleInBall function

Input: Stream of 8-bit bytes $(b_i) \in \mathbb{B}^*$
Parameter: Radius $\tau \leqslant 64$
Output: Polynomial $c = \sum_{i=0}^{n-1} c[i]X^i \in R_{q_D}$ such that $\forall i, c[i] \in \{-1, 0, +1\}$ and $\|c\|_1 = \tau$

$c = 0 \in R_{q_D}$
for $i = 0$ to $\tau - 1$ **do** *// Sample τ signs from the first 8 bytes*
  $s = (b_{\lfloor i/8 \rfloor} \gg (i \bmod 8)) \bmod 2$ *// The i^{th} sign bit is read from bit $(i \bmod 8)$ of input byte $\lfloor i/8 \rfloor$*
  $c[n - \tau + i] = (-1)^s$
end
$k = 8$ *// k is the index in the randomness string: the remainder of the first 8 bytes are dropped*
for $i = 0$ to $\tau - 1$ **do**
  while $b_k > n - \tau + i$ **do** $k = k + 1$ *// Rejection sampling in $[\![0, n - \tau + 1]\!]$*
  $j = b_k$
  $k = k + 1$
  $c[j], c[n - \tau + i] = c[n - \tau + i], c[j]$
end
return c

2.4.1.3 Integer decomposition, rounding and hints

Dilithium uses two kinds of integer-decomposition operations, that split integer coefficients into high-order and low-order parts. Each of these decomposition methods has its own purpose. On one hand, the Power2Round function, exclusively used to reduce the size of the public key, computes the high-order digit as a rounded division by $2^d = 2^{13}$:

$$\begin{aligned} \text{Power2Round}\colon [\![0, q_\text{D} - 1]\!] &\longrightarrow [\![0, \tfrac{q_\text{D}-1}{2^d}]\!] \times [\![-2^{d-1} + 1, 2^{d-1}]\!] \\ x &\longmapsto \left(\left\lfloor \frac{x+\theta}{2^d} \right\rfloor, (x+\theta) \bmod 2^d - \theta \right), \\ &\qquad \text{where } \theta = 2^{d-1} - 1. \end{aligned} \tag{2.9}$$

Note that half-way integers are rounded down, so that the low-order digit is comprised between $-2^{d-1} + 1$ and 2^{d-1}.

The second kind of decomposition, Decompose, is crucial for the zero-knowledge property of the scheme. We here define this function according to Coron *et al.* [CGTZ23, Algorithm 8]: this definition is equivalent to the one given in the Dilithium specification, but it avoids explicitly handling the corner case in which the input value is close to q_D. For all the following, we derive a new symbol from parameter γ_2 of the scheme,

$$\delta = \frac{q_\text{D} - 1}{2\gamma_2}\,.$$

The computation of the high-order digit, named $\text{HighBits}_{\gamma_2}$, is a rounded multiplication by $\frac{\delta}{q_\text{D}}$, followed by reduction modulo δ:

$$\begin{aligned} \text{HighBits}_{\gamma_2}\colon [\![0, q_\text{D} - 1]\!] &\longrightarrow [\![0, \delta - 1]\!] \\ x &\longmapsto \left\lfloor \tfrac{\delta}{q_\text{D}} x \right\rceil \bmod \delta\,. \end{aligned} \tag{2.10}$$

The low-order digit, $\text{LowBits}_{\gamma_2}$, can then be computed easily as

$$\begin{aligned} \text{LowBits}_{\gamma_2}\colon [\![0, q_\text{D} - 1]\!] &\longrightarrow [\![-\gamma_2, \gamma_2]\!] \\ x &\longmapsto (x - 2\gamma_2 \,\text{HighBits}_{\gamma_2}(x)) \bmod q_\text{D} \end{aligned} \tag{2.11}$$

by subtracting the high-order digit multiplied by its weight $2\gamma_2$ from the input value. Finally, the combined function $\text{Decompose}_{\gamma_2}$ returns both digits, most significant first:

$$\begin{aligned} \text{Decompose}_{\gamma_2}\colon [\![0, q_\text{D} - 1]\!] &\longrightarrow [\![0, \delta - 1]\!] \times [\![-\gamma_2, \gamma_2]\!] \\ x &\longmapsto (\text{HighBits}_{\gamma_2}(x), \text{LowBits}_{\gamma_2}(x))\,. \end{aligned} \tag{2.12}$$

Since the coefficients of the public key are rounded to multiples of 2^d for bandwidth efficiency, signature verification may fail unless some additional information is given to correct for this approximation. With this in mind, the concept of *hints* is defined: conceptually, they specify the carry or borrow that would propagate from the low-order digits to the high-order ones when verifying a signature. This information is computed

by the signer and included in the signature, so that the verifier can compensate for the lack of precision of the compressed public key. To do so, two functions are specified: $\text{MakeHint}_{\gamma_2}$ allows the signer to compute the hints as

$$\begin{aligned}\text{MakeHint}_{\gamma_2} \colon [\![0, q_\text{D} - 1]\!]^2 &\longrightarrow \mathbb{b} \\ z, r &\longmapsto \begin{cases} 1 & \text{if } \text{HighBits}_{\gamma_2}(r) = \text{HighBits}_{\gamma_2}(r + z), \\ 0 & \text{otherwise}, \end{cases}\end{aligned} \tag{2.13}$$

while $\text{UseHint}_{\gamma_2}$ is used by the verifier to process them through

$$\begin{aligned}\text{UseHint}_{\gamma_2} \colon \mathbb{b} \times [\![0, q_\text{D} - 1]\!] &\longrightarrow [\![0, \delta - 1]\!] \\ h, r &\longmapsto \begin{cases} (\text{HighBits}_{\gamma_2}(r) + h) \bmod \delta & \text{if } \text{LowBits}_{\gamma_2}(r) > 0, \\ (\text{HighBits}_{\gamma_2}(r) - h) \bmod \delta & \text{otherwise.} \end{cases}\end{aligned} \tag{2.14}$$

To reduce the size of the signature, a compact representation is used for the hints, by only transmitting the indexes of the set hints. In order for the signature to be of a constant size, a fixed space is reserved for this information: any shorter description of the hints is padded to the correct size, while any longer description is rejected as invalid.

2.4.1.4 Key generation

Dilithium key generation, shown in algorithm 2.16, consists in sampling a public matrix

Algorithm 2.16: Dilithium key generation

Output: Private key $sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \in \mathbb{b}^{256} \times \mathbb{b}^{512} \times \mathbb{b}^{256} \times R_{q_\text{D}}^{\ell} \times R_{q_\text{D}}^{k}$
Output: Public key $pk = (\rho, \mathbf{t}_1) \in \mathbb{b}^{256} \times R_{q_\text{K}}^{k}$
$\zeta \xleftarrow{\$} \mathbb{b}^{256}$
$(\rho, \rho', K) = \text{SHAKE256}(\zeta) \in \mathbb{b}^{256} \times \mathbb{b}^{512} \times \mathbb{b}^{256}$
for $i = 0$ **to** $k - 1$ **do** *// Expand the public matrix in NTT domain*
  for $j = 0$ **to** $\ell - 1$ **do**
    $\hat{\mathbf{A}}_{i,j} = \text{ExpandAi}(\text{SHAKE128}(\rho \,\|\, i_\mathbb{B} \,\|\, j_\mathbb{B}))$ *// i and j are considered as 8-bit integers*
  end
end
for $j = 0$ **to** $\ell - 1$ **do** *// Expand the secret vectors*
  $(\mathbf{s}_1)_j = \text{ExpandSi}(\rho' \,\|\, j_{\mathbb{B}^2})$ *// j is considered as a 16-bit integer*
end
for $i = 0$ **to** $k - 1$ **do**
  $(\mathbf{s}_2)_i = \text{ExpandSi}(\rho' \,\|\, (i + \ell)_{\mathbb{B}^2})$ *// $(i + \ell)$ is considered as a 16-bit integer*
end
$\mathbf{t} = \text{NTT}^{-1}(\hat{\mathbf{A}} \cdot \text{NTT}(\mathbf{s}_1) + \text{NTT}(\mathbf{s}_2))$
$\mathbf{t}_1, \mathbf{t}_0 = \text{Power2Round}(\mathbf{t})$
$tr = \text{SHAKE256}(\rho \,\|\, \mathbf{t}_1)$ *// $\mathbf{t}_1$ is packed with 10 bits per coefficient, in little-endian byte order*
$pk = (\rho, \mathbf{t}_1)$
$sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$
return (sk, pk)

with uniformly random coefficients, $\mathbf{A} \in R_{q_D}^{k \times \ell}$, and small secret vectors $\mathbf{s}_1 \in R_{q_D}^{\ell}$ and $\mathbf{s}_2 \in R_{q_D}^{k}$. The public key is the composed of $\mathbf{A}$ (represented as its seed, ρ), together with the LWE samples $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$. To decrease the size of the public key, some low-order digits of $\mathbf{t}$ are discarded, and only $\mathbf{t}_1 = \text{HighBits}(\mathbf{t})$ is actually transmitted. The secret key contains the two secret vectors $\mathbf{s}_1, \mathbf{s}_2$ and a random key K used for deterministic signing, together with some public information: the seed ρ of the public matrix, the hash tr of the public key, and the low-order bits of the public vector, $\mathbf{t}_0 = \text{LowBits}(\mathbf{t})$ (which are not included in the public key, but could be revealed without any loss of security).

2.4.1.5 Signature generation

Dilithium is based on a zero-knowledge proof of knowledge, converted into a signature scheme through the heuristic of *Fiat-Shamir with aborts*. The Fiat-Shamir construction [FS87] turns an interactive proof of knowledge into digital signature by deterministically deriving the verifier's challenge from the commitment and the message to be signed through a random oracle. A direct instantiation of this construction with lattice-based cryptography would however not be secure, since Lyubashevsky showed that in lattice-based proofs of knowledge, a prover responding to all challenges would leak private information [Lyu08]. He corrected this with the notion of *aborts*, whereby the prover would refuse to respond to some challenges to preserve security. For signatures, an abort consists in stopping signature generation when an insecure challenge is obtained, and reattempting signature from the start with different randomness, until a secure signature transcript is obtained [Lyu09].

We show in algorithm 2.17 the process for signature generation. Depending on implementation choices, signature generation may start with some precomputations: expanding matrix $\mathbf{A}$, and computing the NTT of vectors $\mathbf{s}_1$, $\mathbf{s}_2$ and $\mathbf{t}_0$. Then, the public key and message are hashed into μ, and random coins ρ' are either sampled at random, or deterministically derived from μ and secret element K. After this, the rejection-sampling loop starts: this loop iterates as many times as needed until a valid and secure signature is found. Each iteration is a fresh signature attempt, with independent randomness (this is accomplished by postfixing the random coins ρ' with a nonce that gets incremented at each iteration). Apart from precomputations, all iterations are independent of one another, and each has the same success probability, which (heuristically) only depends on the parameters of the scheme [BDK+21]. The number of iterations before success thus follows a geometric distribution. The running time of signature generation may have a more complex distribution instead, since there are several places in which an attempt can abort: thus, aborted iterations may each take a different time.

Each signature attempt is as follows: a mask $\mathbf{y} \in R_{q_D}^{\ell}$ is sampled at random from the coins ρ', concatenated with a zero nonce. It is then multiplied with the public matrix $\mathbf{A}$, giving $\mathbf{w}$. From this, a commitment is derived as the high bits $\mathbf{w}_1$ of $\mathbf{w}$. In accordance with the Fiat-Shamir transform, a challenge $c \in R_{q_D}$ is then derived from the commitment through a hash function, modeled as a random oracle. The response to the challenge is computed as $\mathbf{z} = \mathbf{y} + c \times \mathbf{s}_1$.

Algorithm 2.17: Dilithium signature generation

Input: Private key $sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \in \mathbb{b}^{256} \times \mathbb{b}^{512} \times \mathbb{b}^{256} \times R_{q_\mathrm{D}}^{\ell} \times R_{q_\mathrm{D}}^{k}$
Input: Message to be signed $M \in \mathbb{b}^*$
Parameter: Choice between deterministic and randomized signature generation
Output: Signature $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h}) \in \mathbb{b}^{256} \times R_{q_\mathrm{D}}^{\ell} \times R_{q_\mathrm{D}}^{k}$

for $i = 0$ **to** $k - 1$ **do** . . . . . *// Expand the public matrix in NTT domain*
  for $j = 0$ **to** $\ell - 1$ **do**
    $\hat{\mathbf{A}}_{i,j} = \mathrm{ExpandAi}(\mathrm{SHAKE128}(\rho \parallel i_{\mathbb{B}} \parallel j_{\mathbb{B}}))$ . . . *// i and j are considered as 8-bit integers*
  end
end
$\hat{\mathbf{s}}_1 = \mathrm{NTT}(\mathbf{s}_1)$; $\hat{\mathbf{s}}_2 = \mathrm{NTT}(\mathbf{s}_2)$; $\hat{\mathbf{t}}_0 = \mathrm{NTT}(\mathbf{t}_0)$ . . *// Precompute the NTT of relevant elements*
$\mu = \mathrm{SHAKE256}(tr \parallel M) \in \mathbb{b}^{512}$
if using deterministic signing **then** $\rho' = \mathrm{SHAKE256}(K \parallel \mu)$ **else** $\rho' \xleftarrow{\$} \mathbb{b}^{512}$
$\kappa = 0$
do . . . . . *// Rejection-sampling loop*
  $\mathbf{y} = (\mathrm{ExpandMaski}(\rho' \parallel (\kappa + i)_{\mathbb{B}^2}))_{0 \leqslant i < \ell} \in R_{q_\mathrm{D}}^{\ell}$ . . . *// $\kappa + i$ as a 16-bit little-endian integer*
  $\mathbf{w} = \mathrm{NTT}^{-1}(\hat{\mathbf{A}} \cdot \mathrm{NTT}(\mathbf{y})) \in R_{q_\mathrm{D}}^{k}$
  $\mathbf{w}_1 = \mathrm{HighBits}(\mathbf{w})$
  $\tilde{c} = \mathrm{SHAKE256}(\mu \parallel \mathbf{w}_1) \in \mathbb{b}^{256}$
  $c = \mathrm{SampleInBall}(\tilde{c}) \in R_{q_\mathrm{D}}$
  $\hat{c} = \mathrm{NTT}(c)$
  $\mathbf{z} = \mathbf{y} + \mathrm{NTT}^{-1}(\hat{c} \cdot \hat{\mathbf{s}}_1) \in R_{q_\mathrm{D}}^{\ell}$
  $\mathbf{r} = \mathbf{w} - \mathrm{NTT}^{-1}(\hat{c} \cdot \mathbf{s}_2) \in R_{q_\mathrm{D}}^{k}$
  $\mathbf{r}_0 = \mathrm{LowBits}(\mathbf{r})$
  if $\|\mathbf{z}\|_\infty \geqslant \gamma_1 - \tau\eta$ or $\|\mathbf{r}_0\|_\infty \geqslant \gamma_2 - \tau\eta$ **then**
    $(\mathbf{z}, \mathbf{h}) = \perp$
  else
    $\mathbf{g} = \mathrm{NTT}^{-1}(\hat{c} \cdot \hat{\mathbf{t}}_0)$
    $\mathbf{h} = \mathrm{MakeHint}(\mathbf{g}, \mathbf{r})$
    if $\|\mathbf{g}\|_\infty \geqslant \gamma_2$ or $\|\mathbf{h}\|_1 > \omega$ **then** $(\mathbf{z}, \mathbf{h}) = \perp$
  end
  $\kappa = \kappa + \ell$
while $(\mathbf{z}, \mathbf{h}) = \perp$ . . . . . *// Repeat signature attempt until success*
return $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

The signer then performs two checks on the candidate signature, to make sure it satisfies the zero-knowledge property and does not reveal any information on $\mathbf{s}_1$ and $\mathbf{s}_2$. To this end, both conditions $\|\mathbf{z}\|_\infty < \gamma_1 - \tau\eta$ and $\|\mathbf{r}_0\|_\infty < \gamma_2 - \beta$ must be verified.

If both of these checks pass, the signer finally makes sure that the signature is valid (in the sense that it will pass verification) by ascertaining that $\|c \times \mathbf{t}_0\|_\infty < \gamma_2$. This condition is essential for the verifier to compute the same commitment as the prover. Since the number of set hints is bounded for efficiency reasons (as mentioned in § 2.4.1.3), the signer also ensures that this number is smaller than ω.

Once all conditions are satisfied, signature generation is successful and the challenge, response, and hints $(\tilde{c}, \mathbf{z}, \mathbf{h})$ are packed and sent to the other party. Otherwise, signature

is attempted again until success: at each iteration, $\mathbf{y}$ is sampled from the same random coins ρ', but an incremented value of the nonce κ, expressed over 16 bits[8].

2.4.1.6 Signature verification

The signature-verification process is straightforward, as can be seen in algorithm 2.18. The message digest μ is computed like during signature generation (except that the hash of the public key must be recomputed) and the challenge polynomial is derived from its seed $\tilde{c}$, which is included in the received signature. Several NTT operations are computed, so that the verifier can derive $\mathbf{w}' = \mathbf{A} \times \mathbf{z} - c \times \mathbf{t}_1 \cdot 2^d$. For a valid signature, this value should have the same high-order bits as the $\mathbf{w}$ computed by the signer, up to a difference of one high-order digit.

Algorithm 2.18: Dilithium signature verification

Input: Public key $pk = (\rho, \mathbf{t}_1) \in \mathbb{b}^{256} \times R_{q_\mathrm{D}}^k$
Input: Message $M \in \mathbb{b}^*$
Input: Signature $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h}) \in \mathbb{b}^{256} \times R_{q_\mathrm{D}}^\ell \times R_{q_\mathrm{D}}^k$
Output: Accept ($\top$) or reject ($\bot$) the signature

for $i = 0$ **to** $k - 1$ **do** . . . . . *// Expand the public matrix in NTT domain*
  for $j = 0$ **to** $\ell - 1$ **do**
    $\hat{\mathbf{A}}_{i,j} = \mathrm{ExpandAi}(\mathrm{SHAKE128}(\rho \parallel i_\mathbb{B} \parallel j_\mathbb{B}))$ . . . *// i and j are considered as 8-bit integers*
  end
end
$\mu = \mathrm{SHAKE256}(\mathrm{SHAKE256}(\rho \parallel \mathbf{t}_1) \parallel M) \in \mathbb{b}^{512}$
$c = \mathrm{SampleInBall}(\tilde{c}) \in R_{q_\mathrm{D}}$
$\hat{c} = \mathrm{NTT}(c)$; $\hat{\mathbf{z}} = \mathrm{NTT}(\mathbf{z})$; $\hat{\mathbf{t}}_1 = \mathrm{NTT}(\mathbf{t}_1)$
$\mathbf{w}' = \mathrm{NTT}^{-1}(\hat{\mathbf{A}} \cdot \hat{\mathbf{z}} - \hat{c} \cdot \hat{\mathbf{t}}_1 \cdot 2^d)$
$\mathbf{w}'_1 = \mathrm{UseHint}(\mathbf{h}, \mathbf{w}')$
if $\|\mathbf{z}\|_\infty < \gamma_1 - \tau\eta$ and $\tilde{c} = \mathrm{SHAKE256}(\mu \parallel \mathbf{w}'_1)$ and $\|\mathbf{h}\|_1 < \omega$ **then return** $\top$ . . . . *// Accept*
else return $\bot$ . . . . . *// Reject*

The signer's commitment value is then determined by correcting $\mathbf{w}'$ thanks to the signer-generated hints $\mathbf{h}$, giving $\mathbf{w}'_1$ supposedly equal to $\mathbf{w}_1$. At this point, the verifier performs three checks to validate the signature: vector $\mathbf{z}$ must have a small norm, the challenge polynomial must have been derived genuinely from $\mathbf{w}'_1 = \mathbf{w}_1$ and μ, and the number of set coefficients in the hint vector must be at most ω.

2.4.1.7 Parameter sets

The parameter sets for the different security levels of Dilithium are reported in table 2.4. All three parameter sets share the same modulus and dimension of the polynomial ring, as well as the rounding parameter for the public key.

[8] Due to the size of κ, at most $2^{16}/k \geqslant 8192$ signature attempts are possible before the nonce wraps around. This iteration limit is, however, impossible to reach in normal conditions, since such a number of consecutive rejections only happens with probability less than 2^{-3000}.

Table 2.4: Recommended parameter sets for Dilithium

Param.	Security level			Description
	2	3	5	
n	256			Dimension of the base ring over $\mathbb{F}_{q_K}$.
q_D	8 380 417			Characteristic of the base ring, prime number $2^{23} - 2^{13} + 1$.
d	13			Rounded bits from the public key, parameterizes Power2Round.
k	4	6	8	First dimension of $\mathbf{A}$, dimension of the underlying lattice.
ℓ	4	5	7	Second dimension of $\mathbf{A}$, rank of the underlying lattice.
η	2	4	2	Radius (∞ norm) for sampling the secret key (ExpandSi).
γ_1	2^{17}	2^{19}	2^{19}	Radius (∞ norm) for sampling $\mathbf{y}$ (ExpandMaski).
γ_2	$\frac{q_D-1}{88}$	$\frac{q_D-1}{32}$	$\frac{q_D-1}{32}$	Radius (∞ norm) of the low-order digit for Decompose.
τ	39	49	60	Number of nonzero coefficients in the challenge polynomial c.
ω	80	55	75	Maximum number of set hints.

2.4.1.8 Number of repetitions

Due to the Fiat-Shamir with Aborts paradigm, signature generation is a pseudorandom process that must check the fulfillment of several conditions and may be started over several times before successful completion. The expected number of repetitions, independent of the private key and message, is given for each security level in table 2.5 as computed by the designers of the scheme [BDK+21]. We complement this information with some representative quantiles, knowing that the number of repetitions follows a geometric law.

Table 2.5: Number of signature attempts until success

Security level	2	3	5
Average number of attempts [BDK+21]	4.25	5.1	3.85
Median	3	4	3
99^{th} percentile	18	22	16
Quantile $(1 - 2^{-16})$	42	51	37
Quantile $(1 - 2^{-32})$	83	102	74
Quantile $(1 - 2^{-64})$	166	204	148
Quantile $(1 - 2^{-128})$	331	407	296

2.4.2 Changes from Dilithium to ML-DSA

As already mentioned, Dilithium has been standardized by NIST under the name ML-DSA [FIP24b]. This standard brings a few changes with respect to the third-round candidate, mainly due to secondary flaws that were discovered late in the process.

2.4.2.1 Increase of the size of some digests

In order to provide better strong-unforgeability properties, the size of some digests have been increased. The hash of the public key, tr, is increased from 256 bits to 512 bits for all parameter sets. On the other hand, the length of the challenge seed, $\tilde{c}$, is made dependent on the security level: it is of 256, 384 or 512 bits, respectively, for levels 2, 3 and 5. Since XOFs are used instead of fixed-length hash functions, only their output length needs to be adjusted.

2.4.2.2 Merge of deterministic and randomized versions

Dilithium specifies two separate versions of the signature-generation algorithm, that are interoperable but have different security properties. The randomized version provides better side-channel resilience, while the deterministic version benefits from a tighter security proof in the QROM [KLS18]. The difference between the two versions lies in how the ρ' seed is computed (line 8 of algorithm 2.17). The ML-DSA standard merges these two versions into a so-called *hedged* version, in which the seed is derived as $\rho' = \text{SHAKE256}(K \parallel rnd \parallel \mu) \in \mathbb{b}^{512}$, where rnd is preferably a freshly-sampled string of 256 bits, but may alternately be a fixed string. The mentioned lengths are independent from the security level.

2.4.2.3 Domain separation between security levels

As for ML-KEM, NIST added domain separation to the sampling of ML-DSA key pairs, so that the expansion of a fixed seed gives completely unrelated key pairs in different security levels. Practically, line 2 of algorithm 2.16 becomes $(\rho, \rho', K) = \text{SHAKE256}(\zeta \parallel k_{\mathbb{B}} \parallel \ell_{\mathbb{B}})$, k and ℓ being the dimensions of $\mathbf{A}$ each expressed over 8 bits [Per24, FIP24b].

2.4.2.4 Input validation

Input-validation checks are specified as part of signature verification, to ensure that all input values are well-formed and have a unique representation. First, the lengths of the public key and the signature must be checked. Second, several properties must be respected by the packed representation of the hint vector, to ensure that this representation is unique. Without this latter check, signatures would not be strongly unforgeable since it would be possible to forge a different signature for the same message by changing the packed hints into a different but equivalent representation.

2.4.2.5 Pure or pre-hash versions and context string

Two signing behaviors are defined for ML-DSA: *pure* signing and *pre-hash* signing. While the former signs the relevant message directly, the latter signs a *digest* of the message, derived through a FIPS-approved hash or XOF. This additional hashing step can bring performance improvements when signing large messages, in two notable cases: when signing on a hardware security module having limited bandwidth, since the digest can

be computed before transmission of the message to this module, and when signing the same message with several signature algorithms [Oun24]. The two signing behaviors are domain-separated, and both support a *context string* being associated with the message to be signed: this context string is similarly domain-separated from the message itself.

2.4.3 Security

The authors of the Dilithium submission give a thorough security analysis of their scheme from the theoretical and practical standpoints. In terms of asymptotic analysis, they show that Dilithium respects the property of Strong Unforgeability under Chosen Message attacks (sUF-CMA) [ADR02], based on a loose reduction from the Module LWE and the Module SIS problems. They argue that the looseness of the reduction stems from the Fiat-Shamir with aborts paradigm, and is not known to be of concern for any scheme instantiated in that paradigm. Besides, they are able to make the reduction tight by assuming the hardness of a non-standard problem, SelfTargetMSIS, which is a variant of the inhomogeneous SIS problem in which the target vector is derived from the searched vector through a random oracle. This hardness can be proved for classical adversaries [BDK+21].

Furthermore, a practical security analysis is conducted, based on the Core-SVP methodology [ADPS16] and on current estimates of the cost of sieving, giving the security figures reproduced in table 2.6, which relate to solving the MLWE problem for key recovery. This analysis does not take into account the module structure, since no sieving algorithm is known to benefit from it. Solving the SIS problem has a similar cost: forging a signature for a new message costs a few more bits than key recovery, while creating a different signature for an already signed message costs a few bits less [BDK+21].

Table 2.6: Estimation of the practical security of Dilithium, from [BDK+21]

	Level-2	Level-3	Level-5
at least as hard as	SHA256 collision	AES-192	AES-256
$\log_2$(gate-count cost)	158.6	216.7	285.4
$\log_2$(bits of memory)	97.8	138.7	187.4

2.A Key-confusion attack on Kyber

We mentioned in section 2.3.2 that expanding a unique seed to Kyber key pairs of various security levels poses a strong security risk. To avoid this risk in implementations that are not appropriately protected against misuse scenarios, the NIST added domain separation into ML-KEM key expansion, so that different security levels produce unrelated key pairs that no longer have this vulnerability [Per24, FIP24a].

We briefly develop here why the Kyber proposal, which does not include this domain separation, is vulnerable in such misuse cases. We recall that the generation of a Kyber Ind-CPA key pair from seed $d \in \mathbb{B}^{32}$ first computes two sub-seeds $\rho, \sigma \in \mathbb{B}^{32}$ from the SHA3-512 digest of d, then derives from them a public matrix $\hat{\mathbf{A}} \in \hat{R}_{q_\mathrm{K}}^{k\times k}$, each element of which is defined by

$$\hat{\mathbf{A}}_{i,j} = \mathrm{Parse}(\mathrm{SHAKE128}(\rho \parallel i_\mathbb{B} \parallel j_\mathbb{B}))\,,$$

and two secret vectors $\mathbf{s}, \mathbf{e} \in R_{q_\mathrm{K}}^k$, defined through

$$\begin{aligned}\mathbf{s}_i &= \mathrm{CBD}_{\eta_1}(\mathrm{SHAKE256}(\sigma \parallel i_\mathbb{B}))\,,\\ \mathbf{e}_i &= \mathrm{CBD}_{\eta_1}(\mathrm{SHAKE256}(\sigma \parallel (i+k)_\mathbb{B}))\,.\end{aligned}$$

From these, the second part of the public key is then given by $\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$, where $\hat{\mathbf{s}}$ and $\hat{\mathbf{e}}$ are the NTT-domain counterparts to $\mathbf{s}$ and $\mathbf{e}$.

Since the η_1 parameter is shared by security levels 3 ($k = 3$) and 5 ($k = 4$), these two security levels end up with the same values for elements $\hat{\mathbf{A}}_{i,j}$ and $\mathbf{s}_i$ for $0 \leqslant i, j < 3$.

Let us denote by $\hat{\mathbf{A}}$, $\hat{\mathbf{t}}$, $\hat{\mathbf{s}}$, $\hat{\mathbf{e}}$ the cryptographic material for security level 3, and by $\hat{\mathbf{A}}'$, $\hat{\mathbf{t}}'$, $\hat{\mathbf{s}}'$, $\hat{\mathbf{e}}'$ that of level 5, derived from the same seed. It then trivially holds that $\hat{\mathbf{s}}' = (\hat{\mathbf{s}}_0\ \hat{\mathbf{s}}_1\ \hat{\mathbf{s}}_2\ \hat{\mathbf{e}}_0)^\top$.

Assuming that an attacker has access to the public keys derived from a common seed in security levels 3 and 5, and that it has the ability to break level-3 public keys (recovering $\hat{\mathbf{s}}$ and $\hat{\mathbf{e}}$), it directly knows the $\hat{\mathbf{s}}'$ level-5 secret vector, from which it can also trivially recover $\hat{\mathbf{e}}' = \hat{\mathbf{t}}' - \hat{\mathbf{A}}' \circ \hat{\mathbf{s}}'$. The level-5 key is thus downgraded to level-3 security.

Analyzing the effect of also knowing the level-1 public key is more challenging, since this level uses a different parameter for binomial sampling: $\eta_1 = 3$, which means that the first two elements of the secret vector $\mathbf{e}$ are related, but not equal, between level 1 and higher levels. Since CBD sampling is not injective, the attacker cannot recover its input value for $\eta_1 = 3$ to compute it again with $\eta_1 = 2$. However, the two extremal values of the binomial distribution have a single preimage, so that the attacker knows 6 consecutive bits of the SHAKE128 output for each level-1 secret coefficient equal to ± 3 (which happens for 3 % of the coefficients on average). We thus expect the knowledge of the level-1 public key to decrease the security of the corresponding level-3 and level-5 keys, but the extent of this decrease is unclear. We are not aware either of such analysis in the literature.

A similar attack would have been possible against Dilithium; for this reason, the ML-DSA standard likewise includes domain separation in its key-generation procedure.

3 Physical attacks and countermeasures on lattice-based cryptography

This section details the vulnerabilities of LWE-based schemes to side-channel and fault attacks, as well as countermeasures against these. We start by describing the attacks that exist in the literature, classified according to their underlying technique. Along the way, we describe the ad-hoc countermeasures to each attack, and mention which generic countermeasures are also effective. We then separately describe these generic countermeasures in detail.

Howe *et al.* present in [HPA21] a review of side-channel and fault attacks against lattice-based cryptography. A particularly sensitive module in the key-encapsulation mechanisms is the Fujisaki-Okamoto transform that provides chosen-ciphertext-attack (CCA) security. Several previous works have shown this element to be both difficult to implement securely and easy to attack even when the implementation is correct. We will thus put a particular emphasis on these attacks in the following.

3.1 Timing attacks and simple power analysis (SPA)

3.1.1 Security against timing attacks

Apart from implementation mistakes, Kyber and Dilithium are designed to be mostly constant-time, with a few operations such as rejection sampling that are not constant-time but whose timing variation is independent from any secret value. The easily exploitable channel of coarse-grained timing attacks is thus usually not a concern here.

Interestingly, many software implementations of Kyber have contained a common timing vulnerability, not discovered until the end of 2023 [Ber23b]. Kyber compression function $\text{Compress}_d(x) = \lfloor x2^d/q_\text{K} \rceil \bmod 2^d$ involves rescaling a coefficient x from range $[\![0, q_\text{K} - 1]\!]$ to range $[\![0, 2^d - 1]\!]$, where x may be secret. In particular, a specialization of this function, Compress_1, is used to decode the message during decryption (line 4 of algorithm 2.8). Due to the division by prime number q_K, this function is often compiled to use an explicit division instruction, which has variable timing on many platforms [Ber24]. Consequently, implementations having this behavior may contain an exploitable timing leakage depending on platform and compiler optimizations [Ber23b, Ber24]. A similar timing vulnerability was reported shortly after, in general-case implementations of Compress_d, this time used during the encryption procedure [Rav23]. Both vulnerabilities are corrected (assuming a well-behaved compiler and a constant-time multiplication instruction, which are not a given [dG15]) by rewriting this division as multiplication by the inverse of q.

Dilithium contains several sources of timing variability on operations involving secret values, but these variations do not reveal any sensitive information. One such timing variability is during the rejection sampling of the small uniform polynomials $\mathbf{s}_1$ and $\mathbf{s}_2$ using ExpandSi (algorithm 2.13). However, this variability is not a concern for security: for each sample, the only information embedded in the variability is whether a given sample is accepted or rejected. Since each sample is independent from its neighbors due to the pseudorandomness of the hash that generates them, no information on the value of the accepted samples is revealed through this channel [BDK+21].

Another instance of this timing variability in Dilithium is the rejection sampling of the signature, which reiterates signature generation until a valid and secure signature is obtained. Again, the rejection conditions are independent from any secret value; furthermore, when checking the norms of vectors $\mathbf{z}$ and $\mathbf{r}_0$, the *index* of the coefficient which violates the bound may be revealed without problem [BDK+21].

3.1.2 Security against SPA

Straightforward implementations of Kyber and Dilithium contain no secret-dependent branches, and are therefore without the blatant SPA vulnerabilities linked with the square-and-multiply–type (and similar) algorithms used for RSA and elliptic-curve cryptography [KJJ99, Cor99].

However, SPA attacks have been demonstrated against them, as shown by Steffen *et al.* [SKB22] against a software implementation of Kyber. By targeting a power leakage in the message-encoding step of encryption (line 15 of algorithm 2.7), the authors are able to mount a successful message-recovery attack. Indeed, the encoding step maps each bit of the message to zero or $\lfloor q/2 \rceil$, depending on the value of the bit. Since these two values have different Hamming weights, their processing leads to measurably different power consumptions in software implementations. Steffen *et al.* analyze various strategies for the implementation of message encoding, in order to minimize its leakage. They start from the reference implementation of Kyber, which exhibits strong leakage due to its reliance on a 16-bit mask with either all bits set or all bits cleared, depending on the message bit. They then manage to significantly decrease the signal-to-noise ratio of the encoding procedure by decoding the message to two polynomials: one that is actually used, and a dummy one that will be discarded. The bits of the message are processed by setting coefficients of either the real encoding polynomial, or the dummy one, depending on the bit values. Further refinements based on balanced-weight table lookups and on randomizing the order of processing provide additional noise, ultimately making their SPA attack no better than random guess [SKB22]. It is however noteworthy that the attack does not seem to take into account the presence of shuffling. In this case, its failure is expected and should not be considered to validate the other countermeasures: indeed, with all proposed countermeasures apart from shuffling, the attack has a success rate of 64 %, which is still significant.

3.2 Soft analytical side-channel attacks

While SPA attacks try to directly determine secret bits from independent leakage points, side-channel traces contain much more information than this. In particular, they may reveal intermediate values computed from several secrets, or that are used to derive them. Although the knowledge of these intermediate values may have no interest in itself, it can be used to assist with the recovery of the secret, as soon as the relations between these quantities are known. This is the principle behind SASCA, introduced by Veyrat-Charvillon *et al.* [VGS14]: using the knowledge of the equations between secrets and intermediate values to mutualize the information from all leakage points. To do so efficiently, belief propagation relies on a message-passing algorithm. As a starting point, the attacked algorithm is modeled as a *factor graph*, composed of two kinds of nodes: *variable nodes*, which represent the unknown variables maipulated by the algorithm, and *factor nodes*, which formalize the arithmetic equations between variables. Then, the prior probabilities assigned to the values of each variable, based on side-channel information, are passed along the graph, and successively refined through application of the law of total probabilities and Bayes's rule, until the *belief* on each variable converges. From this final state, marginal (i.e. independent) probability distributions for each secret value are derived.

3.2.1 Attacking the Number-Theoretic Transform

In particular, Primas *et al.* [PPM17] demonstrated this attack against the NTT computation in a generic Ring-LWE scheme. Indeed, computing the NTT of a polynomial involves performing a relatively large number of simple arithmetic operations on pairs of its coefficients, along a regular, layered structure, which is well fit to SASCA. Their attack consists of three phases. First, they perform a profiling phase, in which they build side-channel templates for the three leaking operations within butterfly computations: modular addition, subtraction, and multiplication by the twiddle factor. Then, they run the belief-propagation algorithm on the acquired traces, after template matching. For effective and accurate convergence, they do not attack the full execution of the NTT in this step. Since the operation that leaks most strongly, modular multiplication, is not evenly distributed in the NTT computation graph, restrict the attack to three large subgraphs to better exploit the available side-channel leakage. Due to this choice, they only recover 192 out of 256 coefficients from the third-to-last layer of the NTT. The attack thus necessitates a last phase, in which the full secret polynomial is recovered through lattice reduction, with the help of the corresponding public key. The authors report good attack performance, both for a physical target (shared between the profiling and attack phases) and in simulation, being able to perfectly recover the secret polynomial. Since the attack needs a single trace, it is similarly effective against masked implementations. One important caveat, though, is that the implementation they attack performs modular multiplication in variable time (thus explaining the attack effectiveness).

Pessl *et al.* [PP19b] improve the attack, making it more resilient to noise and reducing the initial effort for templating the targeted implementation. Among other developments,

they facilitate the convergence of belief propagation by reworking the factor graph so that it does not contain short loops, and by using message damping (that we detail in section 4.2.3). Very importantly, they also change the attack target: while the previous attack [PPM17] targets the forward NTT performed during decryption, this later work [PP19b] instead targets the inverse NTT used by encryption. Although encryption does not manipulate the long-term secret key (except indirectly, when called by the decapsulation procedure), thus being a less desirable target, it is also less likely to be strongly protected.

Hamburg *et al.* [HHP+21] continue this line of work and show that, by using sparse chosen ciphertexts, noise tolerance can be further improved and some assumptions made by the previous attacks can be relaxed, at the cost of having to acquire a few traces to recover the complete long-term secret key.

The above-mentioned attacks targeted relatively old software implementations of the NTT (either for Kyber of for generic schemes). Considering the thorough optimizations that were added to more recent implementations of Kyber NTT, in particular, the use of Plantard modular arithmetic [HZZ+22] in the `pqm4` implementation dedicated to ARM Cortex-M4 microcontrollers [KPR+22], a refreshing of these results was due. We adapted the attack to this new setting and found that this more compact implementation was still vulnerable, even in the presence of significant noise in the measurements. Our adapted attack and its results on simulated side-channel traces are detailed in chapter 4.

3.2.2 Attacking Keccak

Another constituent of lattice-based cryptography which has been the target of SASCA is the Keccak family of hash functions: SHA3 and SHAKE. Hashing plays a key role in Kyber, Dilithium, and related schemes, as it notably serves as a source of strong pseudorandomness for the sampling of secrets, errors, challenges and such. Since, in many of these contexts, a given input is only hashed once or a few times, single-trace attacks are especially important. Kannwischer *et al.* [KPP20] thus evaluate belief-propagation attacks against software implementations of Keccak for 8-bit and 32-bit CPUs. Their attack, which is restricted to the first two rounds of the *absorbing* phase, assumes a secret input of limited width (128 or 256 bits), followed by public data: either all zero or random (but known by the attacker).

In this setting, the authors are able to perfectly recover keys of both lengths on 8-bit CPUs, up to noise levels two to four times higher (in terms of standard deviation) than measured on the profiling device. As expected, the attack is most effective against the shorter (128-bit) keys, and against random public data. Understandably, attacking 16-bit and 32-bit devices proves more difficult, and in the latter case, the attack is only successful against 128-bit keys. Still, in this case, the attack can tolerate the noise levels expected on the real device [KPP20].

It is important to note that Kyber and Dilithium use secret seeds of 256 or 512 bits. While these lengths have not been successfully attacked by Kannwischer *et al.* [KPP20] in 32-bit software implementations, their work is the first, and for now only one, to investigate the resilience of Keccak to SASCA. In this respect, significant improvements

to this kind of attack may well be found with further investigation. The authors notably mention the opportunity to perform few-trace attacks, when a fixed secret is hashed several times together with a varying but publicly known nonce — a situation which includes most uses of Keccak in Kyber and Dilithium.

3.2.3 The shuffling countermeasure

A crucial requirement of straightforward SASCA is that the leakage of the secure device can be unambiguously assigned to the correct intermediate values. Consequently, shuffling is frequently mentioned as an effective countermeasure, since it breaks this knowledge of which quantity leaks at what time [VGS14, PPM17, PP19b, KPP20]. Ravi *et al.*, among others, propose different levels of shuffling for the NTT, from simple *fine shuffling*, in which butterfly operations are kept in order but each loads its two operands in a random order, to *full shuffling*, in which the order of butterflies within a layer is randomized without restriction [RPBC20]. Hermelink *et al.* [HSST23] evaluate various improvements to the belief-propagation attack so that it can cope with this countermeasure. They show that fine shuffling can be somewhat counteracted thanks to either of two techniques: *mixing priors*, whereby the prior probabilities assigned to variables are equally weighted between the shuffled and the non-shuffled cases, and the introduction of *shuffle nodes*, which extend the previous technique by progressively skewing the weighting towards the most probable choice. When the unshuffled case could cope with a noise standard deviation of $\sigma = 1.4$, these two techniques can break fine shuffling up to $\sigma = 0.8$ and $\sigma = 0.9$, respectively [HSST23].

The more thorough shuffling configurations introduced by Ravi *et al.* are increasingly effective at countering the attack — even though they do not prevent it altogether. *Coarse in-block shuffling*, which consists in randomizing the processing order of the butterflies having the same twiddle factor, is handled by measuring the leakage of both load and store operations, and matching the distributions of the variables between layers, to determine likely permutations between them. Depending of the case, either the distributions of the variables are mixed according to their likely shuffled positions, or the possible permutations are enumerated from most to least probable, attempting to run belief propagation each time until convergence. The authors evaluate the mixing solution, showing its success up to $\sigma = 0.45$. They argue that enumerating permutations, while not practical in many cases, should allow for the same noise tolerance as attacks against unshuffled NTT [HSST23].

3.3 Oracle-based chosen-ciphertext attacks

Kyber, like many other PQC schemes (e.g. Saber [DKR+20], FrodoKEM [NAB+20], HQC [AAB+22]), gets its security against chosen-ciphertext attacks (CCA) through a variant of the Fujisaki-Okamoto transform [FO99]. However, CCA security is only valid in a black-box context, and physical attacks can be used to build oracles breaking this property. In this context, an attacker requesting the decryption of malicious ciphertexts can derive information on the long-term secret key, ultimately recovering it in full.

Chosen-ciphertext attacks can be classified into three categories, depending on the type of oracle they rely on [RCDB23]:

- Decryption-Failure (DF) oracles,
- Binary (single-bit) or parallel (multiple-bit) Plaintext-Checking (PC) oracles, or
- Full-Decryption (FD) oracles.

We note that the notions of DF and PC oracles are relatively close and are sometimes used interchangeably, *e.g.* Bhasin *et al.* [BDH+21] use the PC denomination to refer to what is more commonly called a DF oracle.

The three oracles presented above are ordered from weakest to strongest, where a stronger oracle can directly implement the weaker ones. This should be kept in mind in the following: apart from the explicitly listed methods for instantiation, each oracle can also be implemented using the methods listed for the stronger ones.

3.3.1 Decryption-failure oracle

3.3.1.1 Principle and exploitation

A decryption-failure oracle reveals, for a legitimate ciphertext ct and a perturbation Δ, whether ct and $\widetilde{ct} = ct + \Delta$ decrypt to the same value. Practically, for LWE schemes, an attacker encrypts an arbitrary message m using the legitimate encryption algorithm, and gets ciphertext $(\mathbf{u}, v)$. Then, for each coefficient index i and various values of $t \in \mathbb{F}_q$, the decryption of $(\mathbf{u}, \tilde{v})$ is observed, where $\tilde{v} = v + t \cdot X^i$ is obtained by adding t to the ith coefficient of v. For a scheme not making use of ciphertext compression (see § 2.3.1.3), the decryption procedure then decrypts the ciphertext into message M' (before decoding) according to eq. (3.1) [RCDB23]

$$M' = \text{Decompress}_1(m[i]) + \mathbf{e}^\top \cdot \mathbf{r} + e_2 - \mathbf{s}^\top \cdot \mathbf{e}_1 + t \cdot x^i \,, \tag{3.1}$$

in which the i^{th} component is decoded correctly when $|M'[i] - \text{Decompress}_1(m[i])| \lesssim \frac{q}{4}$ (the exact boundaries depending on whether $m[i]$ is set or cleared). Using binary search on t, the exact value of $(\mathbf{e}^\top \cdot \mathbf{r} + e_2 - \mathbf{s}^\top \cdot \mathbf{e}_1)[i]$ can be recovered; then, $\mathbf{s}$ being the only unknown quantity in these equations, it can itself be determined by solving a linear system [RCDB23, BDH+21, GJN20]. In this case, the full secret can be recovered with a number of traces of the order of $kn \log_2(q/4)$.

For schemes that use ciphertext compression, like Kyber does, the situation is slightly less straightforward, since the compression and decompression of the ciphertext are lossy. The attacker then gets a system of approximate relations instead of equalities. Still, this system can be solved with little computational effort to recover $\mathbf{s}$ [PP21, DB22]. Assuming a nearly perfect oracle (e.g. one that misclassifies less than 0.5 % of the ciphertexts), Pessl and Prokop [PP21] are able to perform full key recovery on Kyber-512 with ten thousand traces, while D'Anvers and Batsleer [DB22], with their updated attack, can recover keys up to the highest security level of Kyber in the same number of traces, this time assuming a perfect oracle.

3.3.1.2 Instantiation

Numerous different instantiations of the DF oracle on LWE schemes have been proposed. Most of them leverage the CCA transform of the scheme, since it can amplify the effects of decryption failure from the angle of physical attacks. Indeed, the CCA transform involves performing re-encryption of the decrypted message to make sure that the ciphertext was the output of a honest encryption. Since the randomness used in the encryption phase is derived from a hash of the message, any bit flip in the decrypted message due to decryption failure leads to a completely different re-encryption.

In particular, when doing the comparison between the received and the re-encrypted ciphertexts at line 4 of algorithm 2.11, the comparison will fail on the single perturbed coefficient in case of decryption success, while it will fail with high probability on *every* coefficient in case of decryption failure. The round-2 reference implementations of FrodoKEM [NAB+19], LAC [LLJ+19], HQC [AAB+19a], BIKE [ABB+19], Rollo [ABD+19] and RQC [AAB+19b] provided this oracle through a trivial timing leakage, due to the use of an early-aborting ciphertext comparison [GJN20] that terminated as soon as the first mismatching coefficient was found.

Beside this kind of timing vulnerability (which is easy to avoid), many methods exist to instantiate this oracle: any side-channel distinguisher able to discriminate between a successful decryption and a failed one may be used. For example, Bhasin *et al.* [BDH+21] use a distinguisher based on Welch's t-test against the polynomial-comparison operation. While the implementation they target (from Oder *et al.* [OSPG18]) takes a masked ciphertext as input, the comparison is performed few coefficients at a time and its result is separately unmasked for each group. Consequently, the attacker is able to determine whether a single group or all groups of coefficients failed the comparison, thereby realizing the desired oracle.

A straightforward way to implement this oracle using fault injection is to disable the CCA transform by injecting a fault during the message-equality test of decapsulation (4) so as to force a comparison match, as described by Xagawa *et al.* [XIU+21]. The attacker can then determine whether a shared secret has been successfully established, which indicates successful decryption of the chosen ciphertext.

Hermelink *et al.* [HPP21] describe a more sophisticated usage of fault injection to obtain this oracle. Since the re-encryption of the message results in a ciphertext having a small distance to the received one if and only if decryption was successful, they inject faults attempting to correct this small difference. With overwhelming probability, a successful decapsulation indicates that the difference was corrected (either by removing the perturbation from the chosen ciphertext after decryption, or by introducing the same perturbation in the re-encrypted ciphertext), which in turn indicates successful decryption. They mainly study the case of perfectly reliable fault injection, in which they can recover Kyber-1024 keys with less than ten thousand chosen ciphertexts. However, the authors state that their attack also works with unreliable fault injection, at the cost of requiring a larger number of oracle queries, by only considering successful decapsulations. In particular, their attack is applicable to bit-set or bit-reset faults against masked implementations. In this case, perfectly effective faults introduce the desired perturbation

50 % of the time, and four times more decapsulation attempts are required. Finally, the attack is applicable to shuffled implementations, again with the caveat that more decapsulation requests will be needed.

3.3.1.3 Countermeasures

Implementations of this oracle based on a side-channel distinguisher are generically countered by masking. As discussed above, the whole re-encryption step must be protected, but the comparison of the received and reencrypted ciphertext may be the most difficult part to protect. Techniques for the masking of this operation are discussed in § 3.6.3.5, including some that are known to be flawed and have later been corrected.

Regarding the fault-based DF oracle, Hermelink *et al.* [HPP21] review the effectiveness of standard fault countermeasures. They show that shuffling is not a strong countermeasure against this attack path (as often when faults are involved), since it can at best multiply the attack effort by the number of shuffling positions, and at worst have no effect, for instance if the data is faulted while in memory. Introducing redundancy both in the representation of the sensitive data and by computing the comparison several times, however, seems an effective countermeasure by their analysis.

3.3.2 Plaintext-checking oracle

3.3.2.1 Principle and exploitation

The plaintext-checking oracle provides slightly stronger capabilities to an attacker: this time, there is no need to start from a valid ciphertext and alter it by a slight perturbation; instead, a low-weight ciphertext is crafted entirely, in such a way that the value of the decrypted message m' at line 4 of algorithm 2.8 depends on a single or few coefficients of the private key. The plaintext-checking oracle then indicates some information on the value of m' that was obtained: a binary PC oracle compares the decrypted message with a single message hypothesis, and returns a match or mismatch; a parallel PC oracle, instead, gives several bits of information at a time by classifying m' among a larger set of possible values [Nil23].

The first PC-based attack against LWE was proposed by Bauer *et al.* [BGRR19] against NewHope, under the name of a *key-mismatch oracle*. Since NewHope [PAA+19] decodes each message bit from four polynomial coefficients through majority voting, the oracle gives information on the four corresponding coefficients of the secret key. The authors propose a probabilistic algorithm that can recover secret coefficients, one after the other, using 18 oracle queries per coefficient on average (18 500 queries are needed, on average, for the whole secret key in the highest security level). Each coefficient of the secret key has a 1 % probability to belong to a set of values that cannot be distinguished by the algorithm, and which must be recovered through brute force [BGRR19]. Considering that for the highest security level of NewHope, an expected number of ten coefficients are in this case, each with less than one bit of entropy remaining, the brute-forcing effort is low.

Ravi *et al.* [RRCB20] use looser conditions for the crafting of chosen ciphertexts, thereby making the attack able to distinguish between more secret values. Instead of fixing the value of the $\mathbf{u}$ component of the ciphertext and only varying the value of v, they vary the values of both parts. In the case of NewHope, this allows them to recover all possible values of the secret-key coefficients, leading to full key recovery in 26 624 oracle queries for NewHope-1024. The authors also attack Kyber, for which 5 oracle queries per coefficient are enough using a binary search: thus, 5120 queries are sufficient to recover the private key of Kyber-1024.

Later on, Qin *et al.* reduced this number to 2365 oracle queries (on average) for full key recovery in Kyber-1024. This decreased cost is due to the use of an optimal binary search that takes into account the binomial distribution of secret-key coefficients to recognize the most probable values earlier [QCZ+21].

Some LWE-based schemes, such as LAC, include an error-correction step in their decryption procedure, to eliminate small errors that potentially affect the decrypted plaintext [LLJ+19]. However, Wang *et al.* show that this error-correction capability does not prevent the PC attack, which successfully performs key recovery in the highest security level of LAC using only 3200 chosen ciphertexts [WZJ20].

With a more capable *multiple-valued* PC oracle, key recovery can be performed faster by determining several bits of m' at once. Given an integer $\mu \geqslant 2$, Tanaka *et al.* [TUX+23] consider a μ-valued PC oracle able to distinguish the value of m' among a fixed set $\{m_0, \dots m_{\mu-1}\}$, where m' is guaranteed to belong to this set. Using this oracle, key recovery can be performed in roughly $1/\log_2(\mu)$ times as many queries as with the single-bit PC oracle (which corresponds to $\mu = 2$), by probing several coefficients at once.

3.3.2.2 Instantiation

Wang *et al.* [WZJ20] show that the PC oracle can be trivially instantiated in the black-box model against a public-key encryption algorithm that satisfies CPA security but not the stronger CCA security. For instance, if a server establishes a shared key with its client using public-key encryption to transport the shared key, the client knows whether its ciphertext has been decrypted to $m_{\text{candidate}}$ by checking whether it can communicate with the server using $m_{\text{candidate}}$ as symmetric encryption key. We note that this situation is not supposed to arise, since PKE algorithms that are vulnerable to CCA are not supposed to be used in this way, or are meant to generate a different asymmetric key pair for each transaction. However, misuse cases are unfortunately encountered, so this forbidden situation may actually happen in practice [Yil10, RY10].

Since the re-encryption step of the scheme CCA transform derives encryption randomness from the decrypted message, leakage of this randomness can be used to implement a PC oracle. Practically, Ueno *et al.* [UXT+22] record the side-channel leakage from computing the pseudorandom function[1] to derive this randomness, and classify it thanks to a deep-learning–based distinguisher. This distinguisher, profiled on the target device

[1] Ueno *et al.* attack an AES-based pseudorandom function instead of one based on Keccak, to benefit from the availability of masked AES software and hardware; however, they state that similar results should be expected on Keccak.

without knowledge of the secret key, is then able to determine whether the decrypted plaintext is the reference one or not, thus realizing the desired oracle. Having trained their classifier on 30 k traces, the authors obtain 99.8 % accuracy against unprotected implementations, either in hardware or in software. As expected, attacking masked implementations proves more difficult, but 96 % accuracy is still obtained against masked software, after profiling on 900 k traces. Instead, attacking masked hardware after profiling on one million traces gives accuracy barely better than random guess: 0.515, which is statistically significant but insufficient to mount an acceptably efficient attack.

In a later work [TUX+23], the same authors extend their deep-learning classifier to a μ-valued PC oracle instead of a binary one. When the pseudorandom function used to derive coins from the plaintext is based on AES or SHA3-512, implemented in software, they are able to distinguish between up to $\mu = 2^8$ plaintext values with accuracy above 99.9 %.[2] These figures are obtained after training the distinguisher on 1000μ traces.

While very generic, deep-learning classifiers require a large number of traces for profiling. To decrease this cost, Rajendran *et al.* [RRD+23] instead perform μ-valued classification using a simple t-test–based distinguisher that considers the leakage of the whole re-encryption step from the `pqm4` [KPR+22] software implementation of Kyber. They are able to reach $\mu = 2^{10}$ using as little as 5μ profiling traces. Given this low templating cost, and since knowledge of the secret key is not needed for profiling, it becomes possible to perform integrated attacks that execute both the profiling and the actual attack sequentially on the target device. In this setting, the authors compute the optimal parallelization factor μ to minimize the total number of queries over both phases: with 5 traces per set for profiling, they recover the full key of Kyber768 in 613 queries.

We mentioned above the implementation of a DF oracle using the fault-injection attack of Xagawa *et al.* [XIU+21], by disabling the ciphertext-comparison check using a single fault. This attack can be strengthened into a μ-valued PC oracle, with a precomputation cost of $O(\mu)$, by computing the result of key derivation for all message hypotheses m_i (see line 5 of algorithm 2.11), and checking which shared secret allows for communication with the server.

3.3.3 Full-decryption oracle

3.3.3.1 Principle and exploitation

The extreme case of the PC oracle is the full-decryption (FD) oracle, which gives out the full plaintext decrypted from any (legitimate or not) ciphertext [Nil23].

The FD oracle is typically exploited in the same way as the multiple-valued PC oracle for secret-key recovery, using chosen ciphertexts constructed in such a way that their decryption leaks information on a secret coefficient. Here, however, n coefficients can be attacked simultaneously since all n bits of the plaintext are recovered. Still, it is

[2] Surprisingly, the authors obtain significantly lower accuracy when attacking a software implementation of SHAKE128, for which accuracy drops below 99 % when distinguishing between more than 2^7 plaintexts. Instead, we would expect similar accuracy between SHAKE128 and SHA3-512. The authors do not develop this topic.

generally difficult to recover the plaintext perfectly, and the oracle reply may contain some erroneous plaintext bits. To handle this limitation, Ngo *et al.* [NDGJ21] use an error-correction mechanism: they craft the successive chosen ciphertexts so that their decryption maps to the codewords of an extended Hamming code. By encoding four bits of information over each group of eight bits ($[8, 4, 4]$ extended Hamming code), they are able to correct one erroneous bit for each 8-bit block, and detect two errors[3]. Overall, sixteen chosen ciphertexts suffice to recover the secret key of LightSaber, with the help of a quick lattice-reduction postprocessing to recover the few coefficients that could not be reliably determined through the oracle. Key recovery in Kyber-512 would be at most as hard, since its secret key has the same dimension but a narrower coefficient distribution.

3.3.3.2 Instantiation

A particularity of the above-mentioned attack of Ngo *et al.* is that they target a masked implementation, for which they do not need to deactivate the countermeasure during the profiling phase. By employing a deep learning model, they are able to recover the plaintext bits without explicitly determining the value of the mask. They exploit two leakage points in the masked software implementation of Saber by Van Beirendonck *et al.* [VBDK+21]: the step of masked logical shifting that accomplishes message decoding (line 4 of algorithm 2.8), and the packing of the single-bit decoded coefficients into the 256-bit plaintext.

Other instantiations of the FD oracle typically exploit one of two leakage points: the message-decoding operation at the end of decryption (line 4), or message encoding at the beginning of reencryption line 15, since these two operations process the bits of the message individually, and are likely to have a high signal to noise ratio [RBRC22, SPH22].

Beside, some less obvious operations have been successfully targeted for message recovery. For instance, the previously mentioned attack of Pessl and Primas [PP19b], as well as that of Hamburg *et al.* [HHP+21], are both successful at performing full message recovery with a belief-propagation attack on the NTT during re-encryption. In the latter case, they even attack a masked implementation by recovering the two shares in parallel.

3.3.3.3 Related SCA-assisted chosen-ciphertext attack

Sim *et al.* demonstrated in [SPH22] a CCA exploiting the side-channel leakage of Barrett reduction in the round-3 reference implementation of Kyber for ARM Cortex-M4. Although not an FD oracle, we mention it here since has roughly similar capabilities. Barrett reduction reduces an integer a modulo q to $a' = a - t$ where $t = \lfloor a/q \rfloor q$. This reduction method is used during decryption, before decoding message m' from $M' = v' - U' \bmod q$ where $U' = \mathbf{s}^\top \mathbf{u}'$. In this implementation, U has been partially reduced using Montgomery reduction, so each of its coefficients is comprised between $-q + 1$ and $q - 1$. The subtrahend computed during Barrett reduction, t, is thus contained in $\{-q, 0, q\}$. The authors construct chosen ciphertexts such that each t_i is either $-q$ or 0 depending on

[3]This technique based on error-correcting codes can be generalized to lower-rate codes to cope with low-accuracy oracles.

the value of the private key: these are the two values that have maximum Hamming distance. Then, an SPA can easily recover each t_i, giving as many bits of information at once as an FD oracle. Since three chosen ciphertexts are enough to determine the value of a secret coefficient, and 256 coefficients can be recovered in parallel, full key recovery on Kyber-512 can be achieved with only six chosen ciphertexts.

3.4 Fault-injection attacks

Fault-injection attacks are a powerful tool for an attacker to alter the behavior of the targeted device, so that it deviates from the specified algorithm. We already saw how fault-injection attacks can be used as a generic tool to implement other attacks, in particular oracle-based attacks. In this section, we instead focus on those attacks that can only be implemented with fault injection, because they require tampering (or consistently failing to tamper) with the data manipulated by the target, or with its behavior.

3.4.1 Ineffective-fault attacks

Safe-error attacks [YJ00], also called ineffective-fault attacks [Cla07], derive secret information from the knowledge of whether a successful fault injection changes or not the result of a cryptographic algorithm.

This kind of attack has been demonstrated by Bettale *et al.* [BMR21] against LWE-based cryptography and NTRU, by targeting the multiplication of the ciphertext with the secret during decryption (line 3 of algorithm 2.8), or during signature generation (line 17 and line 18 of algorithm 2.17). The secrets involved in these schemes have small-valued coefficients, a significant proportion of which are equal to zero. Consequently, assuming the ability for an attacker to reliably inject zeroing faults on individual coefficients of the secret, the knowledge of which faults are effective allows to determine which secret coefficients are zero. The authors thus report that the attack can decrease the security of the scheme, in *bikz*[4], by around 30 % and 45 % for Saber and NTRU respectively. A security decrease of 30 % to 45 % is reported for a variant of Dilithium using binomial rather than uniform sampling for the secrets. Finally, Kyber is shown as essentially immune to this attack, since its secret polynomials are converted into the NTT domain as part of key generation: the secret coefficients are thus spread over $[\![0, q_{\mathrm{K}} - 1]\!]$, with a negligible proportion of zeros.

In terms of countermeasures, Bettale *et al.* propose three solutions to prevent an attacker from determining which coefficients are zero [BMR21]:

- for schemes compatible with NTT multiplication, keeping the secrets in the NTT domain at all times, similarly to Kyber;

[4]The *bikz* unit refers to the block size needed to solve the problem instance with BKZ reduction; one *bikz* roughly corresponds to 0.265 bits of security.

- employing shuffling, so that the attacker is not able to determine the position of the zero coefficients[5];
- masking the secret coefficients, to distribute the value of the shares uniformly in $[\![0, q-1]\!]$.

Note that usual countermeasures against data faults, such as duplication and checksums, tend to facilitate ineffective-fault attacks by increasing the attack surface.

3.4.2 Nonce-reuse attacks

A nonce-reuse attack consists in forcing the target device to reuse several times a quantity that is supposed to be used only once. This can prove extremely effective to reduce the security of a scheme, as shown by Ravi *et al.* against LWE-based schemes [RRB+19]. Since these schemes crucially rely on error sampling for security, tampering with this sampling makes them vulnerable to attacks.

Practically, the attack of Ravi *et al.* relies on the form of LWE instances, $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$, where $\mathbf{s}$ and $\mathbf{e}$ are small errors sampled *independently* at random, and both $\mathbf{A}$ and $\mathbf{t}$ are publicly known. Breaking this independence assumption, and forcing the two errors to be equal, instead yields the equation $\mathbf{t} = (\mathbf{A} + \mathbf{I})\mathbf{s}$ ($\mathbf{I}$ being the identity matrix), which can trivially be solved for $\mathbf{s}$ [RRB+19].

This attack readily applies to the key generation of Kyber, Dilithium, as well as other schemes such as round-1 FrodoKEM[6] [NAB+17] and round-2 NewHope [PAA+19], since they sample secret $\mathbf{s}$ and error $\mathbf{e}$ from a common large (e.g., 256-bit) secret seed followed by a short (one or two bytes) nonce in the form of an incremented integer (see lines 9 to 12 of algorithm 2.6 and lines 9 to 12 of algorithm 2.16). Thus, preventing the update of this nonce makes the two quantities equal. A complication in the case of Dilithium and Kyber is that, since the secret and error are in fact vectors of polynomials, the nonce is actually incremented several times: once for each polynomial, which may require the injection of several consecutive faults, to nullify all increment operations [RRB+19]. It is important to notice that the resulting key pair is valid although insecure, so that the tampering will remain unnoticed.

The authors note that for Dilithium, the full value of $\mathbf{t}$ is not actually made public: only its high bits, $\mathbf{t}_1$. While this complicates key recovery, they suggest that the attack is still valid, since $\mathbf{t}_0$ is not considered secret, and is progressively revealed through released signatures [RRB+19].

For the lowest security level of round-3 Kyber and ML-KEM, the attack does not actually make $\mathbf{s}$ and $\mathbf{e}$ equal, since they are sampled from binomial distributions with different widths. However, it should still be possible to exploit the strong correlation between the two. Furthermore, if the same nonce is used for the $k = 2$ polynomials of

[5] This assumes that merely leaking the number of zeros in the secret has no importance; while this number can only reveal a few bits of information on the secret key, it is not strictly zero.

[6] Starting from round 2, FrodoKEM changes its way of expanding the secret and error, instead sampling both of them consecutively from a single XOF call on the secret seed. This change was explicitly made to prevent the nonce-reuse attack described here [NAB+19].

each vector, then the LWE problem is effectively of size $n = 256$ rather than $kn = 512$, which more or less halves its bits of security.

Furthermore, Ravi *et al.* show how their attack can be adapted for message recovery on Kyber (or other LWE-based KEMs). The idea of the fault attack is exactly the same, namely making vectors $\mathbf{r}$ and $\mathbf{e}_1$ equal at line 7 and line 10 of algorithm 2.7. The resulting $(\mathbf{A}, \mathbf{u})$ is then an insecure LWE instance, which allows the message to be recovered. Let us call the target of this attack Alice and her message m_{Alice}. If the attacker stops at this point, the other party (Bob) will detect during decapsulation that the ciphertext has been altered; the authors thus propose to complement this fault-injection attack with a MITM attack, pictured in fig. 3.1. The attacker, Mallory, opens a genuine connection with Bob using message m_{Mallory}. She now shares secret ss_{Mallory} with Bob (through her genuine connection) and a different one, ss_{Alice}, with Alice (thanks to the message-recovery attack). She can thus intercept messages between Alice and Bob, decrypt them, and re-encrypt them using the secret shared with the other party. Since Alice encapsulated her secret with Bob's authentic public key, and no cryptographic authentication usually exists in the other direction, Mallory's MITM attack will go unnoticed.

3.4.3 Instruction-skipping faults

Apart from interfering with conditional structures, fault injection can prevent whole sub-operations from being executed, e.g. through skipping software function calls, or manipulating the finite-state machines in hardware implementations [NUN+23].

Bauer *et al.* [BDS23] employ such faults against Dilithium, to make the target device accept a forged signature as valid. The soundness of Dilithium lies in the following: an adversary choosing $\mathbf{w} \in R_{q_D}^k$ and computing $c = \text{SampleInBall}(\text{SHAKE256}(\mu \parallel \text{HighBits}(\mathbf{w})))$ is unable to find a small $\mathbf{z} \in R_{q_D}^\ell$ such that $\mathbf{w}'$ in eq. (3.2) is sufficiently close to $\mathbf{w}$.

$$\mathbf{w}' = \mathbf{A}\mathbf{z} - c\mathbf{t}_1 \cdot 2^d \tag{3.2}$$

However, this task is trivial if the subtraction of $c\mathbf{t}_1 \cdot 2^d$ is skipped, since the attacker can choose $\mathbf{z}$ first, and only then compute $\mathbf{w} = \mathbf{A}\mathbf{z}$. Thus, signatures forged in this way are made accepting by faulting the target device during signature verification, so as to skip the subtraction of $c\mathbf{t}_1 \cdot 2^d$ at line 9 of algorithm 2.18.

A similar effect has been achieved by Calle Viera *et al.* [CVBH24] in two additional ways: skipping the computation of c altogether, so that an all-zero polynomial is used

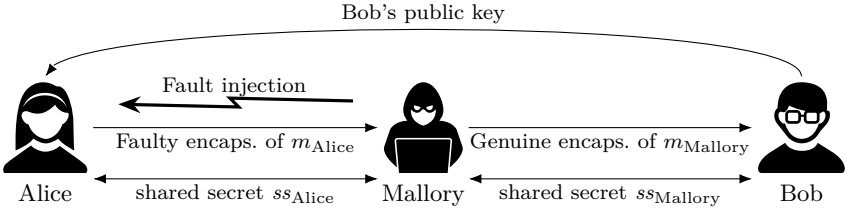


Figure 3.1: Fault-assisted MITM attack on KEMs, described by Ravi *et al.* [RRB+19]

instead, and skipping the multiplication by 2^d, so that $c\mathbf{t}_1$ does not influence the high bits of the result.

Countermeasures Several countermeasures against instruction-skipping fault attacks have been proposed; their common point is to modify the structure of the computations in such a way that skipping part of them necessarily results in rejecting the signature. Bauer *et al.*, for instance, propose to add a random polynomial $\mathbf{u}$ to both terms of the subtraction before the computation of eq. (3.2), giving $\mathbf{w}' = (\mathbf{Az} + \mathbf{u}) - (c\mathbf{t}_1 \cdot 2^d + \mathbf{u})$. Skipping the subtraction thus makes $\mathbf{u}$ remain, which makes signature verification fail. More simply, Calle Viera *et al.* suggest writing $\mathbf{w}'$ to the location initially occupied by $c\mathbf{t}_1 \cdot 2^d$. In this way, if the subtraction is skipped, the subsequent computation step will operate on $c\mathbf{t}_1 \cdot 2^d$ and the verifier will not compute the same commitment as the attacker, consequently rejecting the signature. They also show that the multiplication by 2^d can be protected by rewriting eq. (3.2) as $\mathbf{w}' = (\mathbf{Az} \cdot 2^{-d} - c\mathbf{t}_1) \cdot 2^d$, again making $\mathbf{w}'$ entirely incorrect if this multiplication is skipped. The authors furthermore suggest checking the coefficient distribution, or at least the norm, of $c\mathbf{t}_1 \cdot 2^d$, to make sure it has the expected form.

3.5 Generic pre- and post-processing techniques for attacks

3.5.1 Exploiting the ciphertext malleability of LWE-based encryption

LWE-based encryption essentially adds pseudorandom-looking noise to the message to be encrypted, where the added noise can only be determined (approximately) by the holder of the secret key. Consequently, given a ciphertext $ct = (\mathbf{u}, v)$ that decrypts to m', an adversary can easily construct a ciphertext $\widetilde{ct}$ that decrypts to $\widetilde{m}' = m' + X^i$ by choosing $\widetilde{ct} = (\mathbf{u}, v + \lfloor q/2 \rceil X^i)$. Ravi *et al.* [RBRC22] use this property to build a message-recovery attack from a classifier that is only able to recover the Hamming weight of single bytes of the decrypted message: by successively flipping each bit of m' and checking whether the Hamming weight increases or decreases, the value of each message bit can be determined. Since this can be applied in parallel to all message bytes, nine traces are enough to recover the whole message.

3.5.2 Lattice-reduction post-processing

If the above attacks do not yield the targeted secret in whole, a final step of lattice reduction can be used for full recovery. Indeed, lattice reduction has unreasonable cost on the schemes in a black-box setting, but partial information on the secrets progressively eases the corresponding lattice-reduction problem. Thus, several attacks propose using lattice reduction to increase the success probability or decrease the cost of the attack [NDGJ21, PP21, HPP21, PP19b, MUTS22, PPM17].

Even if a side-channel attack only manages to get very partial secret information, this knowledge may still be beneficial to the attacker: indeed, it has been shown that this information can be taken into account into a cryptanalytic attack, by using these hints

to ease the process of lattice reduction for solving LWE. Dachman-Soled *et al.* thus show how to estimate the cost of solving this type of LWE-with-hints problems [DDGR20].

3.6 Masking against vertical side-channel attacks

Like other cryptosystems, lattice-based cryptography is vulnerable to vertical side-channel attacks, which consist in observing several cryptographic executions using the same key, and deriving secret information on this key from the side-channel leakage over all these executions. A well studied and provably secure countermeasure against these is *masking*, which we briefly introduce here in a general context, before examining how it has been used in lattice-based cryptography.

3.6.1 Principles of masking

The *masking* countermeasure was first proposed by Chari *et al.* [CJRR99], using the previously known method of *secret sharing*[7]. To make sure an attacker cannot determine it, secret data is split into several *shares*, each being chosen at random with a distribution independent from the secret, but such that all shares can be recombined to determine the value of the secret. This recombination, however, does not happen during computations: instead, the operations performed by the scheme are adapted to work on the individual shares without mixing them.

Chari *et al.* proposed a first proof of soundness of this countermeasure, showing that increasing the number of shares allowed for an exponential decrease of the secret information leaked through noisy measurements of the share values. Prouff and Rivain [PR13] extended this result to acknowledge the fact that the *computations* performed on the shares also leak information. In parallel, several works examined the accuracy of various leakage models. Ishai, Sahai and Wagner [ISW03] studied masked circuits in the context of the *t-threshold probing model*, which assumes that an attacker can place up to t probes on internal variables of the target circuit and monitor their values. For hardware implementations, this model was found insufficient in practice, due to the presence of *glitches* and *transitions*, two physical imperfections that can render a circuit insecure if not properly considered. As a consequence, Faust *et al.* [FGP$^+$18] included these two behaviors into the *robust-probing* model, together with a third one: *coupling*, which aims to express that adjacent wires and logic gates in a circuit tend to influence one another, thereby leaking their values not only individually, but also jointly. While this latter defect is difficult to model without knowing the precise characteristics of a hardware implementation, the first two can be accounted for even in the early stages of designing a secure circuit.

[7] Secret sharing was initially introduced as a way to split a secret among several parties such that no incomplete subset of these participants, or no subset of less than t participants for threshold secret sharing, have any information on the secret. See, *e.g.*, [Sha79, Bla79].

3.6.1.1 Masked gadgets

To simplify their analysis, masked circuits are built as the assembly of *gadgets*, that is, sub-circuits realizing masked elementary functions: for instance, a masked two-input AND gate. Within gadgets, operations that mix input shares having the same index are said to be *same-domain* operations, while the operations that involve input shares with different indexes are referred to as being *cross-domain*.

3.6.1.2 Masking schemes

In the following, we will mention two types of masking, which we refer to as Boolean masking and arithmetic masking. We will furthermore distinguish between power-of-two arithmetic masking and prime arithmetic masking, since they are treated in a somewhat different manner. The type of masking that we use will be referred to as a *masking scheme*, which should be chosen based on the main types of operations to be performed on the masked data.

Boolean masking In w-bit Boolean masking, we consider shares as part of the finite group $(\mathbb{F}_2^w, \oplus)$, where $\oplus$ represents bit-wise exclusive OR. That is, a set of shares $(s^{\mathbf{0}}, \ldots, s^{\boldsymbol{d}})$, where each $s^{\boldsymbol{i}}$ is an element of $\mathbb{F}_2^w$, represents $s = \bigoplus_{\boldsymbol{i}=0}^{d} s^{\boldsymbol{i}} \in \mathbb{F}_2^w$.

This masking scheme is well adapted to the protection of Boolean arithmetic: the exclusive-OR operation, being linear in the considered group, can be implemented in a share-wise manner. Bit-wise NOT, being equivalent to the Boolean addition of the all-ones vector, can be applied by negating a single share. The AND operation, instead, needs a more complex gadget involving cross-domain computations, since it is nonlinear.

Arithmetic masking Arithmetic masking modulo m instead considers the additive finite group $(\mathbb{Z}/m\mathbb{Z}, +)$. If m is a power of two or a prime number, we will speak of power-of-two arithmetic masking and prime arithmetic masking respectively. This time, a set of shares $(s^{\mathbf{0}}, \ldots, s^{\boldsymbol{d}})$, where each $s^{\boldsymbol{i}}$ is an element of $\mathbb{Z}/m\mathbb{Z}$, represents $s = (\sum_{\boldsymbol{i}=0}^{d} s^{\boldsymbol{i}}) \bmod m \in \mathbb{Z}/m\mathbb{Z}$.

We note that arithmetic masking is best suited to performing arithmetic operations modulo m: modular addition, as well as multiplication by a constant, are linear and can be computed share-wise. The addition of a constant is accomplished by adding the same constant to a single share of the masked value. Finally, the multiplication of two sharings is a nonlinear operation that involves cross-domain terms, which must be implemented with care to avoid a loss of security.

3.6.2 Generic masked operations

Before going into the details of each scheme, we describe the generic masked operations that have been published in the literature, and that apply both to Kyber and Dilithium. Both schemes, since they use a mix of arithmetic and logic operations, need conversions from Boolean to arithmetic masking (B2A) and from arithmetic to Boolean (A2B). A

way to implement these conversions is secure addition over Boolean shares, developed in chapter 5 in the context of one of our contributions.

3.6.2.1 Conversions between Boolean and arithmetic masking

Conversions between Boolean and arithmetic masking were first introduced by Goubin [Gou01], under the names BooleanToArithmetic and ArithmeticToBoolean, for first-order masking only. The first one relies on function $r \mapsto ((x' \oplus r) - r) \bmod 2^K$ being affine over $\mathbb{F}_2^K$ for any $x' \in \mathbb{F}_2^K$. Based on this property, Goubin's algorithm takes as input a K-bit Boolean sharing $(b^{\mathbf{0}}, b^{\mathbf{1}}) \in \mathcal{B}(\mathbb{F}_2^K)$, and computes the corresponding arithmetic sharing $(a^{\mathbf{0}}, a^{\mathbf{1}}) \in \mathcal{A}_{2^K}(\mathbb{Z}/2^K\mathbb{Z})$, where $a^{\mathbf{0}} = (b^{\mathbf{0}} \oplus b^{\mathbf{1}}) - b^{\mathbf{1}}$ and $a^{\mathbf{1}} = b^{\mathbf{1}}$. The computation is done according to algorithm 3.1.

Algorithm 3.1: Goubin's first-order B2A conversion

Input: Boolean sharing $(b^{\mathbf{0}}, b^{\mathbf{1}}) \in \mathcal{B}(\mathbb{F}_2^k)$
Output: Arithmetic sharing $(a^{\mathbf{0}}, a^{\mathbf{1}}) \in \mathcal{A}_{2^K}([\![0, 2^K - 1]\!])$, such that $b^{\mathbf{0}} \oplus b^{\mathbf{1}} \equiv a^{\mathbf{0}} + a^{\mathbf{1}} \pmod{2^K}$

$\Gamma \xleftarrow{\$} \mathbb{F}_2^K$
$a^{\mathbf{1}} = b^{\mathbf{1}}$
$T = b^{\mathbf{0}} \oplus \Gamma$
$T = T - \Gamma$
$T = T \oplus b^{\mathbf{0}}$
$\Gamma = \Gamma \oplus b^{\mathbf{1}}$
$a^{\mathbf{0}} = b^{\mathbf{0}} \oplus \Gamma$
$a^{\mathbf{0}} = a^{\mathbf{0}} - \Gamma$
$a^{\mathbf{0}} = a^{\mathbf{0}} \oplus T$
return $(a^{\mathbf{0}}, a^{\mathbf{1}})$

The conversion proposed by Goubin for the other direction, from arithmetic to Boolean masking (still at the first order only), involves a more expensive computation, with a number of bit operations that is quadratic in K instead of linear [Gou01].

Arithmetic-to-Boolean conversions with lower complexity in software have been proposed by Coron and Tchulkine [CT03] (later corrected by Debraize [Deb12]) as well as Neiße and Pulkus [NP04]. Both methods are table-based, with two steps: a precomputation step, in which one or two masked lookup tables are freshly generated, and a conversion step, in which the conversion is done one digit at a time through successive table lookups. The digit size is chosen so as to minimize the overall computation time, since the lookup table must be recomputed with a fresh mask before each conversion. Table-based methods are however difficult to use in hardware, due to the need for storing the tables in a dedicated memory, and the difficulty to deal with glitches securely: hardware implementations of these methods do not thus seem to be documented in the literature[8].

Alternately, it is possible to implement the Boolean-to-arithmetic and arithmetic-to-Boolean conversions by using as a subroutine the operation of secure addition over

[8] The insecurity of these methods against glitches is mentioned, without further reference, in [FBR+22].

Boolean shares (SecAdd) [CGV14]. This technique has three major advantages: it can be easily adapted to any masking order (by adapting the order of the internal SecAdd operation), it allows for resource reutilization since the same SecAdd operation provides for both directions of conversion as well as other operations, and very importantly, it is applicable to prime arithmetic masking, as needed for Kyber, Dilithium and other lattice-based schemes [FBR+22, BBE+18, AEV24].

We show in algorithms 3.2 and 3.3 the implementation of the conversions for prime modulus, A2B_q and B2A_q, for first-order security. These descriptions are adapted from Fritzmann *et al.* [FBR+22], together with our optimization from [AEV24] using secure subtraction over Boolean shares (SecSub) instead of secure addition. The same conversions can be obtained for power-of-two arithmetic masking by replacing modular addition and subtraction with standard SecAdd and SecSub, and adapting the sampling of the random refresh bits. The techniques for the implementation of secure modular addition and subtraction are detailed in chapter 5.

3.6.2.2 Single-bit or few-bit Boolean-to-arithmetic conversion

Schneider *et al.* [SPOG19] propose a B2A conversion specialized to single-bit values, but that works for any arithmetic modulus q. Their technique hinges on the relation

$$\forall a, b \in \{0,1\} \quad a \oplus b = a + b - 2ab\,,$$

which allows to express Boolean addition in terms of arithmetic operations. Then, for first-order B2A conversion of $(x^{\mathbf{0}}, x^{\mathbf{1}}) \in \mathcal{B}^2(\mathbb{F}_2)$, they start by splitting the first share into a new arithmetic sharing $b^{\mathbf{0}}, b^{\mathbf{1}} \in \mathbb{Z}/q\mathbb{Z}$ such that $x^{\mathbf{0}} \equiv b^{\mathbf{0}} + b^{\mathbf{1}} \pmod q$, and similarly for the second share, $x^{\mathbf{1}} \equiv c^{\mathbf{0}} + c^{\mathbf{1}} \pmod q$. They can then compute an arithmetic sharing of x as $(a^{\mathbf{0}}, a^{\mathbf{1}})$ where

$$\begin{aligned} a^{\mathbf{0}} &= (b^{\mathbf{0}} + c^{\mathbf{0}} - 2b^{\mathbf{0}}c^{\mathbf{0}} + r - 2b^{\mathbf{0}}c^{\mathbf{1}}) \bmod q\,, \\ a^{\mathbf{1}} &= (b^{\mathbf{1}} + c^{\mathbf{1}} - 2b^{\mathbf{1}}c^{\mathbf{1}} - r - 2b^{\mathbf{0}}c^{\mathbf{1}}) \bmod q\,, \end{aligned}$$

Algorithm 3.2: Mod-q-Arithmetic to Boolean conversion (A2B_q) using SecAdd_q, from [FBR+22]

Input: $a^\star \in [\![0, q-1]\!]^2$
Output: $b^\star \in \mathcal{B}[\![0, q-1]\!]$ with $b_0 \oplus b_1 = a_0 + a_1 \bmod q$

1 $m_0 \xleftarrow{\$} \mathbb{F}_2^n \qquad m_1 \xleftarrow{\$} \mathbb{F}_2^n$
2 $augend^\star = a_0 \oplus m_0, m_0$
3 $addend^\star = a_1 \oplus m_1, m_1$
4 $b_0, b_1 = \text{SecAdd}_q(augend^\star, addend^\star)$
// At this point $b_0 \oplus b_1 = a$
5 **return** b_0, b_1

Algorithm 3.3: Boolean to Mod-q-Arithmetic conversion (B2A_q) using SecSub_q, adapted from [AEV24]

Input: $b^\star \in \mathcal{B}[\![0, q-1]\!]$
Output: $a^\star \in [\![0, q-1]\!]^2$ with $a_0 + a_1 \bmod q = b_0 \oplus b_1$

1 $r \xleftarrow{\$} [\![0, q-1]\!] \qquad m \xleftarrow{\$} \mathbb{F}_2^n$
2 $addend^\star = r \oplus m, m$
3 $s_0, s_1 = \text{SecSub}_q(b^\star, addend^\star)$
// At this point $s_0 \oplus s_1 = (b - r) \bmod q$
4 $a_0, a_1 = s_0 \oplus s_1, r$
5 **return** a_0, a_1

$r \in \mathbb{Z}/q\mathbb{Z}$ being a random refresh mask, needed for the security of the multiplication.

The higher-order conversion uses the same technique to add the Boolean input sharing, one share at a time, into an accumulator represented as arithmetic shares. The number of shares of the accumulator is progressively increased to match the number of input shares that it represent. We refer the reader to [SPOG19] for the detailed algorithm and the proof of security based on the SNI (strong non-interference) notion.

The authors then propose a new $\mathrm{B2A}_q$ conversion for k-bit Boolean sharings, by iterating their single-bit $\mathrm{B2A}_q$ operation k times and multiplying each converted bit by its binary weight. Under the assumption that Boolean operations, modular addition and modular multiplication have the same unit cost, their $\mathrm{B2A}_q$ algorithm is significantly more efficient than the solution of Barthe *et al.* [BBE+18] based on SecAdd_q, particularly so for small values of k. This cost model is however debatable for hardware implementations, in which modular multiplication typically requires a much larger area than bitwise Boolean operations.

3.6.3 Masked implementation of Kyber

Many side-channel countermeasures have been proposed for Kyber, with varying degrees of effectiveness. In this subsection, we recall the masked implementations that are known in the literature, and focus on state-of-the-art masking of the nonlinear operations.

3.6.3.1 Sensitivity analysis

Since the main point of masking is the protection of long-term secrets against vertical side-channel attacks, the most crucial operation to protect in Kyber is its decapsulation. Indeed, key generation is usually only performed once, so its main vulnerability regarding side-channel attacks lies in single-trace attacks, against which masking is an inadequate countermeasure.

We reproduce in fig. 3.2 the sensitivity analysis of Kyber decapsulation by Bos *et al.* [BGR+21]. Note that, even if one decides to mask key generation and encapsulation, no additional types of low-level operations need to be masked since decapsulation already includes all low-level operations of the scheme. We notice that almost all operations need to be masked, the only exceptions being the decompression of the ciphertext $(\mathbf{u}, v)$, the computation of its digest (through H in the figure) and of the NTT of its first part, as well as the final key derivation function. The authors also choose not to protect the implicit-rejection seed, z, as well as the selection between z and $\bar{K}'$ for key derivation. In the presence of side-channel attacks, or once z has been determined through side-channel attacks, this makes Kyber explicitly rejecting, which should not practically downgrade its security [BGR+21].

The following operations thus need to be protected, the last bullet point grouping linear arithmetic operations, which can be computed share-wise on arithmetic sharings:

- Hash functions SHA3-512 and SHAKE256 (respectively G and PRF in the figure), which are not detailed here,

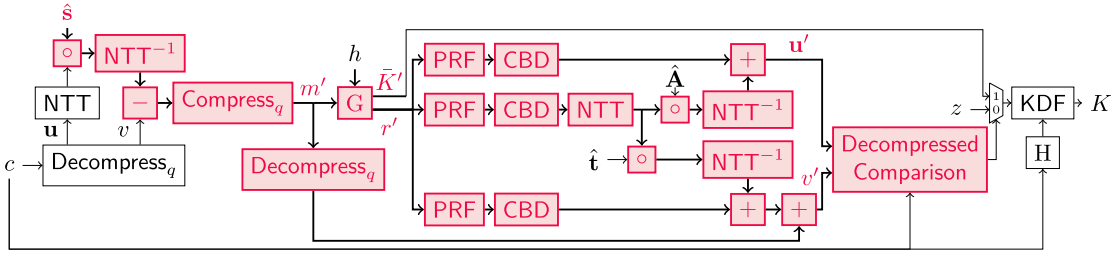


Figure 3.2: Sensitivity analysis of Kyber decapsulation, adapted from [BGR+21]. Sensitive operations and intermediate variables are highlighted in red.

- Message encoding and decoding (Compress_q and Decompress_q in the figure),
- Centered binomial sampling (denoted by CBD in the figure),
- Comparison of the received and reencrypted ciphertexts,
- Forward and inverse NTT, point-wise multiplication between a secret and a public input, polynomial addition and subtraction.

3.6.3.2 Masked message decoding

The message-decoding operation, also referred to as compression to 1 bit (eq. (2.7)), is a nonlinear operation that takes as input a polynomial coefficient $c \in [\![0, q-1]\!]$ and outputs

$$\text{Compress}_1(c) = \begin{cases} 0 & \text{when } c \in]-q/4, q/4[\pmod q, \\ 1 & \text{when } c \in]q/4, 3q/4[\pmod q\,. \end{cases}$$

Since this operation is not linear, it cannot be applied to first-order arithmetic shares $c^{\mathbf{0}}, c^{\mathbf{1}}$ independently. However, Reparaz *et al.* [RRVV15] show that, by splitting the range $[\![0, q-1]\!]$ into four equal quadrants (up to the rounding error) and determining which quadrant each share belongs to, we can conclude on the output of Compress_1 in half of the cases: when the shares belong to consecutive quadrants (see fig. 3.3). To perform the decoding in the other cases (when the two shares are in equal or opposite quadrants), the value is re-shared as $(c^{\mathbf{0}} + \Delta) \bmod q, (c^{\mathbf{1}} - \Delta) \bmod q$ (which still represents the same value), for different values of Δ until the result is no longer indeterminate. Since LWE decryption has some inherent failure probability, getting incorrect outputs from the decoder from time to time is not truly problematic, as long as the added error probability is negligible. Furthermore, due to the shape of added LWE noise, some unmasked values are more likely than others. These two remarks allows to limit the number of iterations of the above process to a reasonable value. The authors show that with a 13-bit modulus $q = 7681$, sixteen iterations are enough so that the overall decryption-failure probability rises by less than 20 %; however, this assumes an already very high decryption-failure probability around 2^{-15}. The case of current LWE schemes' (including Kyber's) negligible failure probability must be evaluated separately. The authors also

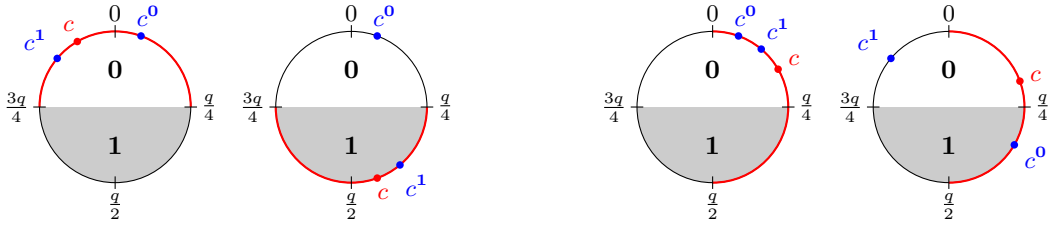


Figure 3.3: Graphical representation of the masked decoder of Reparaz *et al.* [RRVV15]. Input coefficients are represented along the circle. The upper, white half is decoded to 0 while the lower, gray one is decoded to 1. The first two diagrams represent shared coefficients that can be decoded share-wise, while share-wise decoding of the last two is insufficient to conclude.

note that their approach to masked decoding scales well with the masking order, since the decoding is performed after the computation of the quadrants, the latter being done on each share separately.

An improvement to this masked decoder is proposed by Zijlstra *et al.* [ZBT19], where the sixteen iterations of adjusting the shares with $\pm\Delta$ are replaced with a single one using the optimal value for Δ, derived from the distance of each share to the frontiers of the quadrants. The two distances are then compared — using lookup tables to avoid a leak, since the difference between the distances depends directly on part of the unmasked value — to determine the decoded output. This output is masked with a random input bit, to be fed to masked message re-encryption. The authors list four advantages to their new masked decoder with respect to that of Reparaz *et al.*: lower area, faster operation (there are no longer sixteen iterations), and as a consequence of the second, elimination of the DPA target constituted by the sixteen iterations; additionally, the new masked decoder is exact, and does not increase the failure probability of decryption. However, they give no security analysis of their solution, so more work would be needed on this point.

Oder *et al.* [OSPG18] employ a slightly more general technique, based on modulus switching. The masked input coefficient, $c^\star$, is first shifted by $-q/4$, so that the decoding boundaries now lie at 0 and $q/2$. Then, they change the arithmetic modulus of the sharing from q to 2^w, where $2^w > 2q$. This modulus switch is done with an A2B conversion and several arithmetic operations, giving a new sharing $d^\star$ modulo 2^w. Then, d is shifted by $-q/2$, so that the decoding boundaries are now at 0 and 2^{w-1}. Extracting the sign bit of d thanks to an A2B conversion thus gives the desired result. The authors provide a proof of security of their masked decoder at the first order.

Fritzmann *et al.* [FBR+22] simplify the technique of Oder *et al.* by reformulating the ad-hoc modulus-switching operation into a standard A2B_q operation, which simplifies the algorithm, gives it a security proof at any order, and allows to use one less bit when converting to the Boolean domain. The authors' illustration of how their decoder works is reproduced in fig. 3.4.

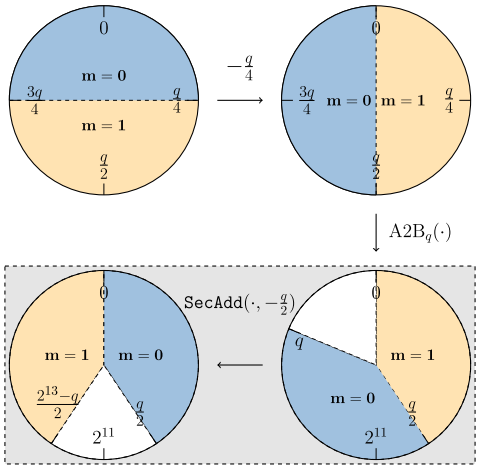


Figure 3.4: Graphical representation of the masked decoder of Fritzmann *et al.* [FBR+22] (minor adjustement of original authors' illustration). Input coefficients are represented along the circle. The colored sectors represent the decoded result, white being unused. The bottom part, highlighted in gray, uses Boolean masking.

Finally, message decoding can be implemented exactly as ciphertext compression to 1 bit, without specializing the operation. We discuss below, in paragraph *Masked ciphertext compression*, how this compression is masked in the general case.

Note that a large part of the complexity of masked message decoding for Kyber is due to the use of a prime modulus: indeed, this operation is much easier to implement with Saber's power-of-two modulus, as demonstrated by Van Beirendonck *et al.* [VBDK+21] as well as Fritzmann *et al.* [FBR+22]: after shifting the input coefficient by $q/4$, it can be decoded by simply examining its most significant bit, which is obtained with an A2B conversion.

3.6.3.3 Masked message encoding

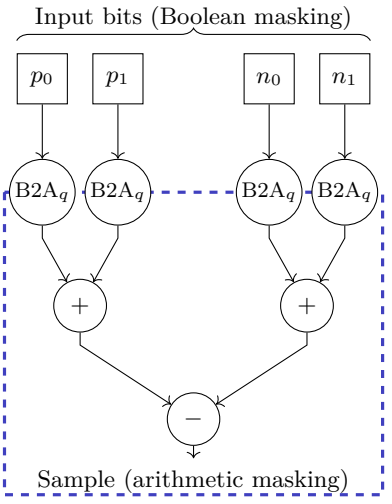
Message encoding is the reverse operation, whereby each bit of the message is transformed into a coefficient among $\{0, \lfloor q_\mathrm{K}/2 \rceil\}$ depending on its value. In unprotected implementations, this operation over bit b can be simply expressed as $\mathrm{Decompress}_1(b) = b\lfloor q_\mathrm{K}/2 \rceil$. However, since the message to be encrypted is initially specified as Boolean shares, converting it to arithmetic shares modulo q_K is necessary before multiplication by $\lfloor q_\mathrm{K}/2 \rceil$. One-bit B2A operations from the literature can be used for this task.

A different approach is undertaken by Oder *et al.* [OSPG18] for first-order masking. Notice that, if the message is given as Boolean shares $b^{\mathbf{0}}, b^{\mathbf{1}}$ and each share is multiplied by $\lfloor q_\mathrm{K}/2 \rceil$, the result will be a correct arithmetic sharing of the encoded bit, except when the input sharing is $(1, 1)$, in which case the result is $\lfloor q_\mathrm{K}/2 \rceil (b^{\mathbf{0}} + b^{\mathbf{1}}) \equiv 1 \pmod{q_\mathrm{K}}$ when it should be 0. Consequently, the correct encoded message is given by $\lfloor q_\mathrm{K}/2 \rceil b^{\mathbf{0}} + \lfloor q_\mathrm{K}/2 \rceil b^{\mathbf{1}} - b^{\mathbf{0}} \cdot b^{\mathbf{1}}$. The multiplication of the two Boolean shares is performed in a secure manner by splitting each share into two fresh arithmetic shares modulo q_K.

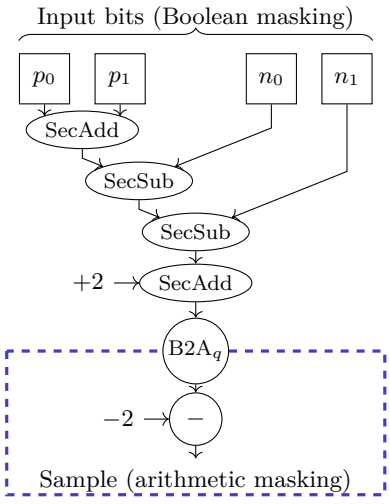
3.6.3.4 Masked binomial sampling

Binomial sampling consists in the addition of a few input bits, each sampled uniformly at random (see algorithm 2.4). More specifically, to obtain a centered binomial sample in $[\![-\eta, \eta]\!]$, two groups of η input bits are taken, the Hamming weight of each group is computed, and the difference of the two Hamming weights is determined. Since the input bits are generated by an XOF, masked implementations typically protect them as a Boolean sharing. However, the results of binomial sampling will all be involved in arithmetic computations, which require arithmetic masking for efficiency. Consequently, binomial sampling includes the Boolean-to-arithmetic conversion of the samples, either during the computation of the Hamming weights, or directly after.

Schneider *et al.* [SPOG19] propose both approaches and compare their efficiency in software. As a first approach, extending the work of Oder *et al.* [OSPG18], they convert each of the 2η input bits to an arithmetic sharing modulo q using their single-bit $\mathrm{B2A}_q$ conversion, and compute the difference of Hamming weights over the arithmetic sharings. In the second approach, they compute the difference of Hamming weights under a Boolean masking, using secure addition and secure subtraction over Boolean shares (SecAdd and SecSub), and convert the result to an arithmetic sharing modulo q. Both approaches are graphically represented in fig. 3.5. The main advantage of the second approach is that the computation of the Hamming-weight difference over Boolean shares can be expressed



(a) 2η single-bit $\mathrm{B2A}_q$ conversions followed by a difference of arithmetic sums



(b) Hamming-weight difference over Boolean shares, followed by a few-bit $\mathrm{B2A}_q$ conversion

Figure 3.5: Comparison of the two approaches proposed by Schneider *et al.* [SPOG19] for secure binomial sampling with $\eta = 2$. Operators $+$ and $-$ denote modular addition and subtraction. The boxed bottom part uses arithmetic masking modulo q while the upper part uses Boolean masking.

as a short sequence of Boolean operations. In a software implementation, this allows to parallelize the computation over several samples through bitslicing.

The authors show that their second approach outperforms the first by a large margin in the case of NewHope, for $\eta = 8$. While they do not report the obtained cycle counts for Kyber ($\eta \in \{2, 3\}$), the second approach should also be preferable in this case since it specifically outperforms the first for low values of η [SPOG19].

In a similar manner, Fritzmann *et al.* [FBR+22] compute the Hamming-weight difference over Boolean shares, using a tree of half adders implemented in a tightly-coupled hardware accelerator. By employing bitslicing, they are able to compute 32 binomial samples in parallel: a major advantage of hardware implementations in this respect is the ease of reordering the bits before and after the bitsliced operation, since it consists in a simple rewiring.

3.6.3.5 Masked ciphertext comparison

At the end of decapsulation, the equality between the received ciphertext ct and the reencrypted ciphertext ct' must be checked (see line 4 of algorithm 2.11). Two main approaches have been proposed for the protection of this operation. The first one consists in masking the ciphertext-compression operation and doing the comparison on the compressed ciphertexts. The second approach involves comparing the ciphertexts in a hybrid manner, in which the re-encrypted ciphertext, which is masked, is kept uncompressed, while the received ciphertext (considered public) is uncompressed in a special way.

Masked ciphertext compression It seems that Fritzmann *et al.* [FBR+22] have been the first to propose a masked compression operation for prime moduli: indeed, this topic had been previously addressed for schemes employing a power-of-two modulus [VBDK+21], like Saber, but this work cannot be extended to prime moduli since it relies on a simple masked logical shifting operation. Kyber compression, instead, uses a rounded multiplication satisfying

$$\mathrm{Compress}_d(x) = \left\lfloor x \frac{2^d}{q_\mathrm{K}} \right\rceil \bmod 2^d .$$

Since the fractional part of $x\frac{2^d}{q_\mathrm{K}}$ is never exactly $\frac{1}{2}$, the computation of this rounded division can tolerate a small error, bounded by $\frac{1}{2q_\mathrm{K}}$, and still return a correct value. Fritzmann *et al.* use this property to compute the operation over Boolean shares: they reformulate compression as multiplication by a fixed-point approximation of $\frac{2^d}{q_\mathrm{K}}$, followed by truncation of the unneeded precision, as shown in algorithm 3.4. The number f of fractional digits for the fixed-point approximation depends on the masking order: for first-order masking, $f = 13$, and it grows by 1 for each doubling of the number of shares. A more detailed analysis of this method for high-order masking has been proposed by Coron *et al.* [CGMZ23].

Algorithm 3.4: Masked ciphertext compression from Fritzmann *et al.* [FBR+22]

Parameter: Number of bits d of the compressed coefficient, number of extra bits of precision f
Input: $x^\star \in \mathcal{A}_{q_K}(\mathbb{F}_{q_K})$
Output: $z^\star \in \mathcal{B}(\mathbb{b}^d)$ such that $\oplus_i z^i = \text{Compress}(\sum_i x^i \bmod q_K)$

$y^\star = \lfloor 2^{d+f} x^\star / q \rceil \in \mathcal{A}_{2^{d+f}}(\mathbb{Z}/2^{d+f}\mathbb{Z})$ // *Fixed-point rescaling with f fraction bits*
$y^0 = y^0 + 2^{f-1}$ // *Add $\frac{1}{2}$ for rounding*
$z^\star = \text{A2B}_{2^{d+f}}(y^\star)$ // *A2B conversion over $d+f$ bits*
$z^\star = z^\star \gg f$ // *Truncate the fractional part*
return $z^\star$

We propose a novel method for the masking of ciphertext compression, which we will detail in section 6.1.

Masked comparison of the compressed ciphertext Assuming a protected version of the ciphertext-compression operation exists, it can be used during the reencryption step of decapsulation to compute the shares of the compressed ciphertext. At this point, the only remaining task is to check whether the value represented by these shares is equal to the received ciphertext, which is not protected. Again, several methods have been described in the literature for this equality check.

Generic masked comparison over Boolean shares The straightforward solution for masked comparison, alluded to in [BDH+21], consists in subtracting the two values to be compared (using arithmetic or Boolean subtraction depending on their masking scheme), converting the difference to Boolean shares if needed, and performing a masked logical reduction to get a single masked bit indicating whether the difference is zero. Although this method is generic and applicable to any masking order, it does not seem to be described in detail in the literature.

Hash-based approach The solution chosen in several works [OSPG18, VBDK+21, FBR+22] is to perform a bitwise exclusive-or between the received ciphertext and one share of the reencrypted one, hash it with an unprotected hash, and compare the obtained hash with that of the other share. The collision resistance of the hash implies that the comparison passes only if the ciphertexts are equal, and its preimage resistance, together with the randomness of the shares, implies that no information on the reencrypted ciphertext can be retrieved by observing the comparison of the two hashes. It is extremely important to hash both ciphertext parts $\mathbf{u}$ and v into a single digest, since an adversary learning the check result for each part separately would be able to mount a decryption-failure oracle [BDH+21, VBDK+21]. This flaw was present, for instance, in [OSPG18].

Arithmetic approach Bache *et al.*[BPO+20], followed by Bhasin *et al.* [BDH+21], employ a radically different method for the comparison: they essentially evaluate a linear combination of the coefficients of ct' with random weights in $\mathbb{F}_q$ and compare it with the same linear combination of the coefficients of ct: the two results match when the

two polynomials are equal, except with a $1/q$ probability of a false match (declaring equality when the polynomials are different). An important limitation of this approach is that it requires the masked polynomial to employ a prime modulus q, which makes the comparison of the compressed ciphertexts more involved since they belong to a power-of-two range.

Where the two works differ most prominently is in the number and constitution of the linear combinations. Bache *et al.* split the ciphertext into groups of nonoverlapping coefficients which are separately compared in this way. However, Bhasin *et al.* show that this setup has a nonnegligible false-positive rate of $1/q$, independent of the number of groups, in the context of chosen-ciphertext attacks. To correct this vulnerability, they instead compare the whole ciphertext at once (so that a side-channel attacker cannot know whether the difference between the ciphertexts is localized or generalized), and repeat the comparison several times with different weights (to decrease the false-match probability).

D'Anvers *et al.* [DHP+22] further tweak this solution by instead choosing a larger value for q, to make the false-match probability acceptably low after a single such comparison. They thus employ a Boolean-to-arithmetic conversion to transform the shares of the reencrypted ciphertext from their initial Boolean masking into an arithmetic masking modulo this large q.

Hybrid comparison of compressed and uncompressed ciphertexts Bos *et al.* [BGR+21] choose not to provide a masked compression operation. During encapsulation, they compute ciphertext compression in an unprotected manner, by unmasking the ciphertext before compression. For Kyber-768 and Kyber-1024, this does not lead to a quantifiable decrease of security, since the ciphertext compression is not accounted for in the security analysis. For Kyber-512, instead, 6 bits of security are attributed to the presence of ciphertext compression, so revealing the uncompressed ciphertext may decrease the security of the scheme to slightly below the targeted value [SAB+20].

In the case of decapsulation, doing an unprotected ciphertext compression would lead to a complete loss of side-channel security due to the powerful oracle-based attacks it would enable (see section 3.3). Bos *et al.* circumvent the issue by decompressing the received ciphertext (which is not masked) instead of compressing the recomputed one, and doing the comparison on the decompressed values. However, since compression is a lossy operation, the decompression and the comparison need to be done in a special manner. Each coefficient c_i of the received ciphertext[9] is decompressed to the two values

$$\begin{aligned} a_i &= \min\{y \in \mathbb{F}_{q_\mathrm{K}} \mid \mathrm{Compress}(y) = c_i\} \quad \text{and} \\ b_i &= \max\{y \in \mathbb{F}_{q_\mathrm{K}} \mid \mathrm{Compress}(y) = c_i\} + 1\,, \end{aligned}$$

[9]To simplify the notation, we ignore here that ct is actually made of two parts $\mathbf{u}$ and v which are compressed to a different number of bits per coefficient. This inhomogeneity of the ciphertext is easy to manage, by decompressing each part with the appropriate parameter.

and the ciphertext comparison consists in checking whether for each coefficient c'_i of the reencrypted ciphertext, $a_i \leqslant c'_i < b_i$ holds. All of these inequalities are checked in a masked manner (masked inequality check is described in more detail for Dilithium, § 3.6.4.5), the results of all these checks are securely multiplied together, and only the final result is unmasked.

3.6.4 Masked implementation of Dilithium

Since Dilithium is already difficult to implement on resource-constrained devices due to its high memory usage and its resource-intensive computations, few protected implementations of the scheme have been described in the literature. In particular, there does not seem to be any dedicated analysis of masking Dilithium in hardware, except for some general considerations by Steffen *et al.* [SLKG23]. We will thus describe here the state-of-the art masking solutions in software.

3.6.4.1 Sensitivity analysis

We here recall the sensitivity of Dilithium intermediate variables to side-channel attacks. Since signature verification only involves publicly known material[10], only key generation and signature generation are studied here.

Key generation Azouaoui *et al.* propose in [ABC+23] a sensitivity analysis of Dilithium key generation, which we reproduce in fig. 3.6 with the updated symbol ρ' for the secret-key seed (compared to ς as in previous Dilithium specifications). In their analysis, the sensitive variables are ζ, ρ', K, $\mathbf{s}_1$ and $\mathbf{s}_2$. Indeed, $\mathbf{s}_1$ and $\mathbf{s}_2$ are crucial parts of the secret key, which must both be protected to prevent key recovery. Likewise, K is the secret seed used to derive random coins for signature generation in the deterministic variant of Dilithium: if this variable is revealed, then an adversary will be able to reconstruct the sensitive mask vector $\mathbf{y}$ used during signature generation. It is then straightforward to see that the seeds of these quantities, ρ' and ζ, must also be protected. In the other direction, the result of product $\mathbf{A}\mathbf{s}_1$ is still sensitive since the LWE error has not yet been added.

On the other hand, ρ, $\mathbf{A}$, $\mathbf{t}$, $\mathbf{t}_0$, $\mathbf{t}_1$ can be considered public: while $\mathbf{t}_0$ is not included in the public key, it is not considered sensitive in Dilithium security analysis [BDK+21].

The following operations must thus be protected, among which the last group, being linear, can simply be applied share-wise over arithmetic sharings:

- Hash SHAKE256, which is not detailed here,
- ExpandSi, which also involves the computation of an XOF,
- Forward and inverse NTT, addition and point-wise multiplication.

[10] Unless the signature pertains to sensitive data, in which case the computation of the message digest should be protected at line 6 of algorithm 2.18.

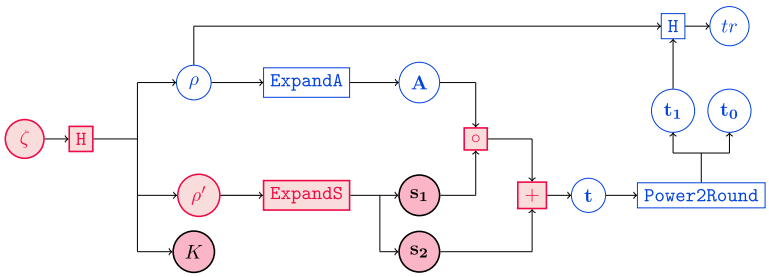


Figure 3.6: Sensitivity of variables in Dilithium key generation, adapted from [ABC+23]. Red circles (with filled background) are sensitive variables, blue ones are variables that do not need protection. Likewise, red and blue rectangles are sensitive and unprotected operations respectively.

Signature generation We also reproduce in fig. 3.7 the analysis of Azouaoui *et al.* regarding signature generation [ABC+23]. This analysis is slightly involved since Dilithium uses rejection sampling: several signature attempts are made, and only the successful one can safely be published: consequently, some quantities that may eventually be part of the signature must be kept secret until the signer has checked that the candidate signature respects the zero-knowledge property.

Let us first list which data can safely be revealed. First, tr, M, μ, ρ, $\mathbf{t}_0$ are trivially public since they are part of the public-key material, the public message, or are directly deducible from them. Furthermore, $\mathbf{w}_1$ can also safely be revealed even for aborted signatures: Azouaoui *et al.* argue that, together with $\mathbf{A}$, it essentially constitutes an LWR instance hiding $\mathbf{y}$. Since the rounding noise of this LWR problem has much larger amplitude (2^{12} rather than 2^1 or 2^2) than the noise used in the LWE problem Dilithium relies upon, it seems reasonable to consider the LWR problem harder than the LWE one. This analysis matches with the one of Coron *et al.* [CGTZ23], who furthermore point out that this heuristic assumption has recently been proved by Devevey *et al.* in a more generic context: since $\mathbf{w}_1$ is the commitment of a Fiat-Shamir signature scheme, it can be revealed even for aborted transcripts [DFP+24]. From this, it becomes clear that $\tilde{c}$ and c are not sensitive either, since they are computed exclusively from non-sensitive data.

After both norm checks on $\mathbf{z}$ and $\tilde{\mathbf{r}}$ have been verified, both of these quantities can safely be revealed, since the first will become part of the signature, and the second can be computed from non-sensitive values as $\tilde{\mathbf{r}} = \mathbf{A}\mathbf{z} - c\mathbf{t} - \alpha \cdot \mathbf{w}_1$. Provided they are only computed *after* both norm checks, $\mathbf{g} = c\mathbf{t}_0$ and $\mathbf{h} = \text{MakeHint}(\mathbf{g}, \mathbf{r})$ do not thus need protection. At this point, the signature may still be rejected if too many set hints are needed; however, this rejection only happens for efficiency reasons, not for security, so the signature could safely be released even if this check failed.

The remaining variables must then be protected: K, ρ', $\mathbf{y}$, $\mathbf{w}$, $\mathbf{w}_0$, as well as $\mathbf{z}$ and $\tilde{\mathbf{r}}$ until their norm has been successfully checked. Note that the signer can safely reveal which condition triggered rejection: for the norm checks, it is even acceptable to indicate

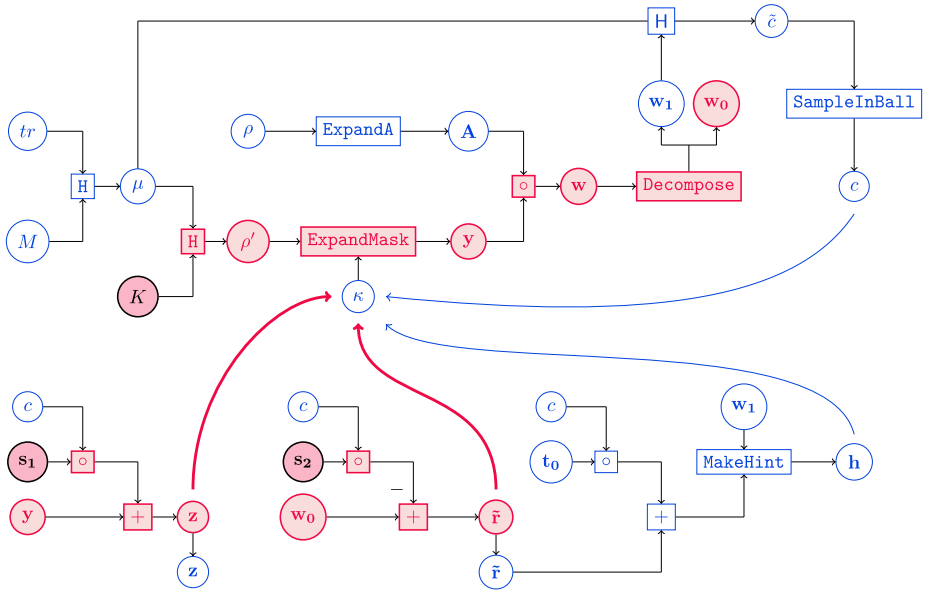


Figure 3.7: Sensitivity of variables in Dilithium signature generation, adapted from [ABC+23]. Red circles (with filled background) are sensitive variables, blue ones are variables that do not need protection. Likewise, red and blue rectangles are sensitive and unprotected operations respectively. Curved arrows represent the rejection-sampling conditions, still with the same color coding.

which coefficient index exceeded the bound [BDK+21], provided that the *exact value* and *sign* of this coefficient are not revealed. Therefore, beside those already used in key generation, the following operations need protection:

- ExpandMaski,
- Decompose, with the exception that the high-order output digit can be revealed,
- and the two norm checks $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ and $\|\mathbf{r}_0\|_\infty < \gamma_2 - \beta$.

3.6.4.2 Masked uniform rejection sampling

In general, masked uniform sampling is not difficult to implement, since it is sufficient to sample each share uniformly at random, and independently from each other. However, two obstacles arise when trying to use this method for the ExpandSi function. First, this function performs uniform sampling in $[\![-\eta, \eta]\!]$, but the following operations perform arithmetic computations modulo q_D on its result. Consequently, the sampling is not uniform over the whole modular-arithmetic range. The second obstacle arises when trying to implement a scheme that is not only interoperable with the specification of Dilithium (or of any other scheme for the matter), but that strictly adheres to it: indeed, secret-key expansion is defined as a specific mapping from a seed to the obtained secret key. Preserving the secret-key distribution while using a different mapping will lead to a

compatible[11], but non-compliant implementation. To the best of our knowledge, the first obstacle has been solved in several ways in the literature, but the second one remains.

Migliore *et al.* proposed a masked implementation of Dilithium as early as 2019 [MGTF19], while the second round of the PQC standardization process was barely starting. Their implementation of masked rejection sampling from $[\![-\eta, \eta]\!]$ is shown in a simplified manner in algorithm 3.5. The first step of their algorithm is a rejection loop (lines 1 to 5), that repeatedly samples fresh B-bit Boolean shares (B being the minimum number of bits needed to represent the sampling range), until the shares represent an integer strictly less than $2\beta + 1$. This condition is checked by subtracting $2\beta + 1$ using secure addition over Boolean shares, and checking the sign bit of the result. Having done this, the sampling procedure proceeds by folding $x \in [\![\beta + 1, 2\beta]\!]$ to $[\![-\beta, -1]\!]$. This is accomplished by comparing x with $\beta + 1$, and adding to it $q_\mathrm{D} - 2\beta + 1$ or 0 depending on the sign of the comparison. Finally, the obtained value is converted into Boolean shares modulo q_D using the $\mathrm{B2A}_{q_\mathrm{D}}$ gadget. Note that the proposal of Migliore *et al.* contains no explicit refreshing of the intermediate sharings, but some may be needed for security.

Algorithm 3.5: Masked rejection sampling in $[\![-\eta, \eta]\!]$ from [MGTF19]

Parameter: Rejection-sampling bound η, sample bit size $B = \lceil \log_2(2\beta + 1) \rceil$
Parameter: Word size $w \in \mathbb{N}$ such that $q_\mathrm{D} \leqslant 2^w$
Parameter: Masking order $d \geqslant 1$
Output: Uniform sample from $[\![-\eta, \eta]\!]$ as arithmetic shares $r^\star$ modulo q_D

1 **do**
2 **for** $\boldsymbol{i} = 0$ to d **do** $x^{\boldsymbol{i}} \xleftarrow{\$} \mathbb{b}^B$ . . . . . *// Sample each share uniformly at random*
3 $b^\star = \mathrm{SecAdd}(x^\star, -2\beta - 1)$
4 $b^\star = b^\star \gg (w - 1)$
5 **while** $\bigoplus_{\boldsymbol{i}} b^{\boldsymbol{i}} = 0$ . . . . . *// Retry as long as $x \geqslant 2\beta + 1$*
6 $b^\star = \mathrm{SecAdd}(x^\star, -\beta - 1)$ . . . . . *// Compare x with $\beta + 1$*
7 $b^\star = |_{i=0}^{w-1}(b^\star \gg i)$ . . . *// w-bit Boolean mask based on sign of b, i.e. based on whether $x < \beta - 1$*
8 $b^\star = \mathrm{SecAnd}(\mathrm{SecNot}(b^\star), q_\mathrm{D} - 2\beta - 1)$ *// SecNot absent from [MGTF19] but needed for correctness*
9 $x^\star = \mathrm{SecAdd}(x^\star, b^\star)$ . . . . . *// Add $q_\mathrm{D} - 2\beta - 1$ if $x \geqslant \beta + 1$*
10 $r^\star = \mathrm{B2A}_{q_\mathrm{D}}(x^\star)$
11 **return** $r^\star$

Another solution, proposed by Coron *et al.* [CGTZ23], samples uniform and independent arithmetic shares modulo $2\eta + 1$, and converts them to arithmetic shares modulo q_D using a table-based arithmetic-to-arithmetic conversion, from modulus $2\eta + 1$ to modulus q_D. Since the referenced source for this conversion [CGMZ22] does not include direct arithmetic-to-arithmetic conversion except for logical shifting over arithmetic shares, this operation is probably implemented as an $\mathrm{A2B}_{2\eta+1}$ conversion followed by $\mathrm{B2A}_{q_\mathrm{D}}$.

A third construction, shown in algorithm 3.6, has been proposed by Azouaoui *et al.* [ABC+23]. It is essentially a simplification of the technique of Migliore *et al.*, that performs the offsetting from $[\![0, 2\eta]\!]$ to $[\![-\eta, \eta]\!]$ *after* the conversion to arithmetic shares modulo q_D: thus, the complex process in lines 6 to 9 of algorithm 3.5 is replaced with

[11] Even compatibility may break if the secret key is expanded from its seed for each signing operation.

the arithmetic subtraction of η from the first share of r (line 7 of algorithm 3.6), which saves two costly SecAdd operations.

Algorithm 3.6: Masked rejection sampling in $[\![-\eta, \eta]\!]$ from [ABC+23]

Parameter: Rejection-sampling bound η, sample bit size $B = \lceil \log_2(2\eta + 1) \rceil$
Parameter: Word size $w \in \mathbb{N}$ such that $q_\mathrm{D} \leqslant 2^w$
Parameter: Masking order $d \geqslant 1$
Output: Uniform sample from $[\![-\eta, \eta]\!]$ as arithmetic shares $r^\star$ modulo q_D

1 **do**
2 **for** $\boldsymbol{i} = 0$ to d **do** $x^{\boldsymbol{i}} \xleftarrow{\$} \mathbb{b}^B$ . . . .// *The authors take this randomness from an input string*
3 $b^\star = \mathrm{SecAdd}(x^\star, 2^w - 2\eta - 1)$
4 $b^\star = b^\star \gg (w - 1)$
5 **while** $\bigoplus_{\boldsymbol{i}} b^{\boldsymbol{i}} = 0$ . . . . . // *Retry as long as* $x \geqslant 2\eta + 1$
6 $r^\star = \mathrm{B2A}_{q_\mathrm{D}}(x^\star)$
7 $r^{\mathbf{0}} = (r^{\mathbf{0}} - \eta) \bmod q_\mathrm{D}$ . . . . . .// *Offset* r *by* $-\eta$
8 // *Add line* $r^\star = (-r^\star) \bmod q_\mathrm{D}$ *here to make this sampling compliant with Dilithium for* $\eta = 2$
9 **return** $r^\star$

As alluded to, none of these constructions respects the particular mapping specified by Dilithium to expand the secret-key polynomials from their seed. Indeed, in the first two, the shares of the candidate sample are chosen independently, with no determined mapping from the input randomness to the value they represent. The solution of Azouaoui *et al.* could easily implement the specified sampling for $\eta = 4$ by adding the commented step at line 8 of algorithm 3.6, but case $\eta = 2$ is more complex since it involves reducing x modulo 5 (see line 7 of algorithm 2.13).

We will describe in section 6.2 how we implement masked uniform rejection sampling, in a way that perfectly complies with the specification of Dilithium and ML-DSA.

3.6.4.3 Masked sampling of the mask polynomials

Since the third round of the NIST PQC standardization, the coefficients of the mask polynomial vector $\mathbf{y}$ are sampled from $[\![-2^{\gamma_1} + 1, 2^{\gamma_1}]\!]$, which contains a power-of-two number of elements[12]. No rejection sampling is thus needed, which greatly simplifies the masking of this operation.

This sampling is implemented with the following two steps: first, each coefficient is sampled as Boolean shares, taken from the output data of a masked extendable output function. These Boolean shares are then converted to arithmetic shares modulo q_D thanks to $\mathrm{B2A}_{q_\mathrm{D}}$ [CGTZ23, ABC+23].

3.6.4.4 Masked decomposition

In addition to being nonlinear, Dilithium integer decomposition is specified as a somewhat irregular function, with a corner case that is handled separately from the general situation [BDK+21]. Due to this, the masked implementation proposed by Migliore *et al.*

[12] In previous versions, $\mathbf{y}$ had uniformly distributed coefficients from $[\![-2^{\gamma_1} + 1, 2^{\gamma_1} - 1]\!]$.

[MGTF19] is very complex and needs expensive processing to handle all cases in a masked and constant-time manner: their solution requires an $\mathrm{A2B}_{q_\mathrm{D}}$ conversion, together with twelve arithmetic additions over Boolean shares (including three arithmetic negations). The authors furthermore note that independent computation of the high or low digit is difficult, due to the dependencies between the computation of the two: both must thus be computed almost entirely, even if only one of them is needed. Their masked software implementation of LowBits thus expends nearly eight thousand cycles to compute the lower digit of a single coefficient on a 32-bit ARM microcontroller.

It is however possible to remove much of the complexity of this function. First, as mentioned in § 2.4.1.3, the Decompose function can actually be expressed without special cases. Second, state-of-the art sensitivity analyses conclude that the high-order bits of the decomposition do not need to be protected, which simplifies the last steps of the computation by removing the need for a $\mathrm{B2A}_{q_\mathrm{D}}$ conversion. We reproduce in algorithm 3.7 the algorithmic description of the masked decomposition of Azouaoui *et al.* [ABC+23]. Two different techniques are used to compute the high bits, depending on the value of γ_2. For $\gamma_2 = (q_\mathrm{D}-1)/32$, it uses the fact that $\mathrm{HighBits}_{\gamma_2}(w) = (-\delta w + (q_\mathrm{D}-1)/2) \bmod q_\mathrm{K} \bmod \delta$, where $\delta = -(\gamma_2)^{-1} \bmod q_\mathrm{K}$. On the other hand, when $\gamma_2 = (q_\mathrm{D}-1)/88$, the authors rely on the masked compression using modulus switching (SecCompress), introduced by Coron *et al.* [CGMZ23] for the computation of the high bits. After unmasking these, the computation of the low bits is easily accomplished by subtracting the weighted high-order digit from the input value.

Algorithm 3.7: Secure decomposition with unmasked high digit, from [ABC+23]

Parameter: Half-size of low digit, $\gamma_2 \in \{(q_\mathrm{D}-1)/32, (q_\mathrm{D}-1)/88\}$
Parameter: Masking order $d \geqslant 1$
Input: Integer coefficient as arithmetic shares $w^\star$ modulo q_D
Output: Unmasked high-order digit w_1, masked low-order digit $w_0^\star$ as arithmetic shares modulo q_D

if $\gamma_2 = (q_\mathrm{D}-1)/32$ **then** *// Security levels 3 and 5*
  $b^\star = w^\star$
  $b^{\mathbf{0}} = (b^{\mathbf{0}} + \gamma_2) \bmod q_\mathrm{D}$
  $b'^\star = ((2\gamma_2)^{-1} b^\star) \bmod q_\mathrm{D}$ *// $(2\gamma_2)^{-1} \equiv -16 \pmod{q_\mathrm{D}}$*
  $b'^{\mathbf{0}} = (b'^{\mathbf{0}} - 1) \bmod q_\mathrm{D}$
  $b'^\star = \mathrm{A2B}_{q_\mathrm{D}}(b'^\star)$
  $w_1'^\star = b'^\star \circ (2^4 - 1)$ *// Only keep the four LSBs of each share*
else *// $\gamma_2 = (q_\mathrm{D}-1)/88$, security level 2*
  $w_1'^\star = \mathrm{SecCompress}_{q_\mathrm{D}, -(2\gamma_2)^{-1}}(w^\star)$ *// $-(2\gamma_2)^{-1} \bmod q_\mathrm{D} = 44$; SecCompress from [CGMZ23]*
end
$w_1 = \bigoplus_{i=0}^{d} w_1'^{\boldsymbol{i}}$ *// Unmask w_1 since it does not need protection*
$w_0{}^\star = w^\star$
$w_0{}^{\mathbf{0}} = (w_0{}^{\mathbf{0}} - 2\gamma_2 w_1) \bmod q_\mathrm{D}$
return $w_1, w_0^\star$

Coron *et al.* [CGTZ23] extend both approaches of Azouaoui *et al.* so that each supports both values of γ_2. For a software implementation, they show that the approach based on masked compression both has a better baseline performance (one thousand cycles for first-order masking, rather than three thousand) and scales better with the number of shares.

3.6.4.5 Masked norm checking

The zero-knowledge security of Dilithium crucially relies on checking the norm of the intermediate values $\mathbf{z}$ and $\mathbf{r}_0$, and rejecting signature attempts for which the norm is too large. While the result of the norm check is not sensitive, it is absolutely necessary to keep these values secret if they exceed their bound. The bound checking must thus be done in a masked manner, and only the result of the check can be unmasked. According to the designers of Dilithium, it is acceptable to reveal the result of this check separately for each coefficient [BDK+21].

Each bound check consists in checking whether all coefficients x of the above-mentioned polynomial vectors respect an equation of the form $|x| < B$, where $B = \gamma_1 - \beta$ for $\mathbf{z}$ and $B = \gamma_2 - \beta$ for $\mathbf{r}_0$. Migliore *et al.* [MGTF19] instead perform the converse check, i.e. $B \leqslant x \leqslant q_\mathrm{D} - B$. Given x as Boolean shares, they test the left inequality by computing[13] $x + (-B)$ using arithmetic addition over Boolean shares (SecAdd), and examining the sign bit $b_0^\star$ of the result. They repeat this process for the other bound, computing $x + (B - q_\mathrm{D} - 1)$ and extracting the sign bit $b_1^\star$ of the result. Then, unmasking $b_0^\star \oplus b_1^\star$ indicates whether $|x| < B$. Actually, the authors do not unmask this bit directly: they compute the AND reduction of this bit over all coefficients, and only then unmask the result, which indicates whether all coefficients satisfy the bound: they argue that while this provision is not strictly needed for security, it does not significantly decrease the performance of the norm check.

When the $\mathbf{z}$ and $\mathbf{r}_0$ values are masked as arithmetic shares modulo q_D, a different method must be used for norm checking. Following Azouaoui *et al.* [ABC+23], testing whether $-\lambda_0 \leqslant x \leqslant \lambda_1 \pmod{q_\mathrm{D}}$ can be done with the sequence of operations in algorithm 3.8: the input value is first shifted by λ_0 in the arithmetic domain, giving $x'^\star$. Due to the modular arithmetic sharing, the result is implicitly reduced modulo q_D. The bound check is thus now single-ended: the input values lower than $-\lambda_0$ have been reduced modulo q_D to a large positive value. It thus suffices to convert $x'^\star$ to a Boolean

Algorithm 3.8: Secure norm checking, from [ABC+23]

Parameter: Bounds $\lambda_0, \lambda_1 \in [\![0, (q_\mathrm{D} - 1)/2]\!]$
Parameter: Word size $w \in \mathbb{N}$ such that $q_\mathrm{D} \leqslant 2^{w-1}$
Parameter: Masking order $d \geqslant 1$
Input: Input coefficient $x^\star$ as arithmetic shares modulo q_D
Output: Unmasked bit $b \in \mathbb{b}$ such that $b = 1 \Leftrightarrow x \in [\![-\lambda_0, \lambda_1]\!] \pmod{q_\mathrm{D}}$

1 $x^{\mathbf{0}} = (x^{\mathbf{0}} + \lambda_0) \bmod q_\mathrm{D}$
2 $x'^\star = \mathrm{A2B}_{q_\mathrm{D}}(x^\star)$
3 $y^\star = \mathrm{SecAdd}(x'^\star, 2^w - 1 - (\lambda_0 + \lambda_1))$ *// $b = 1 \Leftrightarrow x' \leqslant \lambda_0 + \lambda_1$*
4 $b'^\star = y^\star \gg (w - 1)$ *// Sign bit of the comparison, set when the above condition is true*
5 $b = \bigoplus_{i=0}^{d} b'^{\boldsymbol{i}}$ *// Unmask it*
6 **return** b

[13] Algorithm 10 of [MGTF19] seems to have sign and off-by-one errors in the bounds; we try to correct these typos in our description.

sharing $y^\star$, and compare it with $\lambda_0 + \lambda_1$ in a masked manner. This last operation is performed by subtracting $\lambda_0 + \lambda_1 + 1$ from $y^\star$, and checking the sign of the difference. Contrary to the solution of Migliore *et al.*, this sign bit can be safely unmasked, since it simultaneously describes the upper and lower bounds.

3.6.5 Performance impact of masking

While very effective and provably sound, masking is an expensive countermeasure in terms of memory usage, computation time, and in the case of hardware implementations, logic area. Masking linear functions over $d+1$ shares requires $d+1$ independent applications of the function in a share-wise manner, which increases the latency and memory usage of the algorithm linearly with the number of shares. The required area, instead, is not affected if the shares are processed sequentially by a single computational unit[14].

In hardware implementations, the cost of masking LWE-based key-encapsulation mechanisms at the first order amounts to an increase by about four times of the area × latency figure [AMD+21]. Both for Kyber and for Dilithium, a significant part of the cost of masking is due to the choice of a prime, rather than power-of-two modulus. Migliore *et al.* [MGTF19], for instance, show that most masked gadgets for Dilithium could be sped up by a factor of 7–8 if the scheme used a power-of-two modulus of similar size to q_D. Fritzmann *et al.* [FBR+22] similarly show how Saber is more efficiently masked than Kyber, due to two beneficial characteristics: its use of a power-of-two modulus, and its reliance on LWR rather than LWE, that allows it to use less random sampling. Having implemented both schemes on a RISC-V CPU with custom instruction-set extensions, they determine that protecting Saber with first-order masking multiplies its latency by 2.3–2.7, while a factor of 4–5 is reached for Kyber.

3.6.6 Protocol-level masking

While masking is generally applied at the level of individual arithmetic or bitwise operations, it is also possible to perform masking at the protocol level, by making use of the additively homomorphic nature of LWE. This technique is applied by Banerjee *et al.* [BUC19] to the Ring-LWE scheme NewHope in the following way: on receiving a ciphertext $(\mathbf{u}, v)$, the decapsulator first samples a random one-time message μ' and encrypts it as usual, giving another ciphertext $(\mathbf{u}', v')$; then, the sum of the two ciphertexts $(\mathbf{u}+\mathbf{u}', v+v')$ is decapsulated giving message μ''. The original message is recovered as $\mu = \mu'' - \mu'$.

This technique has a major pitfall: it raises the probability of a decryption failure by increasing the ciphertext noise[15]. Additionally, it can only be used for the decryption

[14] As noted by Cassiers and Standaert [CS21], corresponding shares should not be processed consecutively or transition-based leakage will occur. However, when dealing with whole polynomials, this pitfall can be avoided easily if all coefficients are independently shared, by first processing the first share of all coefficients, then all second shares, and so on.

[15] We can estimate the resulting failure probability using the scripts provided by the CRYSTALS design team [DS21]. If we consider that first-order protocol-level masking doubles the variance of the error vectors sampled for encryption, and also doubles the rounding error introduced by ciphertext compression, we get failure probabilities of $2^{-42.3}$, $2^{-48.3}$ and $2^{-75.2}$, respectively, for security levels 1, 3 and 5.

step of decapsulation, so the re-encryption step must be protected in the usual way, by masking each of its low-level operations.

On the other hand, this technique might have an accessory advantage: it should complicate the attacks based on Decryption-Failure and Plaintext-Checking oracles (section 3.3), due to the above-mentioned additional ciphertext noise. While it is unlikely to prevent the attack altogether, it may force the attacker to acquire several times more traces before he can discount the probabilistic effect of protocol-level masking on the failure of individual ciphertexts.

3.7 Blinding or hiding

As a reduced variant of masking, blinding (or hiding) multiplies all coefficients of each polynomial with a single random integer, which is equivalent to a (multiplicative) arithmetic masking scheme where all coefficients of the mask are equal. This technique was first proposed by Saarinen [Saa18] in the context of the BLISS lattice signature. Since this scheme makes use of the NTT, the author further proposes to use randomly-chosen roots of unity for blinding, instead of randomly-chosen modular integers. This allows to make the countermeasure essentially free, since application of the NTT already involves multiplying the coefficients by roots of unity, so the two can be merged.

While the cost of this countermeasure is much lower than that of actual masking, its effectiveness is also reduced since the same blinding value is used for all coefficients of a polynomial, and it may be entirely ineffective depending on the attack model [PP19b].

Consequently, Ravi *et al.* [RPBC20] propose an intermediate countermeasure between masking and Saarinen's multiplicative blinding, specifically for the computation of the forward and inverse NTT. Instead of blinding all coefficients of a polynomial with the same twiddle factor, they propose a finer-grained approach, in which the computation of each NTT layer is blinded by an independent random twiddle factor. An even finer countermeasure can be applied, by choosing different blinding values for each butterfly operation. It is also possible to vary the level of protection between these extremes by using an intermediate number of different twiddle factors for each NTT layer. The authors furthermore provide modified versions of the NTT butterfly operations, that are able to handle different blinding factors for their two input values and their two output values.

An additional and orthogonal blinding method proposed by Saarinen [Saa18] consists in applying a random negacyclic shift to polynomial coefficients, which has the effect of multiplying them by X^r where $0 \leqslant r < n$ is a random integer. The effect of this countermeasure is similar to a weak version of the shuffling countermeasure, which is described below.

3.8 The shuffling countermeasure

Another class of countermeasures against side-channel attacks is shuffling, that randomizes the order of secret data, either in time or in space (location in memory). An example

implementation using such measures is given by Jati *et al.* [JGCS24], that include three such techniques. First, random delays in the form of dummy clock cycles are included throughout the computation. This randomization is mainly effective when the signal-to-noise ratio of power traces is already low, since automated trace-alignment techniques (that try to undo this randomization) perform badly in this context. Second, address randomization is used to randomize the order of data processing when the same operation must be applied to several values in an arbitrary order. Third, instruction randomization reorders at random sequences of independent instructions, to cause strong misalignment between the power traces of different executions of the algorithm. These three countermeasures, together with countermeasures against fault injection, are reported to take up less than 5 % of the total area. Their practical effectiveness is, however, not analyzed.

Yet, several attacks present shuffling as a useful countermeasure. In the case of [SPH22], shuffling the order of computations on the coefficients of the polynomials prevents from attacking all coefficients at once, consequently multiplying the required number of side-channel traces by 256 before the attack can succeed. The combined chosen-ciphertext and fault attack of [HPP21] is also only partially mitigated by shuffling, which may force the attacker to run 256 times more faulty decryptions to test all shuffling positions.

In some cases, shuffling can defeat entirely those attacks that need to associate each point in time with the precise datum being processed. This is the case, notably, for the attacks of Pessl and his coauthors [PP21, PP19b, PPM17].

We now present various techniques for shuffling the order of k-bit addresses in the range $[\![0, 2^k - 1]\!]$, and describe their properties. We note that, in the case of Kyber and Dilithium, polynomials have 256 coefficients, so the shuffling will typically involve 8-bit addresses. Our own solution for shuffling will be discussed in section 6.3.

LFSR A standard technique to cheaply generate randomized sequences is using a linear-feedback shift register (LFSR), and this method is applied by Zijlstra *et al.* [ZBT19] to shuffle the order of computation of the NTT and other polynomial operations. Given a degree-k binary polynomial $f \in \mathbb{F}_2[X]$ and an initial state $a \in \mathbb{F}_2[X]/f(X)$, the LFSR can be used to compute the sequence $(X^i a \bmod f)_{i \geqslant 0}$. Provided f is irreducible and $a \neq 0$, the $2^k - 1$ first terms of this sequence are exactly all nonzero elements of $\mathbb{F}_2[X]/f(X)$. Consequently, each random nonzero a generates distinct permutations of $[\![1, 2^k - 1]\!]$. Since modular multiplication by X modulo f can be performed with only a logical shift and a few XORs (one fewer than the number of nonzero coefficients in f), this technique uses a very low hardware area.

However, as noted by the authors, the LFSR method has some drawbacks, the first one being the limited amount of randomness introduced: there are only as many shuffling orders as initial values a, namely $2^k - 1$; furthermore, they only differ in their start index. Consequently, this method introduces slightly less than k bits worth of shuffling. Furthermore, the restriction on a being nonzero forces to either use rejection sampling, or accept a non-uniform distribution of the initial values. Very importantly also, the LFSR never generates an all-zero address, so this address must be handled otherwise. Zijlstra *et al.* propose not to shuffle the first operation, so that it always processes address zero [ZBT19].

Permutation network Given the drawbacks of the previous method, Zijlstra *et al.* [ZBT19] propose an other solution, based on permutation networks. With this method, the list of integers from 0 to $2^k - 1$ is shuffled in pairs using k layers each having 2^{k-1} boxes that can swap two fixed elements based on a control bit. In total, this network is able to generate $2^{2^k k/2}$ distinct permutations, which is both a power of two — avoiding the need for any rejection sampling — and close to the total number of permutations of $[\![0, 2^k - 1]\!]$, namely $2^k! \approx 2^{2^k(k-\log_2 e)}$.

We note, however, that such shuffling has a very high cost, particularly in terms of area due to the large number of swapping blocks — 1024 for degree-256 polynomials. This effect on area is clearly visible in the implementation by Zijlstra *et al.* [ZBT19] of a stripped-down version of NewHope on an Artix-7 FPGA, where the sole addition of the shuffling countermeasure to a complete implementation of the decryption raises its area by more than six times (in LUT count).

Fisher-Yates shuffle A widely used method to shuffle a list of arbitrary length, using a permutation chosen uniformly from the set of all permutations, is Fisher-Yates shuffle [FY48], of which we described a specialization when introducing Dilithium challenge sampling (SampleInBall, algorithm 2.15). This is the address-randomization method chosen by Jati *et al.* [JGCS24] notably. The principle of this shuffle is simple: given a list of ℓ elements, sample an integer i uniformly from $[\![0, \ell - 1]\!]$, and swap the i^{th} element of the list with the last one. Proceed in the same way for the $\ell - 1$ first elements of the list, choosing one of them and swapping it with the last one. Continue analogously until the whole list has been shuffled.

It can be easily seen that this method does generate all $\ell!$ possible permutations, and uniformly so, thus providing maximum entropy. However, two difficulties may arise in using it: first, a dedicated RAM space must be reserved to compute and store the permutation, which may constitute a large area for low-cost device when randomizing 256 addresses of 8 bits each. Second, this shuffle relies on successively sampling integers in $[\![0, i - 1]\!]$ for each i from ℓ to 2, thus requiring generic rejection sampling, which may use large amounts of randomness.

3.9 Fault detection

Several implements are given by Jati *et al.* [JGCS24] to detect whether the device was subjected to fault injection, and take appropriate measures. They propose error-detection codes to check the data stored in registers or in memory, as well as duplication of critical finite-state machines and control signals in inverted logic. Additionally, cycle counters ensure that each operation lasted the expected number of cycles, to make sure no instruction-skip fault occurred.

A possible target for fault attacks is error sampling, since incorrect error sampling during key generation or message encapsulation can lead to decreasing the hardness of recovering the private key or the one-time message (respectively). The schemes specify random sampling as a two-level process: given a random seed of a fixed length, this seed

is first deterministically expanded with an extendable output function, after which a second deterministic process reshapes this randomness to the appropriate distribution. Howes *et al.* [HPA21] propose statistical tests to be performed on the output of the error sampler, to verify that it satisfies some expected properties. They provide three levels of validation: the cheapest one simply checks that the sampler never outputs a sequence of identical values having unlikely length. On top of this check, the second level of validation computes the sample mean and sample variance over a number of error samples, and compares them with their respective confidence intervals. Finally, the most thorough validation they propose consists in building a histogram of the sampled errors, and matching it with the expected distribution in terms of goodness of fit.

Heinz and Pöppelmann [HP23] use a redundant number representation for combined side-channel and fault protection. This technique, previously used by Zijlstra *et al.* [ZBT19] for the sole purpose of side-channel protection, consists in computing all arithmetic operations of the scheme modulo $q'q$ instead of modulo q, where integer q' should be coprime with q and with all the constants involved in the scheme. Using the Chinese Remainder Theorem, the results of these computations modulo $q'q$ can be split into results modulo q and results modulo q', the former allowing to get the expected result for the scheme, and the latter giving the ability to detect fault injection. Indeed, reproducing the scheme's computations modulo q' allows to check the values obtained modulo $q'q$, a mismatch indicating an effective fault injection. This countermeasure protects against several fault models: instruction skipping, bit flipping, random fault (where the fault causes the result of a computation to be uniformly random) and zeroing fault. The side-channel protection provided by this method is however less clear: in particular, Heinz *et al.* [HP23] show that some chosen-ciphertext attacks may lead to a derandomization that disables the protection.

4 Belief-propagation attack on optimized Cortex-M4 implementation of Kyber

In this chapter, we describe our work on a belief-propagation attack against an optimized software implementation of Kyber. This work has been published in [AEVR23], whose contents are reproduced mostly unchanged here.

As already mentioned in section 3.2.1, several works [PPM17, PP19b, HHP+21, HSST23] have shown that lattice-based cryptography schemes are vulnerable to soft-analytical side-channel attacks (SASCA). In particular, the number-theoretic transform (NTT) used in Kyber is a prime target for this kind of attacks due to its regular arithmetic structure that is well adapted to belief-propagation attacks. Among these attacks, those that were performed on real traces targeted an ARM Cortex-M4 microcontroller, either running the reference implementation of Kyber, or running an older version of the Kyber implementation proposed by the `pqm4` project [KPR+22]. Besides being an important target for practical applications using PQC algorithms, Cortex-M4 microcontrollers have been chosen by NIST as the primary evaluation platform for embedded devices during the PQC standardization process. However, to the best of our knowledge, no such attacks target the most recent version of the Kyber implementation provided by `pqm4`, which is thoroughly optimized for Cortex-M4 microcontrollers. Hamburg *et al.* [HHP+21] hint that attacking this implementation requires some care to account for its specifics. In the following, we show that by closely modeling this implementation, we can perform very effective single-trace attacks.

Our contribution We apply the belief-propagation attack to an optimized software implementation of the NTT making use of the specific arithmetic instructions available in the ARMv7E-M instruction set. We show that considering a reasonable leakage model, the attack succeeds even in the presence of noise, despite the compactness of the implementation, that limits the amount of side-channel leakage. We also study the influence of measurement-noise variance on the success rate and the execution time of the attack, and show that a precise knowledge of the amount of noise is not required to get satisfactory results.

By specializing our attack to situations when the NTT is run over small-valued coefficients, we are able to recover all secret coefficients with high probability up to much stronger noise levels, while keeping the computational effort well within the capabilities of any individual attacker.

Outline In section 4.1, we give some additional notations used in this chapter only, and introduce the relevant background. In section 4.2, we detail the implementation of

our attack, and evaluate its results on simulated side-channel traces in section 4.3. We explore in section 4.4 how the attack can be exploited in practice, and how to protect against it. Finally, in section 4.5 we conclude on our contribution and describe various possible follow-ups to our work.

4.1 Preliminaries

After introducing a change of notation and recalling the use and structure of the NTT, we briefly describe Kyber key-encapsulation mechanism, then summarize how Huang *et al.* [HZZ+22] optimize the implementation of its NTT for Cortex-M4 microcontrollers, and wrap up with a description of the Belief Propagation algorithm in the context of Soft-Analytical Side-Channel Attacks.

4.1.1 Notations

This chapter will often involve pairs of 16-bit coefficients being packed into a 32-bit integer. For two 16-bit coefficients h and ℓ, we will denote the corresponding packed value by $2^{16}h \mid \ell$, where $\mid$ stands for the bitwise OR operation. Conversely, the top and bottom 16-bit parts of a 32-bit value are respectively selected by subscripts $_\mathtt{t}$ and $_\mathtt{b}$, *i.e.* $(2^{16}h \mid \ell)_\mathtt{t} = h$ and $(2^{16}h \mid \ell)_\mathtt{b} = \ell$.

We recall that Kyber uses matrices and vectors of polynomials from a fixed ring $R_{q_\mathrm{K}} = \mathbb{F}_{q_\mathrm{K}}[X]/(X^n + 1)$, where the degree of the reducing polynomial is $n = 256$ and the integer modulus is prime $q_\mathrm{K} = 3329$. We denote by B_η the distribution over R_{q_K}, such that each coefficient independently follows a binomial law of parameter η, having support $[\![-\eta, \eta]\!]$.

4.1.2 Number-Theoretic Transform

Cryptography algorithms based on module or ideal lattices make a large use of arithmetic operations on integer polynomials, in particular polynomial multiplication. To optimize the efficiency of these operations, several schemes [SAB+20, ADPS16, LS19] have parameters that allow (or mandate) the polynomial multiplications to be performed using the NTT, which is asymptotically the most efficient algorithm for that task.

The NTT is the equivalent of the Discrete Fourier Transform (DFT) for prime-order finite fields. It provides a bijective mapping from a polynomial ring R to the corresponding so-called *NTT domain*. Polynomial multiplication (or convolution) in the former domain translates to point-wise multiplication in the latter. More concretely, given two polynomials f and g from R, their product can be computed as $fg = \mathrm{NTT}^{-1}(\mathrm{NTT}(f) \circ \mathrm{NTT}(g))$ where $\circ$ represents point-wise multiplication and NTT, NTT^{-1} are the NTT in the forward and inverse direction respectively.

The implementation of an n-point NTT (n being a power of two) can be done efficiently with a regular structure of so-called *butterfly* operations, arranged in several *layers*. These butterflies take as input a pair of coefficients, perform simple modular arithmetic on the pair together with one root of unity (in the sense of modular exponentiation), and

output another pair of coefficients that will be processed by the next layer. The roots of unity are often called *twiddle factors*. In the case of Kyber, the NTT is processed in 7 layers, each applying 128 butterfly operations.

The NTT is often implemented with Cooley-Tukey (CT) butterflies in the forward direction, and Gentleman-Sande (GS) butterflies in the reverse direction [HZZ+22]. These operations, applied to two integer values a and b and parameterized by twiddle factor ζ, are respectively described by

$$\mathrm{CT}_\zeta(a,b) = \Big((a+\zeta b) \bmod q_\mathrm{K} \quad (a-\zeta b) \bmod q_\mathrm{K}\Big) \text{ and} \tag{4.1}$$

$$\mathrm{GS}_\zeta(a,b) = \Big((a+b) \bmod q_\mathrm{K} \quad ((b-a)\zeta) \bmod q_\mathrm{K}\Big). \tag{4.2}$$

We note that the GS butterfly sometimes includes modular divisions by 2, but they are often left out from the butterfly and carried out during a pre- or post-processing.

4.1.3 Simplified specification of Kyber

We now describe a simplified layout of Kyber's Ind-CPA key generation, encryption and decryption operations. We omit several aspects from this description, in particular the compression and decompression of the ciphertext. These aspects have been described in section 2.3.1 but they do not matter for the understanding of this chapter.

We recall in algorithm 4.1 the procedure for key generation. A private key for Kyber is given by a polynomial vector $\mathbf{s} \in R_{q_\mathrm{K}}^k$ ($k \in \{2,3,4\}$ depending on the security level) whose coefficients are sampled from a binomial distribution, while the corresponding public key is a tuple containing a matrix $\mathbf{A}$ of polynomials sampled uniformly at random, and the product of $\mathbf{A}$ by the private vector $\mathbf{s}$ with the addition of secret noise $\mathbf{e}$.

Algorithm 4.1: Kyber key generation (simplified)

Output: Private key $sk = \hat{\mathbf{s}} \in R_{q_\mathrm{K}}^k$
Output: Public key $pk = (\hat{\mathbf{A}}, \hat{\mathbf{t}}) \in R_{q_\mathrm{K}}^{k\times k} \times R_{q_\mathrm{K}}^k$
1 $\hat{\mathbf{A}} \xleftarrow{\$} R_{q_\mathrm{K}}^{k\times k}$
2 $\mathbf{s} \xleftarrow{\$} B_{\eta_1}^k$
3 $\mathbf{e} \xleftarrow{\$} B_{\eta_1}^k$
4 $\hat{\mathbf{s}} = \mathrm{NTT}(\mathbf{s})$
5 $\hat{\mathbf{e}} = \mathrm{NTT}(\mathbf{e})$
6 $\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$
7 **return** $sk = \hat{\mathbf{s}}$, $pk = (\hat{\mathbf{A}}, \hat{\mathbf{t}})$

Algorithm 4.2 shows how the encryption of a 256-bit message m is performed: a one-time secret vector $\mathbf{r}$ is sampled from a binomial distribution, and multiplied on one hand by public matrix $\mathbf{A}$, and on the other hand by public vector $\mathbf{t}$. Before being released, these two results are added with some independently-sampled noise ($\mathbf{e_1}$ and e_2 respectively), and the second one is further added with a polynomial representation of the message (each coefficient being either 0 or $\lfloor q_\mathrm{K}/2 \rceil$ depending on the corresponding message bit).

Algorithm 4.2: Kyber encryption (simplified)

Input: Public key $pk = (\hat{\mathbf{A}}, \hat{\mathbf{t}}) \in R_{q_\mathrm{K}}^{k \times k} \times R_{q_\mathrm{K}}^{k}$
Input: Message $m \in \mathbb{b}^n$
Output: Ciphertext $c = (\mathbf{u}, v) \in R_{q_\mathrm{K}}^{k} \times R_{q_\mathrm{K}}$

$\mathbf{r} \xleftarrow{\$} B_{\eta_1}^{k}$
$\mathbf{e_1} \xleftarrow{\$} B_{\eta_1}^{k}$
$e_2 \xleftarrow{\$} B_{\eta_2}$
$\hat{\mathbf{r}} = \mathrm{NTT}(\mathbf{r})$
$\mathbf{u} = \mathrm{NTT}^{-1}(\hat{\mathbf{A}}^\mathsf{T} \circ \hat{\mathbf{r}}) + \mathbf{e_1}$
$v = \mathrm{NTT}^{-1}(\hat{\mathbf{t}}^\mathsf{T} \circ \hat{\mathbf{r}}) + e_2 + \lfloor q_\mathrm{K}/2 \rceil m$
return $c = (\mathbf{u}, v)$

Finally, decryption (shown in algorithm 4.3) involves multiplying the first part $\mathbf{u}$ of the ciphertext with the private vector, subtracting it from the second part v of the ciphertext, and decoding one message bit per coefficient of the resulting polynomial, by determining for each bit whether it is closer to 0 or to $\lfloor q_\mathrm{K}/2 \rceil$.

Algorithm 4.3: Kyber decryption (simplified)

Input: Private key $sk = \hat{\mathbf{s}} \in R_{q_\mathrm{K}}^{k}$
Input: Ciphertext $c = (\mathbf{u}, v) \in R_{q_\mathrm{K}}^{k} \times R_{q_\mathrm{K}}$
Output: Message $m' \in \mathbb{b}^n$

$M' = v - \mathrm{NTT}^{-1}(\hat{\mathbf{s}}^\mathsf{T} \circ \mathrm{NTT}(\mathbf{u}))$
$m' = \lfloor (2/q_\mathrm{K}) M' \rceil \bmod 2$
return m'

4.1.4 Optimized implementation of Kyber NTT for Cortex-M4

Being based on the ARMv7E-M 32-bit architecture, Cortex-M4 microcontrollers have a number of arithmetic instructions that can efficiently operate on half-word (16-bit) and byte (8-bit) quantities. Thanks to these, Huang *et al.* [HZZ+22] propose a fast implementation of the NTT, able to perform two simultaneous butterfly operations (either CT or GS) in seven single-cycle instructions. One novelty of their implementation is the use of an improved version of Plantard modular reduction [Pla21] with wider input range and narrower output range, thus being beneficial to lazy-reduction techniques.

We recall their solution for a CT butterfly in algorithm 4.4. This solution is specifically tailored for Kyber's modulus $q_\mathrm{K} = 3329$ using an additional constant $\alpha = 3$ such that $q_\mathrm{K} 2^{\alpha+1} < 2^{16}$. To be able to use this technique, the twiddle factors ζ used for the NTT need to be expressed over 32 bits and multiplied by some constant, but we do not develop this step here and refer the reader to [HZZ+22] for details.

Due to its benefits, that optimized implementation is currently used for Kyber's forward and inverse NTT in the `pqm4` project [KPR+22], which proposes implementations of candidates to NIST PQC standardization, optimized for Cortex-M4 microcontrollers.

Algorithm 4.4: Double Cooley-Tukey butterfly for Cortex-M4

Input: Packed pairs of signed 16-bit coefficients $a = 2^{16}a_{\mathsf{t}} \mid a_{\mathsf{b}}$, $b = 2^{16}b_{\mathsf{t}} \mid b_{\mathsf{b}}$
Input: 32-bit corrected twiddle factor ζ (from real twiddle factor ζ_0)
Input: q_{K} and $q_{\mathrm{K}}2^{\alpha}$ in two separate registers
Output: $a' = 2^{16}a'_{\mathsf{t}} \mid a'_{\mathsf{b}}$, $b' = 2^{16}b'_{\mathsf{t}} \mid b'_{\mathsf{b}}$
such that $a'_{\mathsf{t}} \equiv a_{\mathsf{t}} + b_{\mathsf{t}}\zeta_0$, $a'_{\mathsf{b}} \equiv (a_{\mathsf{b}} + b_{\mathsf{b}}\zeta_0)$, $b'_{\mathsf{t}} \equiv a_{\mathsf{t}} - b_{\mathsf{t}}\zeta_0$, $b'_{\mathsf{b}} \equiv a_{\mathsf{b}} - b_{\mathsf{b}}\zeta_0 \pmod{q_{\mathrm{K}}}$

1 **smulwb** t, ζ, b . . . . . // $t = \zeta b_{\mathsf{b}} \gg 16$
2 **smulwt** b, ζ, b . . . . . // $b = \zeta b_{\mathsf{t}} \gg 16$
3 **smlabb** $t, t, q_{\mathrm{K}}, q_{\mathrm{K}}2^{\alpha}$ . . . . . // $t = t_{\mathsf{b}}q_{\mathrm{K}} + q_{\mathrm{K}}2^{\alpha}$
4 **smlabb** $b, b, q_{\mathrm{K}}, q_{\mathrm{K}}2^{\alpha}$ . . . . . // $b = b_{\mathsf{b}}q_{\mathrm{K}} + q_{\mathrm{K}}2^{\alpha}$
5 **pkhtb** $t, b, t, \mathrm{asr}\#16$ . . . . . // $t = 2^{16}t_{\mathsf{b}} \mid b_{\mathsf{b}}$
6 **usub16** b', a, t . . . . . // $b' = 2^{16}(a_{\mathsf{t}} - t_{\mathsf{t}}) \mid (a_{\mathsf{b}} - t_{\mathsf{b}})$
7 **uadd16** a', a, t . . . . . // $a' = 2^{16}(a_{\mathsf{t}} + t_{\mathsf{t}}) \mid (a_{\mathsf{b}} + t_{\mathsf{b}})$
8 **return** a, b

4.1.5 Belief Propagation

We base our work upon the attack of Primas, Pessl and Mangard [PPM17] and its improvements by Pessl and Primas [PP19b], that use belief propagation in the context of soft-analytical side-channel attacks (SASCA). This approach starts with a side-channel template attack, that recovers probability distributions for some intermediate values manipulated by the target, depending of its leakage. Then, the operations performed by the implementation under attack are modeled, and this model is used to solve for the secret values that satisfy the leaked information.

Belief Propagation is an optimization algorithm that is well adapted to this task. We here give a high-level overview of this technique, and refer the reader to MacKay [Mac03] for details and definitions. The aim of belief propagation in our setup is to marginalize a joint probability distribution, that is, to derive from it independent probability distributions for each involved random variable. Practically, our joint probability distribution is the sum of the information we have on the algorithm being attacked, together with the information we get through side-channel leakage. Belief propagation then helps us determine probable values for each variable, which represents a secret or intermediate value used by the algorithm. To do so, the algorithm is modeled as an undirected graph having *variable nodes*, that represent the secret and intermediate values, and *factor nodes*, that represent the information we have on variables, in the form of joint probability distributions over subsets of them. Edges in this *factor graph* link factor nodes to the variables they depend on. The core of the belief-propagation algorithm then consists in passing *messages* between nodes, that is, local approximations of the marginal probability distributions of variables.

Let us illustrate how message passing works with a simple example. We depict in fig. 4.1 a simple factor graph having three factor nodes $F_1, \ldots, F_3$ and four variable nodes $V_{\mathrm{a}}, \ldots, V_{\mathrm{d}}$. Each factor has a *potential*, that is, a function that assigns a probability to each possible tuple of outcomes for the variables it is linked to. Two types of messages are exchanged between nodes: variable-to-factor messages, where the message from variable

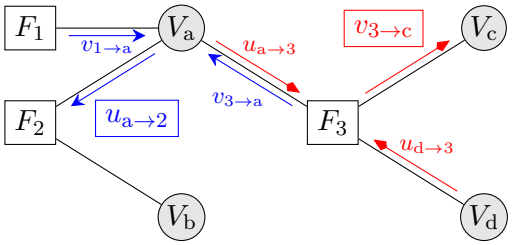


Figure 4.1: Example factor graph with two update rules depicted: from variable a to factor 2 (blue arrows) and from factor 3 to variable c (red arrows).

n to factor m will be denoted by $u_{n\to m}$, and factor-to-variable messages, similarly denoted by $v_{m\to n}$ for the same pair of nodes. Both $u_{n\to m}$ and $v_{m\to n}$ represent an approximation of the probability distribution of variable n.

The rule for updating variable-to-factor messages is straightforward: the message from variable n to factor m is the point-wise product of all messages sent to variable n by factors other than m. For instance, the blue-framed message in fig. 4.1 is updated for each outcome x of variable a through

$$u_{\mathrm{a}\to 2}(x) = v_{1\to \mathrm{a}}(x) v_{3\to \mathrm{a}}(x)\,. \tag{4.3}$$

The update of factor-to-variable messages is more computationally expensive. The message from factor m to variable n is updated by computing the probability distribution of variable n from *(i)* the prior probability distributions of the other variables linked to factor m (as given by the message each other variable sends to factor m) and *(ii)* the joint probability distribution of all variables involved in factor m (as given by the potential of the factor). Practically, the red-framed message in fig. 4.1 is updated according to

$$v_{3\to \mathrm{c}}(x) = \sum_{x_a, x_d} F_3(x_a, x, x_d) u_{\mathrm{a}\to 3}(x_a) u_{\mathrm{d}\to 3}(x_d) \tag{4.4}$$

for each outcome x of variable c, where (x_a, x_d) in the sum runs over the Cartesian product of the possible outcomes for variables a and d.

These message updates are repeatedly applied to each edge of the factor graph until convergence or another termination condition is reached[1]. Then, the marginal probability distribution of each variable is determined by point-wise multiplying all its incoming messages, and normalizing.

[1]For acyclic factor graphs, convergence to the optimal solution is guaranteed independently from the order in which messages are updated; when, instead, the graph contains cycles, the message-passing order matters and neither optimality nor convergence are guaranteed.

4.2 Attack implementation

4.2.1 Factor graph

We model Kyber's optimized NTT implementation from the `pqm4` project as a factor graph, considering the actual operations performed by the microcontroller. In particular, the model includes that most input, intermediate and output values are processed in pairs.

We assume that each arithmetic operation leaks information on the result that it writes to a register, and that each load or store operation between RAM and registers leaks information on the corresponding data word. In this latter case, we assume a single leakage point for each pair of coefficients in each layer of the NTT.

We show in fig. 4.2 our factor graph for a 1-layer forward NTT over 4 coefficients. We represent variable nodes with circles, and factor nodes with rectangles.

We use three types of factors denoted by L, V and I to encode the information acquired through side-channel leakage. These factors assign to each outcome of the corresponding variable (for type I) or pair of variables (for types L and V) the probability of the variable taking this outcome, conditioned by the actual side-channel measurements associated with the factor. To this end, the measurements are supposed to be in the Hamming-weight metric, with Gaussian noise having a fixed standard deviation σ_F. Bayes' theorem is used to recover the desired probability from the knowledge of the measurement outcome.

- Factor type L represents one load or store operation of a pair of coefficients, and its potential is based on the measured leakage of the corresponding load or store, before or after the processing of the corresponding double-butterfly operation. This leakage corresponds to the measurement of a or b at the input or output of algorithm 4.4.
- Factor type V represents the operation of packing a pair of coefficients, after Plantard multiplication, into a single 32-bit register. Such factors are only included for variable pairs that are fed to the second input of a butterfly operation, and they are configured based on the leakage of the value computed at line 5 of algorithm 4.4.

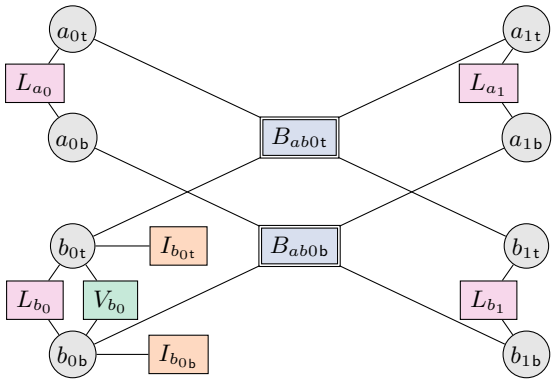


Figure 4.2: Excerpt from the factor graph showing one double-butterfly operation with inputs a_0 and b_0, and outputs a_1 and b_1, each being a pair of 16-bit coefficients.

- Factor type I represents the leakage of the arithmetic operations that depend on a single variable; its potential is based on the measurement of the results of the two arithmetic operations that involve this variable alone; one is provided for each variable that is fed to the second input of a butterfly operation. The factor linked to even-index (resp. odd-index) coefficients is configured based on the leakage of the values computed at lines 1 and 3 (resp. 2 and 4) of algorithm 4.4.

In addition to these, a fourth factor type models the butterfly operations. Following Pessl *et al.* [PP19b], we use a single B-type factor node to model each butterfly, rather than one node for each output of each butterfly. Factors of this type assign equal probability to all tuples of two input values and two output values that respect the butterfly equation (taking into account Plantard multiplication and lazy reduction), and assign zero probability to all other tuples.

Applying this model to a 7-layer NTT over 256 coefficients gives a factor graph with 2048 variable nodes, 896 unary factors (type I), 1920 binary factors (type L or V), and 896 factors having a fanout of four (type B).

4.2.2 Message-updating order

We make use of a message-updating order similar to that of Pessl *et al.* [PP19b]: we propagate messages from the first layer to the last one, then back to the first one. More specifically, we propagate messages one layer after another, where the order in each layer is the following:

1. from L, V and I factor nodes at the input of the layer, to the input variable nodes;
2. then, from the input variables nodes to the B nodes of the layer;
3. then, from the B nodes to the output variable nodes;
4. then, from the output variable nodes to the L, V and I nodes linked to the output variables[2].

Once the messages have been passed across all layers in the forward direction, the above steps are done in reverse across all layers from the last to the first. A single back-and-forth propagation is referred to as one iteration.

In each of the steps enumerated above, all messages are independent from one another, and can be computed in any order, or even in parallel. This property allows the computation to be highly parallelized, up to 256 times, with a nearly linear performance gain.

4.2.3 Message damping

Similar to [PP19b], we use message damping in order to get better and faster convergence. Since our factor graph contains loops, belief propagation can be unstable and make the

[2]The V and I factor nodes linked to output variables are not visible in fig. 4.2 since only a single layer of the NTT is represented.

messages oscillate. Such oscillations are detrimental to both convergence speed and accuracy, and should thus be dampened. To do so, every time a message is updated, a weighted average between its old value and the value given by the message-passing rules is used as its new value. We call *damping value* the weight δ given to the message-passing rule, while the old value of the message is given weight $1 - \delta$. We empirically found a damping value of $\delta = 95\,\%$ to give satisfactory results, so we use it in all our experiments.

4.2.4 Message pruning

To improve the run-time of the algorithm, we adopt a message-pruning strategy, that only processes nonzero outcomes when computing message updates. To make full use of this technique, we also truncate low-probability outcomes to zero. Care must be taken to select a low enough threshold for this truncation to minimize the probability of suppressing the actual value of a variable from a belief. We chose a threshold of 10^{-8} times the highest probability in each message, as we empirically found this value to offer a good compromise between the success rate and the average runtime for various parameters.

We note that the efficiency of message pruning somewhat decreases when also using message damping, since the latter slows down the rate at which probabilities can decrease. However, this effect is only significant for experiments that converge in very few iterations.

4.2.5 Termination

We iterate the algorithm, repeatedly updating messages until one termination condition is met: either the update of messages changed all their values by less than a chosen threshold (set to 10^{-5} to guarantee having reached a stable state); or, one message is all-zeroes, which means the algorithm has reached an inconsistent state with no solutions; or, the number of iterations has reached a given limit (set to 100 after observing that most of the experiments terminated in less than 20 iterations). Once one of these termination conditions has been reached, the marginal probabilities of each variable are computed by multiplying all the messages coming from its adjacent factors, and normalizing.

4.2.6 Progress monitoring

During execution of the belief propagation, we keep track of the residual Shannon entropy of all variables. This measurement indicates the degrees of freedom that remain in the state after each iteration.

4.2.7 Implementation and computing resources

All parts of our simulation, including the model of the Cortex-M4 implementation and the belief-propagation algorithm, are implemented in Python (version 3.10.6). The algorithm implementing the message-update rules is carefully written to be efficient and cache-friendly, and furthermore compiled using the `numba` library [LPS15] (version 0.55.1). The algorithm is run on an AMD EPYC 7713P processor running at 2 GHz.

The CPU has 64 physical cores, but at most 32 of them were used for our algorithm (the CPU being shared with other unrelated tasks). In the following, we name *CPU time* the sum of the runtimes over all active CPU cores. Across all experiments, the memory usage peaked at 10 GB. We do not claim our attack algorithm to be particularly efficient in runtime or memory usage, and only intend to demonstrate its practicality and how its runtime evolves with attack parameters.

4.3 Results

In this section, we show that our attack allows us to recover a uniformly random polynomial at the input of the NTT, even when the side-channel leakage has moderate noise, having a standard deviation of 1 (for 99 % success rate) or even 1.2 (for 90 % success rate). We recall that, since we are measuring the Hamming weight of 32-bit registers, noiseless measurements are in the range from 0 to 32, inclusively. We will discuss in section 4.4.1 on what situations correspond to such amounts of noise in a practical attack.

We also show that the standard deviation of the noise does not need to be precisely known, and that slightly over- or under-estimating it during the attack does not significantly impact the quality or runtime of the secret recovery.

Finally, we specialize our setup to the case when the polynomial input to the NTT is sampled according to a binomial distribution, and are able to recover the secret perfectly in the vast majority of cases for much higher measurement noise, up to a standard deviation of 5.

4.3.1 Noise tolerance for uniformly random input

We start by sampling coefficients uniformly at random between $-\lceil q_\mathrm{K}/2 \rceil$ and $\lceil q_\mathrm{K}/2 \rceil$ at the input of the first NTT layer, and we simulate the execution of the NTT and measure each leakage point with some added noise, then use belief propagation on the obtained measurements to attempt to recover the input and intermediate values. We perform that procedure for various standard deviations (denoted σ_M) of the measurement noise, and sample 100 such experiments for each value of the standard deviation. For now, we assume that σ_M is exactly known, and we accordingly configure the factor nodes of types L, V and I such that $\sigma_F = \sigma_M$. These two quantities being equal, we collectively refer to them as σ. In the following, unless otherwise noted, an experiment is said to be *successful* when it gave the highest probability to the actual value of each input and intermediate variable.

The influence of the amount of measurement noise on convergence rate and convergence time can be seen in fig. 4.3. For low measurement noise ($\sigma \leqslant 0.9$), all experiments are successful in less than two hours of CPU time, which only represent a few minutes of wall-clock time since we can solve the optimization problem with up to 32 CPU cores in parallel. Above this value, the success rate stays above 90 % up to a noise standard-deviation of $\sigma = 1.2$. For higher measurement noise, the success rate then quickly drops, and the runtime sharply increases, reaching an average value of six CPU hours per experiment for $\sigma = 1.3$.

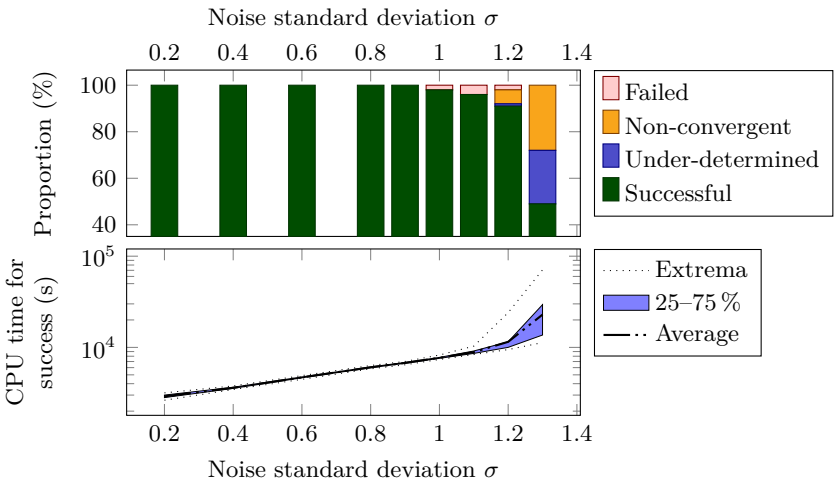


Figure 4.3: Effect of noise variance on convergence rate and convergence time

Starting from $\sigma = 1.2$, the attack starts showing a variety of different behaviors: apart from *successful* experiments, in which belief propagation converges to a low-entropy state (practically, a single candidate secret has nonzero probability), and *failed* experiments, in which an inconsistent state is reached with at least one message having no positive-probability outcomes, two other behaviors are observed, labeled as either *non-convergent* or *under-determined*. We call an experiment non-convergent when belief propagation does not terminate before the iteration limit, and we mark it as under-determined when it does terminate before this limit but with a final state having large residual Shannon entropy (practically, between 6000 and 14 300 bits in the experiments reported in fig. 4.3).

4.3.2 Influence of an incorrect estimation of the amount of noise

Since in practical attacks the standard deviation of the measurement noise is not precisely known, we study the influence of an incorrect estimation thereof on the performance of belief propagation. For a fixed actual value of the standard deviation of the noise and a fixed set of simulated traces, we try to recover their secret values using belief propagation. This time, we sample 100 executions of the NTT, with a fixed measurement noise of standard deviation $\sigma_M = 1.1$, and we attack each of them several times while varying the noise standard deviation σ_F configured for the factor nodes of belief propagation.

The results of this experiment are reproduced in fig. 4.4. Let us first analyze the quality of the outcome of belief propagation. With the same termination conditions as before, we measure the quality of the attack results by the number of variables from the input layer of the NTT, whose most-probable outcome in the final state is the actual value of the variable. Since we are attacking a 256-point NTT, there are 256 first-layer variables to be recovered. As expected, the best-quality results are obtained when the noise assumed by the factor graph matches the actual measurement noise, that is, when $\sigma_F = 1.1$, represented with a dashed vertical line in fig. 4.4. In this situation, all variables are

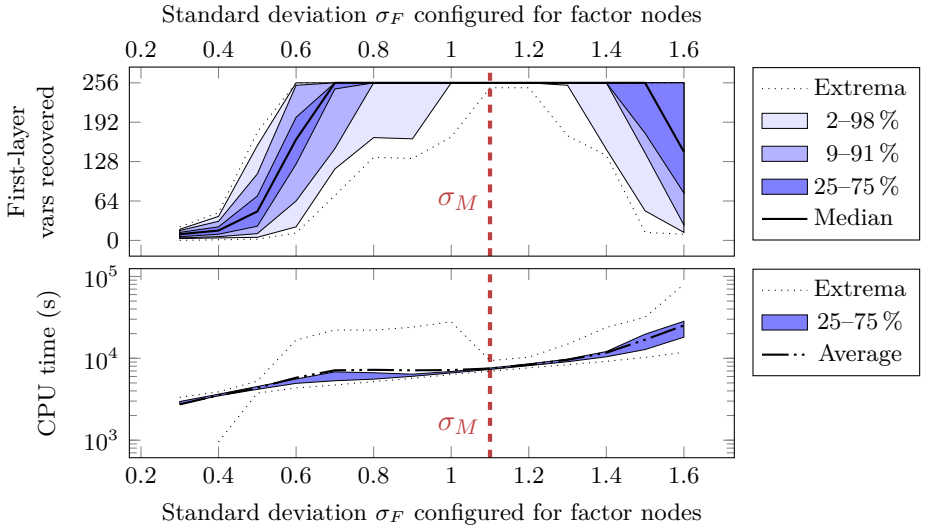


Figure 4.4: Quality and runtime of the attack depending on the noise standard deviation assumed by the factor graph — actual noise standard deviation is $\sigma_M = 1.1$.

successfully recovered in 99 % of the experiments, and 248 out of the 256 input variables are recovered in the remaining one.

We get the exact same quality of results when the measurement noise is slightly overestimated, $\sigma_F = 1.2$. When departing from these values, the results quality progressively decreases, but more than 90 % of the experiments are still perfectly successful for $\sigma_F \in [0.8, 1.4]$. Below and above this range, the results quickly worsen, with only 164 and 144 input variables being recovered in the median case, at $\sigma_F = 0.6$ and $\sigma_F = 1.6$ respectively.

With respect to the runtime of the attack, we notice that the evolution with σ_F differs from the case when σ_F is held equal to σ_M (fig. 4.3): the average runtime of the attack presents a short plateau from $\sigma_F = 0.7$ to $\sigma_F = 1.1$, reaching its minimum at $\sigma_F = 0.9$. Moreover, from $\sigma_F = 0.6$ to $\sigma_F = 1.0$, this average is strongly biased upwards due to a few experiments being non-convergent and taking much more time than usual. Consequently, the average time in this range could be lowered without significantly affecting the results quality by lowering the iteration limit to terminate non-converging experiments faster.

Combining these two parameters of runtime and result quality, we can consider two strategies available to the attacker: if it targets maximum success rate, then closely matching or slightly overestimating the measurement noise is best; if, however, the goal is to minimize the runtime while keeping exact results in the majority of cases, slightly underestimating the noise standard deviation might be best, here selecting $\sigma_F = 0.9$, or $\sigma_F = 0.8$ with earlier termination of non-converging experiments.

4.3.3 NTT over coefficients having a small support

In the above subsections, we assumed the NTT to be run over an input polynomial having uniformly-distributed coefficients between $-\lceil q_K/2 \rceil$ and $\lceil q_K/2 \rceil$. In practice for

the Kyber scheme, this situation only arises when the targeted implementation is using masked arithmetic, like the implementation of Bos *et al.* [BGR+21]. Without masking, the NTT is only ever applied to polynomials having a small support, that is, having coefficients sampled from a centered binomial distribution with small parameter. We now specialize our attack to this situation, by sampling our input polynomials from the centered binomial distribution with parameter $\eta = 3$, and adding this information into the L-type factors of the factor graph. This change of distribution is easily taken care of in the application of Bayes' theorem, with no other changes with respect to section 4.2.1.

Since switching input coefficients from a uniform distribution to a binomial one decreases the Shannon entropy of each from 11.7 to 2.3 bits, it is expected that higher tolerance to measurement noise can be achieved. This effect is made clear in fig. 4.5, where perfect accuracy is obtained for relatively large measurement noise, up to $\sigma = 4$ in all cases but one, and up to $\sigma = 5$ more than 75 % of the time. Above this point, the quality of the results drops: for $\sigma = 6$, only 3 out of 100 experiments recover the first layer perfectly, and 229 coefficients out of 256 are recovered in the median case. Since belief propagation often fails at recovering the first layer exactly when the noise level is that high, the runtime also increases, to an average of more than 18 CPU hours.

We note that the parameter of the binomial distribution was chosen to match the largest one used by Kyber, specifically the η_1 parameter used in its lowest security level, for sampling secret-key elements and one of the polynomial vector involved during encryption. For the other elements and for higher security levels, the binomial distribution has yet smaller support $\eta = 2$, and would allow for an even stronger attack.

Similarly to the study we did in section 4.3.2, we examine how an incorrect estimation σ_F of the amount of measurement noise affects the quality and runtime of belief propaga-

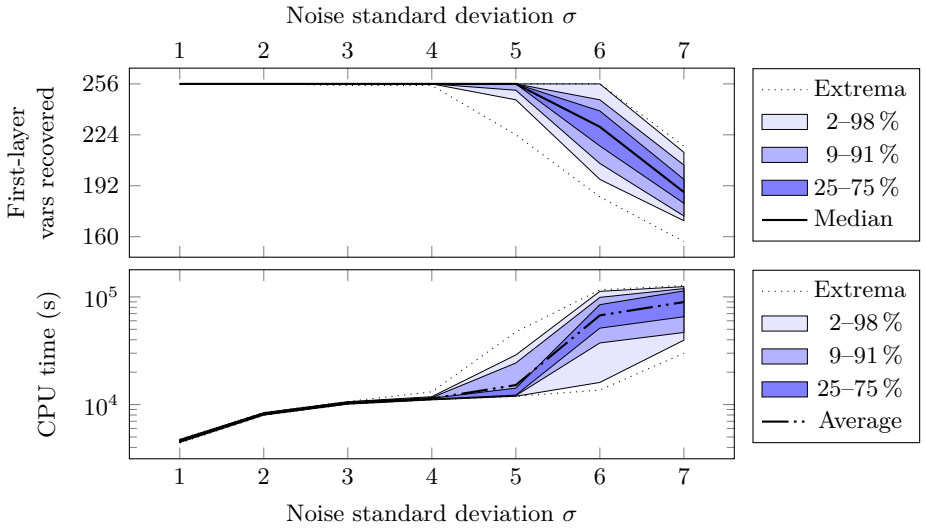


Figure 4.5: Quality and runtime of the attack against an NTT over binomial coefficients with parameter 3

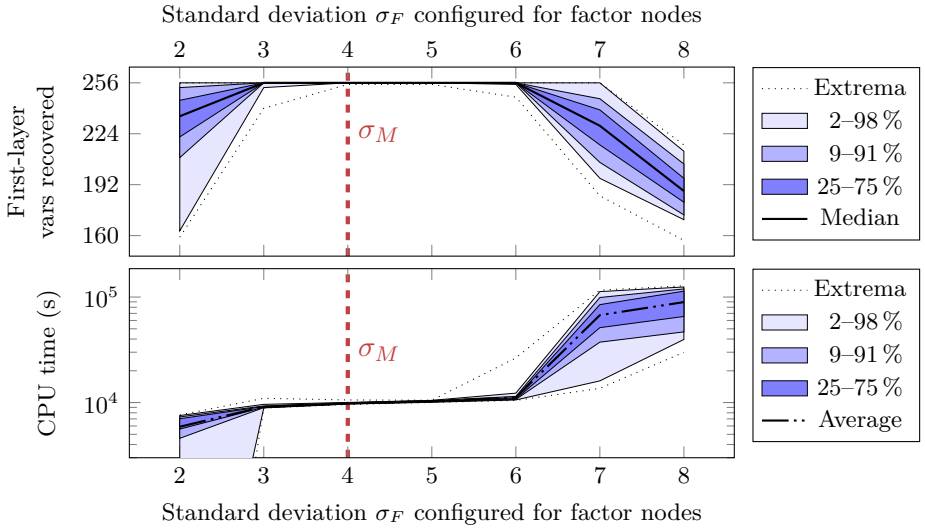


Figure 4.6: Quality and runtime of the attack against an NTT over binomial coefficients with parameter 3, depending on the noise standard deviation configured in the factor graph — actual noise standard deviation is $\sigma_M = 4.0$.

tion: this time, in the case of binomially-distributed input coefficients. We set the actual standard deviation of the measurement noise to $\sigma_M = 4.0$ and run the belief-propagation algorithm with various values of σ_F from 2 to 8. The results of these experiments can be seen in fig. 4.6.

In terms of results quality, we observe largely the same behavior as in the case when input coefficients are uniformly distributed. When the measurement noise is correctly configured or slightly overestimated ($\sigma_F \in [4, 5]$), 99 % of the experiments recover the first layer entirely, and the remaining one recovers 255 coefficients out of 256. In the whole range $\sigma_F \in [3, 6]$, the input coefficients are still recovered perfectly in more than 90 % of the cases. For lower and higher values of σ_F, however, the attack is less successful, with the median case recovering 235 and 229 coefficients at $\sigma_F = 2$ and $\sigma_F = 7$ respectively.

The runtime of belief propagation is closely related to the results quality, with a very regular increase in the range over which the recovery is nearly perfect: from 2.5 CPU hours at $\sigma_F = 3$ to 3.1 CPU hours at $\sigma_F = 6$. When the results quality decreases, the runtime departs from this regular tendency, falling to 1.6 CPU hour on average at $\sigma_F = 2$, and rising to 18.7 hours at $\sigma_F = 7$. Overall, we thus reiterate our remark that a close estimation of the measurement noise is not required, but deliberately over-estimating or under-estimating it can be beneficial in terms of success rate or runtime, respectively.

4.3.4 Masked implementations

To protect implementations of the NTT against multiple-trace attacks, a typical measure is arithmetic masking [RRVV15]. Using this method, each secret value is replaced with two *shares*, each of whose is chosen uniformly at random when considered independently,

but whose sum is equal to the secret. Since the NTT is a linear operation, it can be separately computed over each set of shares of the secret polynomial, and the two resulting polynomials are arithmetic shares of the expected result.

In this situation, the polynomial coefficients at the input of the NTT are no longer distributed over a small support, so it seems that the attacker has to fall back to attacking the NTT over uniformly-distributed coefficients (see section 4.3.1 and section 4.3.2). However, Pessl *et al.* [PP19b] show that the information over the small support can still be used by attacking both share sets at the same time, thus using a factor graph twice the size of the previous one, with additional factor nodes connecting corresponding shares and encoding the information over the small support of their sum. Although their attack tolerates less noise in the masked case than in the unprotected case, perfect success rates can still be achieved against masked implementations having low noise.

We argue that our attack still allows for significant noise in the masked case: indeed, our setup with uniformly-distributed coefficients at the input of the NTT can be applied directly to the masked case by processing shares independently, thus squaring the success rate. We are thus guaranteed to obtain nearly-perfect success up to $\sigma = 1.0$. Specializing the attack to the masked case as described by Pessl *et al.* [PP19b] can only bring further improvements to noise tolerance.

4.4 Practical significance

4.4.1 Attack exploitation

Our attack can recover the polynomial provided at the input of the forward-NTT operation. From section 4.1.3, we can see that the attack allows for recovering the one-time secret $\mathbf{r}$ used in encryption (thus allowing to recover the corresponding plain-text message), or the long-term private key $\mathbf{s}$ (allowing to decrypt all past and future ciphertexts for this key pair), depending on which NTT operation is targeted.

When trying to perform message recovery, the attacker should target the NTT or $\mathbf{r}$ at line 4 of algorithm 4.2. Having recovered $\mathbf{r}$, it is then possible to compute the message as $m = (v - \mathbf{t}^\mathsf{T}\mathbf{r})/\lfloor q_\mathrm{K}/2 \rceil$, where v is part of the ciphertext and $\mathbf{t}$ belongs to the public key.

Much more powerful is a recovery of the long-term private key, which can clearly be carried out during key generation. Attacking the NTT of either $\mathbf{s}$ or $\mathbf{e}$ (lines 4 and 5 of algorithm 4.1) leaks the private vector, either directly or through $\mathbf{s} = \mathbf{A}^{-1}(\mathbf{t} - \mathbf{e})$. This attack can thus be applied at two points during key generation; however, since key generation is only supposed to occur once for a given key pair, and since it might be performed in a more controlled environment than subsequent cryptographic operations, such realization may be difficult for the attacker.

We note that in these practical realizations, the attack must be run k times since k-element vectors have to be recovered. Consequently, for the lowest security level of Kyber ($k = 2$), the success rate of the whole attack would be the square of the success rate of attacking the NTT of each polynomial.

Since the decapsulation of Ind-CCA Kyber involves an encryption step, which is subject to the message recovery attack described above, private-key recovery can also

be performed during decapsulation, with the help of chosen ciphertexts. Indeed, Ngo *et al.* [NDGJ21] devised a message-recovery attack on the Saber KEM [DKR+20] and introduced techniques to convert it into a key-recovery attack when combined with a very few chosen ciphertexts. They showed 8 chosen ciphertexts to be enough when perfect message recovery is possible, or 16 when message recovery is imperfect. While their attack is applied to Saber rather than Kyber, their conversion of message recovery into private-key recovery only relies on high-level characteristics that are shared by the two schemes. We can thus expect similar efficiency for this conversion in the case of Kyber.

The noise levels that our attack can tolerate in simulation are reasonable, given that most Cortex-M4 microcontrollers have high leakage in practice. Since we are mainly dealing with register leakage, we expect that the high signal-to-noise ratio (SNR) required for the success of the attack in sections 4.3.1 and 4.3.2 can only be obtained with a high-quality measurement setup and a high templating effort, but still within the capabilities of a determined attacker. On the other hand, the low SNR that can be accommodated by the attack in section 4.3.3 can be obtained with low-cost equipment and minimal templating. We also stress out that the Hamming-weight leakage model is relevant, as the SNR in this metric generally dominates the SNR in the Hamming-distance metric on these microcontrollers.

In cases when the actual SNR is too low for the attack to succeed, the attacker may always switch to higher-quality equipment, for instance switching from power measurement to electromagnetic measurement, and increase the templating effort. In some cases, such as when attacking the decapsulation of chosen ciphertexts, it is also possible to average the acquired traces over several decapsulations of the same ciphertext, thereby allowing for arbitrarily low noise with sufficiently many traces. We can thus confidently assert that the attack presented here is bound to eventually succeed unless effective countermeasures are used against it.

4.4.2 Countermeasures

As proved in [PP19b] and discussed above, masking is not a sufficient countermeasure to the attack since it merely decreases the maximum allowable noise. However, since the belief-propagation technique relies on correctly associating each leakage point with a particular variable (or set of variables), shuffling has already been proposed as an effective countermeasure [PP19b]. While Hermelink *et al.* [HSST23] have adapted belief propagation to attacking shuffled NTTs, their attack assumes a much more powerful adversary, able to inject reliable zeroing faults. It is also less tolerant to noise, and most importantly, it requires either the shuffling to be partial, or the attacker to get sufficient information on the shuffling order to reduce the situation to the partial-shuffling case.

In a different context, Goy *et al.* [GMGL24] have mentioned shuffling as a countermeasure to belief-propagation attacks. While their work focuses on a code-based scheme, HQC [AAB+22], their conclusion resembles that of Hermelink *et al.*: fine and intermediate levels of shuffling fail to provide adequate protection, but a full shuffling of the operation can quickly make the attack unreasonably costly.

In addition to the already high difficulty of attacking a shuffled NTT, the situation can be further worsened for the attacker by combining shuffling with masking, and selecting different shuffling orders for the two sets of shares (in the case of first-order masking). In this case, corresponding shares cannot be easily associated, so the information on the narrow support of their sum is lost. For even better protection, the shuffling can be applied to the two sets of shares simultaneously so that they are randomly interleaved, thereby forcing the attacker to consider a much larger number of possible permutations.

4.5 Conclusions

In this chapter, we demonstrated a side-channel attack against the NTT from a recognized software implementation of Kyber for ARM Cortex-M4 microcontrollers. By exploiting precise knowledge of the arithmetic instructions used for intermediate computations, we were able to cope with high noise levels in simulation, that should translate to effective attacks against real-life devices in a wide range of conditions.

As future work, our attack can be optimized to the case when the NTT is protected with arithmetic masking, using the technique of Pessl *et al.* [PP19b]. By attacking both shares of the masked polynomial simultaneously and adding factors that give information on the small support of the sum of shares of each coefficient, we should be able to obtain intermediate performance between the unmasked cases of uniformly-distributed input coefficients (fig. 4.3) and binomially-distributed input coefficients (fig. 4.5).

Since the performance results of our attack are based on simulated traces only, a follow-up to our work will be to evaluate it in practice, with real traces. Given that we use a standard and relatively weak leakage model based on the Hamming weight of the secrets and intermediates, we expect the attack to be effective in practice. However, while we hypothesized equal noise for all leakage points in our simulation, we anticipate significant variations of the actual noise levels across different leakage points, in particular since some of them involve values stored in RAM while others involve values stored in registers. This possible unbalance will have to be taken into account for practical attacks to be best effective.

For high noise levels ($\sigma \in [4, 6]$) when input coefficients are binomially distributed, our attack correctly recovers many, but not always all of the secret coefficients. In this case, lattice reduction can be used to recover the remaining coefficients, as shown by Pessl *et al.* [PP19b]. Finally, some room is left for optimizing the runtime and memory usage of our implementation of belief propagation, which would allow us to attack slightly more noisy instances with the same resource cost.

5 Masked modular addition over Boolean shares

It has been shown in § 3.6.2.1 that protected implementations of LWE-based cryptography schemes crucially rely on masking conversions to switch between Boolean and arithmetic masking. As mentioned, a convenient way to implement these conversions is to rely on secure arithmetic addition over Boolean shares. In the case of Kyber, Dilithium, and other schemes relying on prime-order finite fields, these conversions should handle prime moduli, for which secure *modular* addition is needed. While secure addition modulo an arbitrary integer is usually implemented as two consecutive or parallel secure power-of-two additions [BBE+18, SPOG19, FBR+22], we introduced in [AEV24] an area-efficient secure modular adder that mostly merges the two additions, so that modular addition uses barely more area and latency that power-of-two addition. The current chapter reproduces this work and is mostly taken as-is from [AEV24].

5.1 Introduction

When designing masked circuits, it is particularly important to evaluate their security in models that are both accurate and as implementation-agnostic as possible. The first formal basis for evaluating the security of such circuits, the *t-threshold probing model*, was introduced by Ishai, Sahai and Wagner [ISW03]. An important limitation of this notion was that it did not take into consideration the effects of *glitches* and *transitions*, two imperfections of physical circuits that made the model insufficient in practice. Their work was consequently extended by Faust *et al.* [FGP+18] to account for this kind of defects, giving the *robust-probing model*. Other leakage models such as the *noisy-leakage* model [CJRR99, PR13], which describes side-channel leakage as a noisy function of the processed secrets, are closer to practice but more difficult to study theoretically. The work of Duc *et al.* [DDF14], among others, bridges the gap by providing a reduction from the threshold-probing model to the noisy-leakage model, a result extended upon by Cassiers *et al.* [CFOS21]. Since the practical relevance of the threshold-probing model is thus being established in its robust version, several works have proposed incremental security notions to allow for the secure construction of complex masked circuits from smaller components (so-called *gadgets*) using a composition approach: in particular, the work of Cassiers and Standaert [CS21] which proposes trivial composability in the glitch+transition-robust probing model.

In hardware implementations of lattice-based cryptography, conversions between Boolean and arithmetic masking are important contributors to the overall implementation cost, both in terms of area and latency [FBR+22]: resource sharing is therefore critical

for low-cost implementations. Reusing hardware in this context has been partially shown by Fritzmann *et al.* [FBR+22], but they valued high performance over low area, which is undesirable in the context of embedded systems.

Our contribution We describe an area-efficient masked circuit computing secure modular addition over Boolean shares. The proposed hardware gadget can be configured at runtime for addition or subtraction, either modulo a publicly known arbitrary integer or modulo a power of two. We prove first-order security for this construction in a robust-probing model accounting for glitches and transitions. Then, we synthesize it as an application-specific integrated circuit (ASIC), and show that it exhibits no leakage in simulation.

Outline We start by introducing in section 5.2 the background underlying this work. In section 5.3, we describe our proposed construction and formally prove its security (two longer proofs are deferred to section 5.A), and we highlight in section 5.4 how our work compares with the state of the art. We then perform a leakage assessment on simulated executions of our design in section 5.5, and conclude in section 5.6.

5.2 Preliminaries

5.2.1 Notations and terminology

In this chapter, we will not use integer multiplication, but will make a heavy use of multiplication in $\mathbb{F}_2^w$ (bitwise AND). Consequently, implicit multiplication (juxtaposition) will denote bitwise AND in this chapter. We keep the same symbol as before when making this operation explicit: $\odot$, and also keep symbols $\oplus$ for sum in $\mathbb{F}_2^w$ (bitwise XOR) and $|$ for bitwise OR. The one's complement (bitwise negation) of a is $\overline{a}$. In contrast, $+$ and $-$ express arithmetic addition and subtraction in $\mathbb{Z}$. Arithmetic operations on bit strings assume a binary representation using two's complement for negative numbers.

Consider a bit string $s \in \mathbb{F}_2^m$. For $0 \leqslant i \leqslant m-1$, we denote by $s[i]$ the ith bit of s.

We recall that for a quantity represented by shares $a^{\mathbf{0}}, \ldots, a^{\boldsymbol{d-1}}$, the set of all shares is represented by $a^\star$. The symbol without any share index, a, stands for the unmasked value. In this chapter, we will only consider Boolean masking, for which shares are recombined with Boolean addition: $a = \bigoplus_{\boldsymbol{i}} a^{\boldsymbol{i}}$.

If S is a set, $a \xleftarrow{\$} S$ denotes sampling uniformly at random the value of a from set S. We denote by $\mathrm{Reg}[\cdot]$ a register, which is a sequential gate delaying its input by one clock cycle; given a shared value $x^\star \in \mathcal{B}^2(\mathbb{F}_2)$ and a random bit $r \in \mathbb{F}_2$, we define $\mathrm{Refresh}(x^\star, r) = (x^{\mathbf{0}} \oplus r, x^{\mathbf{1}} \oplus r)$.

5.2.2 Security model

We want to formally prove the security of our constructions in a probing model that represents as closely as possible the behavior of ASIC implementations of secure hardware, in particular by including the effects of *glitches* and *transitions*. The former refers to the

progressive and uneven propagation of values across combinational logic, which causes the gates to switch several times before reaching their final value, potentially leaking secrets [FG05]. The latter reflects that logic gates and wires leak depending not only their logic level, but also on their switching, in particular over successive clock cycles, which can also cause vulnerabilities in masked implementations. We thus consider both hardware defects, through the attacker model of *glitch+transition-robust probing* [CS21].

Since the direct security analysis of complex masked circuits is often infeasible, formal security proofs usually rely on composability notions, where the overall circuit is split into smaller individual *gadgets* for which the security properties are easier to prove. To allow for unrestricted gadget composition in the glitch+transition-robust probing model, Cassiers and Standaert introduced the Output Probe-Isolating Non-Interference (O-PINI) notion [CS21], which is a stronger evolution of their earlier PINI notion [CS20]. We recall the O-PINI notion in the specific context of our work, that is, for gadgets having two shares and thus only targeting first-order security.

Definition 5.1 (Output Probe-Isolating Non-Interference [CS21, Definition 20 with $t = 1$])**.** *A gadget G with 2 shares is O-PINI if and only if for any probe I_1 on its internal wires, there exists a share index i so that the observations corresponding to I_1 and probes on all output shares of index i can be simulated using only the input shares with index i.*

In the glitch+transition-robust probing model, each of the probes mentioned in definition 5.1 must be extended across both glitches and transitions within the considered gadget. A glitch-extended probe on a net leaks the value of all combinational inputs contributing to the value of the net. A transition-extended probe on a net leaks both its current value and its value at the previous clock cycle. The combination of the two is done by first transition-extending each probe, then glitch-extending the resulting set of probes.

The circuit model of Cassiers and Standaert [CS21] is based on the one of Ishai *et al.* [ISW03], with added notions that allow for reusing physical gates to implement different logical functions across clock cycles. We only give a high-level overview of this model here, and refer the reader to [CS21] for formal definitions. At the core of this model are the notions of *structural gates* and *structural wires*, which describe the physical configuration of the circuit, including the latency of sequential gates (flip-flops), as a directed graph. On top of this physical view of a circuit, a logical one describes its behavior over time: a *circuit execution* consists of replications of the gates of a structural circuit at each clock cycle, with wires that connect these replicas according to the latency of the involved gates.

The notion of gadget in the model of Cassiers and Standaert is twofold: on one hand, a *gadget execution* is a subset of the gates and wires of a circuit execution. Those wires whose source is not included in the gadget are referred to as its *inputs*, and are partitioned into tuples of d elements, d being the number of shares of the gadget. A gadget furthermore has a set of *outputs* (taken from the outputs of its constituting gates), likewise partitioned into sets of d shares. Disjoint gadget executions can be composed by linking outputs to inputs, respecting the order of shares and ensuring

that the composition graph contains no cycles: the inputs of a gadget execution cannot depend directly or indirectly on one of its outputs.

On the other hand, the notion of *structural gadget* is introduced: it is a set of disjoint gadget executions that are identical except for a translation in time, that is, that all use the same structural gates and wires but at different clock cycles. Distinct structural gadgets must not share any structural gates or wires among them. In the terminology of [CS21], a structural gadget is said to be *pipeline* if its canonical execution uses each of its structural gates and wires only once, which will be the case for all our elementary gadgets.

We note that there exist some automated tools to check the security of masked designs. FullVerif [CGLS21], on one hand, analyzes composite circuits made of gadgets with some known security properties, and checks the global security of the circuit through a composition approach. IronMask [BMRT22] and SILVER [KSM20], on the other hand, analyze the internal construction of gadgets to formally prove their security. However, none of these tools seems to support O-PINI security yet, so they cannot be used to check the glitch+transition-robustness of iterative circuits. This absence of automated tools for our purpose is not a concern, since we prove the security of our individual gadgets by hand, according to a security notion that allows for trivial composition.

5.2.3 Uses of masked addition in lattice-based cryptography

We mentioned already in § 3.6.2.1 that lattice-based cryptography relies on both Boolean logic and modular arithmetic operations. These two classes being somewhat interleaved, masked implementations typically need to perform frequent conversions between Boolean and arithmetic masking. Keeping the cost of these conversions low is thus particularly important. It has been shown, among others, by Fritzmann *et al.* [FBR+22] that masking conversions based on secure addition over Boolean shares (SecAdd [CGV14]) offer ideal versatility and resource efficiency. Precisely, SecAdd and its extension to modular addition proposed by Barthe *et al.* [BBE+18], allow to perform both Boolean-to-Arithmetic and Arithmetic-to-Boolean conversion, where the arithmetic masking may be either modulo a power of two or modulo a prime.

For some protected operations, SecAdd can also be used standalone, without wrapping it in a masking conversion. It is the case, for instance, for the masked norm comparisons needed by protected implementations of Dilithium (see § 3.6.4.5). We will also see in section 6.1 that SecAdd can be used to efficiently implement Kyber masked compression, again without performing explicit masking conversions.

Overall, secure addition over Boolean shares is a versatile building block for many other protected operations. This opportunity for resource sharing is very welcome for embedded systems, which only have a limited budget for side-channel protections.

5.2.4 Binary-addition algorithms

We briefly recall the main architectures for binary addition and highlight their characteristics. The most basic architecture, the ripple carry adder [Mac61], is built from a chain

of n full adders (n being the number of bits of the summands): each, given as input one bit of each summand and an input carry, computes an output bit and an output carry. The full adders are chained, from least to most significant, so that each sends its output carry to the next full adder. The main drawback of this architecture is its long propagation delay, since the input carry ripples through n successive full adders before the sum is complete.

To speed up the propagation of the carry, a carry-select adder [Bed62] can be used: the summands are divided into groups of a smaller width, and two sums are computed for each group of bits: one assuming that the group input carry is set, the other assuming that it is cleared. Then, depending on the actual input carry, the correct sum and output carry are selected for each block. The carry-skip or carry-bypass adder [LB61] similarly divides its input width into groups, but it computes a single sum for each group, once the input carry is available. Only the carry propagation from one block to the next is sped up, thanks to the precomputation of *carry-skip* and *carry-generate* signals for each block.

While the three above architectures have linear latency, addition can also be computed in logarithmic time, using a parallel-prefix adder: this architecture arranges carry-lookahead logic in a tree of logarithmic depth to quickly propagate the carries over groups of increasing size [Skl60]. The drawback of this construction is its larger area, which grows in $O(n \log(n))$.

In unprotected implementations performing addition in a single cycle, it is clear that the ripple-carry adder requires the smallest area and highest latency, while parallel-prefix adders are among the largest and fastest [Mac61, LB61, WT90]. By configuring the size of groups, a wide range of intermediate performances can be obtained from carry-skip, carry-select, and similar architectures [Mac61, Bed62].

While generalizing this comparison to masked implementations is difficult, the available literature confirms a similar trend in terms of area and latency, with slow but small masked ripple carry adders and large but fast masked parallel-prefix adders (table 5.3, table 5.4). We are not aware of any masked implementation of carry-select or carry-skip adders. Previous works [SMG15, FBR+22, BG22, CGM+23] have shown that fully pipelined parallel-prefix adders require a very large area, and converting them to iterative designs while keeping resistance against glitches and transitions would require switching to iterated glitch+transition-robust gadgets, which have higher area and latency than gadgets without this property [CS21, KM22]. We thus do not expect that iterative implementations of parallel-prefix adders can reach a sufficiently low area to be relevant in resource-constrained implementations. This explains why we specifically investigate the ripple-carry adder, whose specific advantages in the case of modular addition are discussed in section 5.3.3.

5.3 Secure modular arithmetic over Boolean shares

In this section, we describe the construction of our masked circuit for secure addition and subtraction. We first recall how modular addition can be performed using regular

addition followed by trial subtraction, and then describe how ripple-carry addition works. We then introduce the nonlinear gadgets we use to implement this operation, prove their security in the robust-probing model, and show exactly how they can be assembled into masked modular addition. We briefly describe how related operations can be implemented with the same circuit: modular subtraction, and both addition and subtraction modulo a power of two. Finally, we recall how secure addition and subtraction can be used as the main tool to implement conversions between Boolean and arithmetic masking.

5.3.1 Modular addition using trial subtraction

Given an integer $q < 2^n$, we can compute the sum of two integers $a, b \in [\![0, q-1]\!]$ modulo q by performing the sum without modular reduction, then subtracting q to attempt modular reduction, and selecting which of these two results is valid based on the sign in the output of the subtraction, as described in eq. (5.1):

$$(a+b) \bmod q = \begin{cases} (a+b)-q & \text{if } a+b-q \geqslant 0, \\ (a+b) & \text{otherwise.} \end{cases} \tag{5.1}$$

When using an n-bit adder with an output carry, the full precision of addition $a+b$ can be kept, but it is not the case for the subtraction of q since it would require subtracting from an $(n+1)$-bit quantity. It is however possible to express this condition in a different way. Since $a + b < 2q < 2^n + q$, modular reduction must be performed exactly when either of the two mutually exclusive conditions in eq. (5.2) is true:

$$a + b \geqslant 2^n \quad \text{or} \quad ((a+b) \bmod 2^n) + (2^n - q) \geqslant 2^n. \tag{5.2}$$

Modular subtraction is similar, except that the condition for selecting between the two results is known directly after the first subtraction. If the carry is cleared, which happens when the result is negative, then a modular reduction has to be performed by adding q to the difference, as in eq. (5.3):

$$(a-b) \bmod q = \begin{cases} a-b & \text{if } a-b \geqslant 0, \\ a-b+q & \text{otherwise.} \end{cases} \tag{5.3}$$

5.3.2 Secure ripple-carry addition

We now describe the algorithm that we use for secure modular addition: as discussed previously, we use ripple-carry addition since it best fits our aim of low area utilization. We show in algorithm 5.1 how ripple-carry addition is performed, based on shift registers to rotate the operands by one bit at a time. An output carry is provided by the operation.

Given the nature of carry propagation, repeated summation (accumulation) operations can be intermeshed: in the case of modular addition where quantities $a + b$ (which we call *raw sum*) and $(a+b)+(2^n-q)$ (named *offsetted sum*) must be computed, the second quantity can be computed simultaneously with the first, without waiting for the first

Algorithm 5.1: Ripple-carry adder over n bits

Input: $augend \in \mathbb{F}_2^n$, $addend \in \mathbb{F}_2^n$
Output: $sum \in \mathbb{F}_2^n$, $c \in \mathbb{F}_2$ such that $(c \ll n) \mid sum = augend + addend$

1 $sum = 0 \in \mathbb{F}_2^n$
2 $c = 0 \in \mathbb{F}_2$
3 **for** $i = 0$ to $n - 1$ **do**
4 $\quad (z \ll 1) \mid s = augend[0] + addend[0] + c$. // z, $s \in \mathbb{F}_2$ respectively take bit 1, bit 0 of the result
5 $\quad sum = (s \ll (n-1)) \mid (sum \gg 1)$
6 $\quad augend = augend \gg 1$
7 $\quad addend = addend \gg 1$
8 $\quad c = z$
9 **end**
10 **return** sum, c

carry to propagate. Secondly, since shifting the operands right at each cycle frees their most significant bit, the freed positions can store the newly computed bits of the sums. Finally, the choice in eq. (5.1) can be implemented by computing the raw sum $(a + b)$ and the bit-wise difference between the offsetted and raw sums $((a + b) \oplus (a + b - q))$, and optionally adding the latter to the former. These transformations give algorithm 5.2, which computes two simultaneous ripple-carry additions and uses their output carries to perform the modular reduction.

Algorithm 5.2: Addition modulo q using n-bit ripple-carry adders

Input: $augend \in \mathbb{F}_2^n$, $addend \in \mathbb{F}_2^n$
Parameter: Modulus $q \leqslant 2^n$
Output: $sum \in \mathbb{F}_2^n$ such that $sum = (augend + addend) \bmod q$

1 $c, d = 0, 0$
2 **for** $i = 0$ to $n - 1$ **do**
3 $\quad (z \ll 1) \mid s = augend[0] + addend[0] + c$ // z, $s \in \mathbb{F}_2$ respectively take bit 1, bit 0 of the result
4 $\quad (\xi \ll 1) \mid \rho = s + (2^n - q)[i] + d$ . . . . // ξ, $\rho \in \mathbb{F}_2$ respectively take bit 1, bit 0 of the result
5 $\quad augend = (s \ll (n-1)) \mid (augend \gg 1)$ . . . . . // Shift augend and save new raw-sum bit
6 $\quad addend = ((\rho \oplus s) \ll (n-1)) \mid (addend \gg 1)$ // Save difference between raw and offsetted sum bits
7 $\quad c, d = z, \xi$ . . . . . // Use the new carries as input for next iteration
8 **end**
9 **return** $augend \oplus (addend$ **if** $c \oplus d$ **else** $0)$

5.3.3 Secure modular addition over Boolean shares

By implementing all operations of algorithm 5.2 using masked gadgets over Boolean sharings, we can compute modular addition securely over Boolean-masked values. This construction is shown in algorithm 5.3, where we assume the presence of secure gadgets SDFA_m, which securely computes the sum and carry bits at lines 3 and 4 of algorithm 5.2 (where parameter m successively holds the bits of $2^n - q$ from least to most significant),

and SMx_n, which outputs its first operand, conditionally XORed with its second operand (each have n bits) depending on the value of the third (a Boolean sharing of a single bit).

Algorithm 5.3: Secure modular addition over Boolean sharings

Input: $augend^\star \in \mathcal{B}^2(\mathbb{F}_2^n)$, $addend^\star \in \mathcal{B}^2(\mathbb{F}_2^n)$
Parameter: $n \in \mathbb{N}$, modulus $q \in [\![1, 2^n]\!]$ where 2^n is represented as the all-0 bit string
Output: $sum^\star \in \mathcal{B}^2(\mathbb{F}_2^n)$ such that $sum = (augend + addend) \bmod q$

1 $c^\star = 0 \in \mathcal{B}^2(\mathbb{F}_2)$ . . . . . *// Initial carry for raw sum*
2 $d^\star = 0 \in \mathcal{B}^2(\mathbb{F}_2)$ . . . . . *// Initial carry for offsetted sum*
3 **for** $i = 0$ to $n - 1$ **do**
4 $\quad m = (2^n - q)[i]$
$\quad$ *// Compute the sum and carry bits for the raw and offsetted sums*
5 $\quad s^\star, z^\star, \delta^\star, \xi^\star = \mathrm{SDFA}_m(augend^\star[0], addend^\star[0], c^\star, d^\star)$
$\quad$ *// Rotate the operand registers and store the sum bits at their top*
6 $\quad augend^\star = (s^\star \ll (n-1)) \mid (augend^\star \gg 1)$
7 $\quad addend^\star = (\delta^\star \ll (n-1)) \mid (addend^\star \gg 1)$ . . . . *// δ corresponds to $\rho \oplus s$ in algorithm 5.2*
8 $\quad c^\star, d^\star = z^\star, \xi^\star$ . . . . . *// Use the new carries as input carries for next iteration*
9 **end**
10 $e^\star = c^\star \oplus d^\star \in \mathcal{B}^2(\mathbb{F}_2)$
11 **return** $\mathrm{SMx}_n(augend^\star, addend^\star, e^\star)$

Fritzmann *et al.* [FBR+22] use a different approach for secure modular addition: by assuming that modulus q has already been subtracted from one of the summands before secure addition, the carry output by the first addition already indicates whether modular reduction should be performed. The second secure addition then conditionally applies this reduction by adding either q or 0. This method is more efficient in their work since it avoids the need for a secure multiplexer. However, the two computationally expensive secure additions remain, and they can no longer be computed in parallel since the second depends on the output carry of the first. That method would thus be highly suboptimal in our setting.

In algorithm 5.3, we have intentionally hidden the latency of operations for better clarity. This latency will be made explicit in § 5.3.3.3. We now describe the inner gadgets of secure modular addition, and prove their security in the robust-probing model.

5.3.3.1 Secure sum and carry computation

We first study SecDualFullAdder, which implements the secure computation of the sum and carry bits at line 5 of algorithm 5.3. To get a glitch+transition-robust O-PINI gadget having reasonable latency, we need to integrate all four computations into an atomic gadget. Furthermore, in order to avoid unnecessary duplication of resources, the gadget implements other operations: besides sum and carry computation (operation SDFA), it provides register operations (SDFAin and SDFAcp with one and two cycles of latency respectively) as well as a secure-selection operation (SDFAmx). We show in fig. 5.1 a schematic representation of the gadget and describe in algorithm 5.4 the logic equations

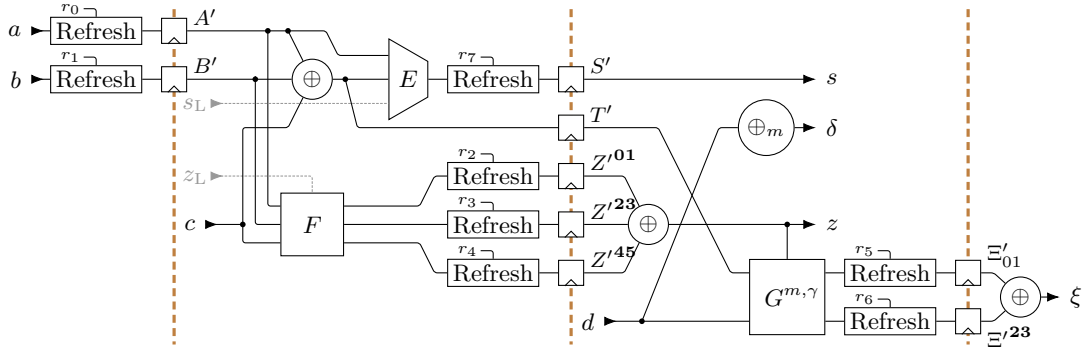


Figure 5.1: The SecDualFullAdder gadget. Given masked inputs a, b, input carry c, and input offsetted carry d, the SDFA operation of the gadget securely computes raw-sum bit s, offset bit δ, output raw carry bit z, and output offsetted carry ξ. Each edge is a 2-bit bus. Node $\oplus_m$ indicates Boolean addition of m (the ith bit of the negated modulus) to the first share.

Algorithm 5.4: SecDualFullAdder

Input: Shares $a^\star, b^\star \in \mathcal{B}^2(\mathbb{F}_2)$ with latency 0, shares $c^\star, s_L^\star, z_L^\star \in \mathcal{B}^2(\mathbb{F}_2)$ with latency 1, shares $d^\star \in \mathcal{B}^2(\mathbb{F}_2)$ with latency 2

Parameter: $m, \gamma \in \mathbb{F}_2$, $E = (E^0, E^1) \in (\mathbb{F}_2^4 \to \mathbb{F}_2)^2$,
$F = (F^0, \ldots, F^5) \in (\mathbb{F}_2^4 \to \mathbb{F}_2)^2 \times (\mathbb{F}_2^3 \to \mathbb{F}_2)^2 \times (\mathbb{F}_2^2 \to \mathbb{F}_2)^2$,
$G_{m,\gamma} = (G^0_{m,\gamma}, \ldots, G^3_{m,\gamma}) \in (\mathbb{F}_2^3 \to \mathbb{F}_2)^2 \times (\mathbb{F}_2^2 \to \mathbb{F}_2)^2$

Input randomness: $r_0, \ldots, r_7 \in \mathbb{F}_2$

Output: Shares $s^\star, z^\star, \delta^\star \in \mathcal{B}^2(\mathbb{F}_2)$ with latency 2 and shares $\xi^\star \in \mathcal{B}^2(\mathbb{F}_2)$ with latency 3, such that $s = E(a, a \oplus b \oplus c, s_L)$, $z = F(A, B, c, z_L)$, $\delta = d \oplus m$, $\xi = G_{m,\gamma}(s, d, z)$

1 $A^\star = \mathrm{Reg}[\mathrm{Refresh}(a^\star, r_0)]$
2 $B^\star = \mathrm{Reg}[\mathrm{Refresh}(b^\star, r_1)]$
3 $T^\star = A^\star \oplus B^\star \oplus c^\star$
4 $S^\star = E^0(A^{\mathbf{0}}, T^{\mathbf{0}}, s_L^{\mathbf{0}}), E^1(A^{\mathbf{1}}, T^{\mathbf{1}}, s_L^{\mathbf{1}})$
5 $Z^{\mathbf{0}}, Z^{\mathbf{1}} = F^0(A^{\mathbf{0}}, B^{\mathbf{0}}, c^{\mathbf{0}}, z_L^{\mathbf{0}}), F^1(A^{\mathbf{1}}, B^{\mathbf{1}}, c^{\mathbf{1}}, z_L^{\mathbf{1}})$
6 $Z^{\mathbf{2}}, Z^{\mathbf{3}} = F^2(A^{\mathbf{1}}, B^{\mathbf{0}}, c^{\mathbf{0}}), F^3(A^{\mathbf{0}}, B^{\mathbf{1}}, c^{\mathbf{1}})$
7 $Z^{\mathbf{4}}, Z^{\mathbf{5}} = F^4(B^{\mathbf{1}}, c^{\mathbf{0}}), F^5(B^{\mathbf{0}}, c^{\mathbf{1}})$
8 $T'^\star = \mathrm{Reg}[T^\star]$
9 $s^\star = \mathrm{Reg}[\mathrm{Refresh}(S^\star, r_7)]$
10 $z^\star = \mathrm{Reg}[\mathrm{Refresh}((Z^{\mathbf{0}}, Z^{\mathbf{1}}), r_2)] \oplus \mathrm{Reg}[\mathrm{Refresh}((Z^{\mathbf{2}}, Z^{\mathbf{3}}), r_3)] \oplus \mathrm{Reg}[\mathrm{Refresh}((Z^{\mathbf{4}}, Z^{\mathbf{5}}), r_4)]$
11 $\delta^\star = d^\star \oplus (m, 0)$
12 $\Xi^{\mathbf{0}}, \Xi^{\mathbf{1}} = G^0_{m,\gamma}(T'^{\mathbf{0}}, d^{\mathbf{0}}, z^{\mathbf{0}}), G^1_{m,\gamma}(T'^{\mathbf{1}}, d^{\mathbf{1}}, z^{\mathbf{1}})$
13 $\Xi^{\mathbf{2}}, \Xi^{\mathbf{3}} = G^2_{m,\gamma}(T'^{\mathbf{1}}, d^{\mathbf{0}}), G^3_{m,\gamma}(T'^{\mathbf{0}}, d^{\mathbf{1}})$
14 $\xi^\star = \mathrm{Reg}[\mathrm{Refresh}((\Xi^{\mathbf{0}}, \Xi^{\mathbf{1}}), r_5)] \oplus \mathrm{Reg}[\mathrm{Refresh}((\Xi^{\mathbf{2}}, \Xi^{\mathbf{3}}), r_6)]$
15 **return** $s^\star, z^\star, \delta^\star, \xi^\star$

Table 5.1: Unmasked output equations of SecDualFullAdder based on configuration

Output	SDFA	SDFAin	SDFAcp	SDFAmx
s	$a \oplus b \oplus c$	s_{L}	a	s_{L}
z	$ab \oplus bc \oplus ac$	z_{L}	b	$a \oplus bc$
δ	$d \oplus m$	$d \oplus m$	$d \oplus m$	$d \oplus m$
ξ	$(sd \oplus sm \oplus dm) \oplus \gamma z$	0	d	0

Table 5.2: Parameterization of SecDualFullAdder depending on configuration

Internal function	SDFA	SDFAin	SDFAcp	SDFAmx
$E^0(A^{\mathbf{0}}, T^{\mathbf{0}}, s_{\mathrm{L}}^{\mathbf{0}})$	$T^{\mathbf{0}}$	$s_{\mathrm{L}}^{\mathbf{0}}$	$A^{\mathbf{0}}$	$s_{\mathrm{L}}^{\mathbf{0}}$
$E^1(A^{\mathbf{1}}, T^{\mathbf{1}}, s_{\mathrm{L}}^{\mathbf{1}})$	$T^{\mathbf{1}}$	$s_{\mathrm{L}}^{\mathbf{1}}$	$A^{\mathbf{1}}$	$s_{\mathrm{L}}^{\mathbf{1}}$
$F^0(A^{\mathbf{0}}, B^{\mathbf{0}}, c^{\mathbf{0}}, z_{\mathrm{L}}^{\mathbf{0}})$	$A^{\mathbf{0}}B^{\mathbf{0}} \oplus A^{\mathbf{0}}c^{\mathbf{0}} \oplus B^{\mathbf{0}}c^{\mathbf{0}}$	$z_{\mathrm{L}}^{\mathbf{0}}$	$B^{\mathbf{0}}$	$A^{\mathbf{0}} \oplus B^{\mathbf{0}}c^{\mathbf{0}}$
$F^1(A^{\mathbf{1}}, B^{\mathbf{1}}, c^{\mathbf{1}}, z_{\mathrm{L}}^{\mathbf{1}})$	$A^{\mathbf{1}}B^{\mathbf{1}} \oplus A^{\mathbf{1}}c^{\mathbf{1}} \oplus B^{\mathbf{1}}c^{\mathbf{1}}$	$z_{\mathrm{L}}^{\mathbf{1}}$	$B^{\mathbf{1}}$	$A^{\mathbf{1}} \oplus B^{\mathbf{1}}c^{\mathbf{1}}$
$F^2(A^{\mathbf{1}}, B^{\mathbf{0}}, c^{\mathbf{0}})$	$A^{\mathbf{1}} \circ (B^{\mathbf{0}} \oplus c^{\mathbf{0}})$	0	0	0
$F^3(A^{\mathbf{0}}, B^{\mathbf{1}}, c^{\mathbf{1}})$	$A^{\mathbf{0}} \circ (B^{\mathbf{1}} \oplus c^{\mathbf{1}})$	0	0	0
$F^4(B^{\mathbf{1}}, c^{\mathbf{0}})$	$B^{\mathbf{1}}c^{\mathbf{0}}$	0	0	$B^{\mathbf{1}}c^{\mathbf{0}}$
$F^5(B^{\mathbf{0}}, c^{\mathbf{1}})$	$B^{\mathbf{0}}c^{\mathbf{1}}$	0	0	$B^{\mathbf{0}}c^{\mathbf{1}}$
$G^0_{m,\gamma}(T'^{\mathbf{0}}, d^{\mathbf{0}}, z^{\mathbf{0}})$	$(T'^{\mathbf{0}}m \oplus d^{\mathbf{0}}m \oplus T'^{\mathbf{0}}d^{\mathbf{0}}) \oplus \gamma z^{\mathbf{0}}$	0	$d^{\mathbf{0}}$	0
$G^1_{m,\gamma}(T'^{\mathbf{1}}, d^{\mathbf{1}}, z^{\mathbf{1}})$	$(T'^{\mathbf{1}}m \oplus d^{\mathbf{1}}m \oplus T'^{\mathbf{1}}d^{\mathbf{1}}) \oplus \gamma z^{\mathbf{1}}$	0	$d^{\mathbf{1}}$	0
$G^2_{m,\gamma}(T'^{\mathbf{1}}, d^{\mathbf{0}})$	$T'^{\mathbf{1}}d^{\mathbf{0}}$	0	0	0
$G^3_{m,\gamma}(T'^{\mathbf{0}}, d^{\mathbf{1}})$	$T'^{\mathbf{0}}d^{\mathbf{1}}$	0	0	0

that are common to all operations[1], using black-box functions E, F and G, which are public parameters. The unmasked computation carried out by each operation is specified in table 5.1, with table 5.2 fully defining the contents of the black-box functions to achieve it.

One full execution of SDFA needs to spread over four cycles so it can achieve its target security (robust O-PINI). The inputs and outputs are distributed over these cycles to minimize the overall latency of secure modular addition, by presenting a single cycle of latency from the carry inputs (c and d) to the corresponding carry outputs (z and ξ). Initially, sharings of one bit of each summand are provided through the a and b inputs, and immediately refreshed. After a first register barrier, the raw input carry, c, is provided, and the masked computation of the raw sum and carry bits (S, Z) is performed. These are refreshed and registered in the second barrier, after which they are output as s and z. A second, independent sharing of s is stored as T', to be used for the nonlinear computation of the offsetted carry bit, Ξ. This computation also involves the input offsetted carry, d, which is provided at the same cycle. After the third register barrier, the refreshed shares of the offsetted carry, $\xi^\star$, are output.

To simplify the composition diagram, two additional linear computations are integrated within SDFA, although they could be performed externally without affecting the security

[1]The algorithm listings specialized to each of the four operations of the gadget are given in section 5.B.

proofs: the computation of the offset, $\delta = d \oplus m$, and the Boolean addition of the two carries z and ξ in the last execution of SDFA, to decide whether modular reduction should be performed (see line 10). This latter result overwrites the ξ output of SDFA when parameter γ equals 1.

The correctness of this gadget is easily checked by recursively evaluating equations, for instance for output ξ of the SDFA operation:

$$\begin{aligned}\xi^{\mathbf{0}} \oplus \xi^{\mathbf{1}} &= \bigoplus_{i=0}^{3} \Xi^{i} = (T'^{\mathbf{0}} \oplus T'^{\mathbf{1}} \oplus d^{\mathbf{0}} \oplus d^{\mathbf{1}}) \circ m \oplus (T'^{\mathbf{0}} \oplus T'^{\mathbf{1}}) \circ (d^{\mathbf{0}} \oplus d^{\mathbf{1}}) \oplus \gamma \circ (z^{\mathbf{0}} \oplus z^{\mathbf{1}}) \\ &= ((S^{\mathbf{0}} \oplus S^{\mathbf{1}}) \oplus d) \circ m \oplus (S^{\mathbf{0}} \oplus S^{\mathbf{1}}) \circ d \oplus \gamma z = (sm \oplus sd \oplus dm) \oplus \gamma z\end{aligned}$$

which corresponds to taking the carry bit of $s + d + m$, and furthermore adding it with z over $\mathbb{F}_2$ when $\gamma = 1$. We now prove its security in the model of [CS21].

Proposition 5.1. SecDualFullAdder *is glitch-robust O-PINI.*

The proof of proposition 5.1 is provided in section 5.A. From this proposition, we deduce proposition 5.2.

Proposition 5.2. SecDualFullAdder *is iterated glitch+transition-robust O-PINI.*

Proof. Since SecDualFullAdder is pipeline and glitch-robust O-PINI, by [CS21, Lemma 2], it is also iterated glitch+transition-robust O-PINI. □

5.3.3.2 Masked multiplexer and dual-register gadget

The summands must be stored in a shift register in order to be sent bit by bit to SecDualFullAdder; moreover, as discussed earlier, the same shift registers are progressively loaded with the calculated bits of the raw sum and the *offset* (the exclusive-or of the raw and offsetted sums). Furthermore, modular addition and subtraction involve optionally XORing the offset with the raw sum, this selection being performed securely. To save area, we implement both the shift registers and the secure selector using the same physical registers, as two configurations of a single gadget, which we call SecMux_n for n-bit size.

The construction of the SecMux_2 gadget is given in fig. 5.2 and its internal equations are laid out in algorithm 5.5. In the secure-multiplexer configuration (SMx), the gadget takes a Boolean sharing $s^\star$ of a single bit, as well as Boolean sharings $a^\star$ and $b^\star$ of two n-bit quantities at the next cycle, and outputs a sharing of either a or $a \oplus b$ depending on s by computing $a \oplus bs$ (where s is broadcast to all bits of b). In the dual-register configuration (SDR), the gadget refreshes its a and b inputs into output sharings x and y after a one-cycle delay. Input s is ignored, and output z is unused.

Proposition 5.3. SecMux *is glitch-robust O-PINI.*

From this first property, whose proof is provided in section 5.A, we deduce the full desired security in proposition 5.4.

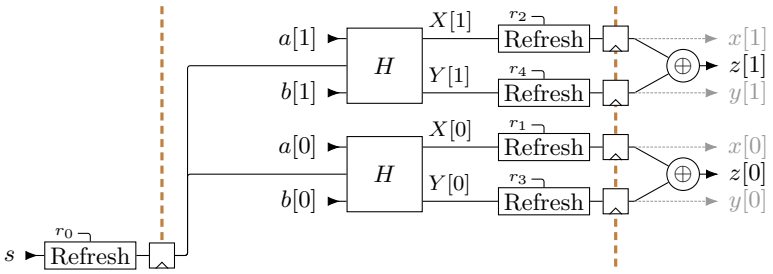


Figure 5.2: The SecMux$_2$ gadget over 2-bit values: given Boolean sharings of $a \in \mathbb{F}_2^2$, $b \in \mathbb{F}_2^2$ and $s \in \mathbb{F}_2$, this gadget outputs as $z \in \mathbb{F}_2^2$ a sharing of either a or $a \oplus b$ depending on the value of s. Outputs $x \in \mathbb{F}_2^2$ and $y \in \mathbb{F}_2^2$ are only used when this gadget is configured to implement a dual register instead of secure selection.

Algorithm 5.5: SecMux$_n$

Input: Shares $s^\star \in \mathcal{B}^2(\mathbb{F}_2)$ with latency 0, shares $a^\star, b^\star \in \mathcal{B}^2(\mathbb{F}_2^n)$ with latency 1.
Parameter: $gadget \in \{\text{SMx}, \text{SDR}\}$
Input randomness: $r_0, \ldots, r_{2n} \in \mathbb{F}_2$
Output: Shares $x^\star, y^\star, z^\star \in \mathcal{B}^2(\mathbb{F}_2^n)$ with latency 2, such that $\begin{cases} gadget = \text{SMx} \Rightarrow z = a \oplus bs \\ gadget = \text{SDR} \Rightarrow (x, y) = (a, b) \end{cases}$

1 $S^\star = \text{Reg}[\text{Refresh}(s^\star, r_0)]$
2 **if** $gadget = \text{SMx}$ **then** $v = 0$ **else** $v = 1$ *// Public override*
3 **for** $i = 0$ to $n - 1$ **do**
4 $\quad X^{\mathbf{0}}[i] = a^{\mathbf{0}}[i] \oplus b^{\mathbf{0}}[i] \circ S^{\mathbf{0}} \circ \overline{v}$
5 $\quad X^{\mathbf{1}}[i] = a^{\mathbf{1}}[i] \oplus b^{\mathbf{1}}[i] \circ S^{\mathbf{1}} \circ \overline{v}$
6 $\quad Y^{\mathbf{0}}[i] = b^{\mathbf{0}}[i] \circ (S^{\mathbf{1}} \mid v)$
7 $\quad Y^{\mathbf{1}}[i] = b^{\mathbf{1}}[i] \circ (S^{\mathbf{0}} \mid v)$
$\quad \Big\} X^\star[i], Y^\star[i] = H(a^\star[i], b^\star[i], S^\star)$
8 $\quad x^\star[i] = \text{Reg}[\text{Refresh}(X^\star[i], r_{i+1})]$
9 $\quad y^\star[i] = \text{Reg}[\text{Refresh}(Y^\star[i], r_{i+n+1})]$
10 $\quad z^\star[i] = x^\star[i] \oplus y^\star[i]$
11 **end**
12 **return** $x^\star, y^\star, z^\star$

Proposition 5.4. SecMux *is iterated glitch+transition-robust O-PINI.*

Proof. Follows from [CS21, Lemma 2] as SecMux is pipeline and glitch-robust O-PINI. □

5.3.3.3 Composition into secure modular addition

We show in fig. 5.3 the overall execution of the modular addition, based on the above defined gadgets. We call the composite gadget SecAdd$_q$. At the top of the figure are represented iterated executions of a single SecMux structural gadget, initially configured as a dual register (labeled as SDR) that holds the operands and results, and configured as a secure multiplexer (labeled as SMx) at the end of the algorithm, to perform the selection between the raw sum and the offsetted sum, in accordance with line 11 of

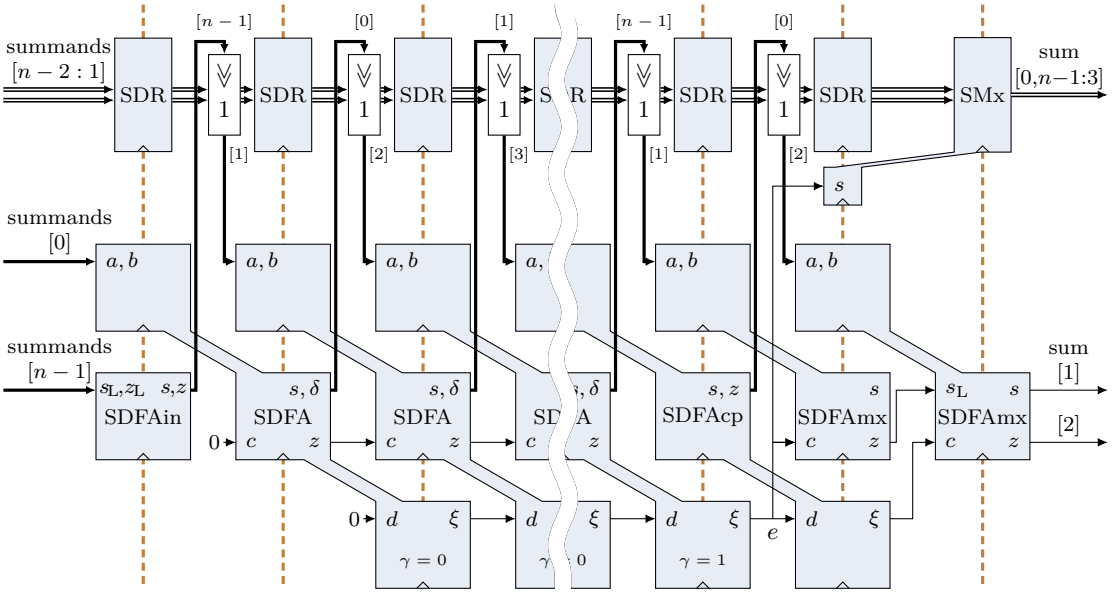


Figure 5.3: Gadget execution of secure modular addition (SecAdd_q). SDR and SMx are two configurations of the same SecMux_{n-2} gadget. Thin wires carry a sharing of a single bit; thick wires, a sharing of two bits; double wires, a sharing of $n-2$ bits. Numbers in brackets indicate the binary weight of the values carried on a wire. All gadgets are O-PINI, white ones being share-isolating. Parts of some gadget executions are not represented due to their output values being discarded.

algorithm 5.3. The iterated executions of SDR are interconnected through a shifter, which implements the operation at lines 6 and 7 of algorithm 5.3: shifting both operands right by one bit, and setting their most significant bit to the sum bits s and δ output by SecDualFullAdder. Since this gadget is obviously share-isolating, it is also robustly O-PINI [CS21, Proposition 1]. The multiplexers that direct the flow of data between gadget executions are not represented: being robustly O-PINI by the same argument, their presence and position have no influence on the security of the composite gadget.

At the bottom, iterated executions of a single SecDualFullAdder structural gadget are shown: in the SDFA configuration, they perform the computation of the sums one bit at a time, in accordance with line 5 of algorithm 5.3. Each execution gets its input carries c and d from the output carries z and ξ of the previous execution, except for their first execution, which gets 0 as input carries.

Due to the two-cycle latency from the a and b inputs of SecDualFullAdder to its s and δ outputs, the shift registers only need to store $n-2$ bits of each operand or result at the middle of the sum execution, which is done with the SecMux_{n-2} gadget. However, this situation is problematic at the beginning of the computation, while the pipeline of SecDualFullAdder is not full, and at its end, when the secure selection implementing modular reduction is performed. Both problems are solved by storing the extra bits inside the flip-flops already present in SecDualFullAdder. This explains the SDFAin

configuration of the gadget at the first cycle, to store the most significant bit of each summand before the sum, and provide them one cycle later at its s and z outputs so they can be assigned to the most significant bit of the dual register. Likewise, toward the end of the addition, the SDFAcp configuration of SecDualFullAdder stores the least significant bit of the sums for two cycles, until the carry output of the offsetted sum is available.

When performing the modular reduction, the two extra bits must not only be stored, but the selection operation done by the secure multiplexer must also be performed on them. To do so, we use two properties of the design. First, the SecDualFullAdder gadget, which is no longer in use for ripple-carry addition at this stage, can be used to store one extra bit of each sum, and to perform the selection between them using logic that is compatible with the carry-computation logic. We denote this configuration by SDFAmx. Second, the two-cycle latency of the secure multiplexer (considered from the selection input to the result output) allows to fit two consecutive executions of SDFAmx in the same timespan, thereby doing the selection for the two missing bits of the sum.

Several aspects of the composite gadget have not been represented: in addition to the already mentioned multiplexers, whose presence is implied by the difference in wiring from cycle to cycle, some inputs and outputs of the gadgets have been omitted when they hold no relevant data. Since all gadgets are robustly O-PINI, how these omitted inputs and outputs are wired has no influence on the security of the composition. Finally, the logic supplying the prime modulus one bit at a time to SecDualFullAdder and governing the configuration of gadgets (choosing, for instance, between the dual-register and the secure-multiplexer functions of SecMux) is not shown: since this logic only processes public parameters, it has no impact on security. This leads us to our main result, theorem 5.1.

Theorem 5.1. *Structural gadget* SecAdd_q *is glitch+transition-robust O-PINI.*

Proof. SecAdd_q is a structural gadget composition (its composing structural gadgets share no structural gates or wires). Since all its composing structural gadgets are iterated glitch+transition-robust O-PINI (by proposition 5.2, proposition 5.4, and the share-isolating characteristic of the other gadgets), the result follows from [CS21, Corollary 1]. □

5.3.4 Other secure summing operations

The SecAdd_q operation described in § 5.3.3.3 can be adapted into secure modular subtraction SecSub_q with small adjustments that can be enabled or disabled at runtime. Comparing eq. (5.3) with eq. (5.1), the following changes can be listed: the computation of the raw result must be performed through subtraction $(a - b)$ instead of addition $(a + b)$; the computation of the offsetted result must add q instead of subtracting it; and, the selection between the raw and offsetted results must depend on the carry output by the raw subtraction, instead of the exclusive-or between the two output carries.

These modifications are implemented in the following way: by complementing the b input of SecDualFullAdder when in SDFA configuration (which is achieved by complementing one share of the value) and inputting a nonzero carry at the beginning of the sum execution, subtraction is computed instead of addition. Then, the last execution of SDFA

is modified so that it copies the complement of its z output into its ξ output, which is achieved by configuring function $G_{m,\gamma=1}$ as $G_{m,1}(T^\star, d^\star, z^\star) = (\overline{z^{\mathbf{0}}}, z^{\mathbf{1}}, 0, 0)$ and keeping $G_{m,\gamma=0}$ unchanged with respect to table 5.2: this implements the modular-reduction condition in accordance with eq. (5.3). Finally, instead of iterating over the bits of $2^n - q$ for the computation of the offsetted sum, the bits of q are used as source for the m parameter.

Furthermore, addition or subtraction can be computed modulo 2^n (these operations are named SecAdd and SecSub respectively) by setting $q = 0$ (i.e., by using parameter $m = 0$ for all executions of SecDualFullAdder).

The latency of power-of-two operations may be reduced by three cycles by stopping the execution of the algorithm as soon as the raw sum is available, and similarly, one cycle of latency can be saved when doing modular subtraction instead of modular addition, since the output carry of the raw sum is available one cycle earlier than that of the offsetted sum. In both cases, the position and ordering of the result bits within the dual shift register and SecDualFullAdder would be changed. We do not describe this solution in detail here.

The configurability among operations SecAdd_q, SecSub_q, SecAdd and SecSub is available at runtime for a very low area overhead: 8.6 % with the above-mentioned timing optimization, 0.4 % without[2]. As soon as the choice of operation is publicly known, this runtime configurability has no security implications since it is achieved with share-isolating gadgets (multiplexers, clearing or complementing of shares) inserted between the previously described O-PINI gadgets. Our design thus includes all four operations natively.

A simplified architectural diagram of our configurable secure adder is show in fig. 5.4 to clarify the interconnections between the gadgets. A controller, that includes a cycle

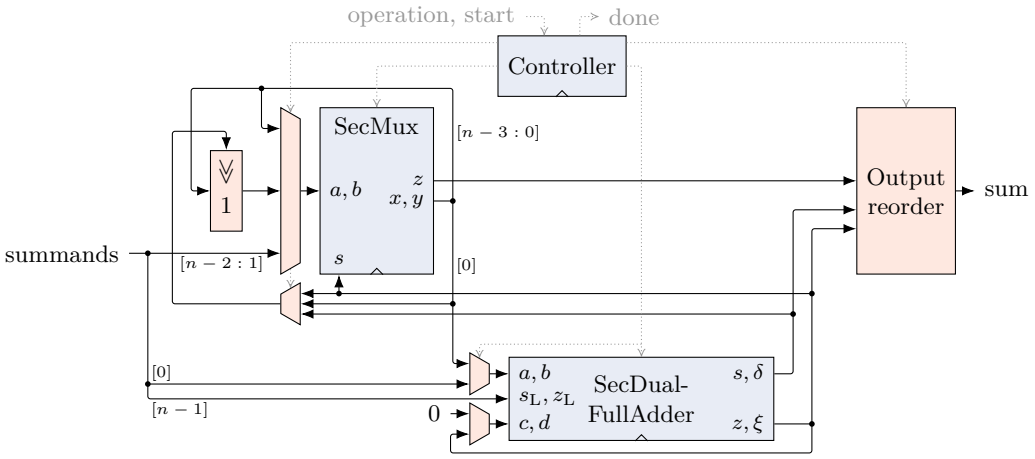


Figure 5.4: Simplified architecture diagram of our secure modular adder over Boolean shares. Light-blue blocks with a bottom notch are sequential components, while orange blocks are combinational only. Numbers in brackets indicate the binary weight of the values carried on a wire. Dashed lines carry control signals. Random bits are not represented.

[2]Total area overhead for an ASIC implementation.

counter and all the logic to generate the control signals (including the enumeration of the bits of the modulus), chooses the configuration of the two gadgets at each cycle. It also drives the multiplexers directing the flow of data between gadgets, and choosing the correct ordering of the result bits for the requested operation. Disabling the timing optimizations mentioned above removes the output-reordering unit, since in this case, all operations have the same latency and the alignment of the result within the shift register is always the same.

5.4 Comparison with previous works

In this section, we compare the performance of our design with the literature. As we are not aware of any previous work that directly implements secure modular addition, we start our comparison in the context of power-of-two addition, and extend it to modular addition by giving the actual performance of our work, and the estimated performance of previous works in this context based on generic conversions from power-of-two to modular addition. We report synthesis results both for an ASIC in a 40 nm technology (the primary target), and for FPGA targets of the Artix-7 family (XC7A100T, speed grade -3) to ease the comparison with previous works[3]. For area comparisons, we mainly focus on the number of flip-flops used. Absent any better way to compare areas between different ASIC and FPGA technologies, we consider this measurement a decent indicator of the overall area[4]. For more accurate comparisons, we also give the ASIC areas in the gate-equivalent (GE) unit, and the LUT counts of our FPGA implementations.

Since our design is meant to be integrated into a larger circuit containing other secure components, we assume a random number generator to be already present, and do not take into account the additional hardware area needed for it.

During synthesis, we took special precautions to prevent the synthesizer from optimizing logic equations in ways that introduce vulnerabilities not present in the register-transfer level description of the design. Practically, we isolated into a separate submodule the logic equation defining each output share of nonlinear gadgets. For the ASIC synthesis, we selectively disabled logic optimizations across these module boundaries, while still allowing other optimizations that do not compromise security. For the FPGA synthesis, having not found such a selective option, we disabled all cross-boundary optimization.

5.4.1 Power-of-two addition

Thanks to the area efficiency of the ripple-carry adder architecture, we obtain a very small hardware design that performs n-bit addition with a latency of $n+1$ clock cycles. We give in table 5.3 and table 5.4 the area utilization and latency we obtain for ASIC and FPGA implementations respectively, compared with previous works. Our figures are for 32-bit addition to match the literature; however, our work is better suited to the smaller integer widths encountered in lattice-based cryptography, e.g. 12 or 23 bits [SAB+20, LDK+20].

[3] Spartan-6 and Artix-7 FPGAs have similar lookup tables: LUT counts should be comparable.

[4] The flip-flops take up 40 % of the overall area of our ASIC implementations.

Table 5.3: ASIC performance of 1st-order–secure 32-bit addition over Boolean shares

Design	Technology	Flip-flops	Area kGE	Latency cycles	Random bit/cyc.	Freq. MHz	Notes
Ripple carry adder							
Ours	40 nm CMOS	146	2.05	33	69	400	Mod-q add.: 36 cycles
Ours (no opt)	40 nm CMOS	146	1.90	36	69	400	No timing optimization
[CGM+23]	Nangate 45	3100**	19.23	31*	32		Fully pipelined
Fully pipelined carry-lookahead adders							
[CGM+23]	Nangate 45		18.30	5*	374		Kogge-Stone
[CGM+23]	Nangate 45		13.77	6*	172		Sklansky
[CGM+23]	Nangate 45		12.07	9*	115		Brent-Kung

* Throughput of one addition per cycle. ** Figure absent from paper, estimated from the description.

When it includes the timing optimizations for subtraction and power-of-two operations, mentioned in section 5.3.4, the ASIC design takes up 2.05 kGE and executes power-of-two operations in 33 cycles, and modular subtraction and addition in 35 and 36 cycles respectively. Without these optimizations, all operations have a latency of 36 cycles, but the design only occupies 1.90 kGE due to having less combinational logic.

The above ASIC synthesis results are reported for a target frequency of 400 MHz in the worst PVT corner; however, adjusting the synthesis constraints allows to reach up to 660 MHz at the cost of increasing the area by 50 % (or by 40 % for the design without timing optimizations), still in the worst corner.

As expected, our ripple-carry adder has six to thirty times smaller area than previous works based on pipelined parallel-prefix adders [SMG15, FBR+22, BG22, CGM+23], at the expected cost of much higher latency. We also reduce the number of flip-flops by one third with respect to the ripple-carry adder of Schneider *et al.* [SMG15], thanks to the use of two shares per secret value instead of three for their threshold implementation. This lower number of shares still achieves the same security order, and additionally benefits from a proof of robustness against transitions and glitches, which is not explicitly the case for [SMG15]. Indeed, threshold implementations are not robustly composable, and while Schneider *et al.* discuss how they avoid transition-based leakage in a specific part of their design, they do not provide a full robustness analysis[5]. With and without timing optimization, our work uses 1.95 and 1.67 times as many LUTs as that of [SMG15], but it is difficult to know how much additional logic it represents since they do not synthesize for ASIC. Anyhow, this extra logic area is a low price to pay for the additional modular reduction.

Since the other designs from the state of the art are fully pipelined, they can perform one addition per clock cycle in steady operation, compared to one addition per 33 or 32

[5]Replacing the TI gadgets used by Schneider *et al.* with glitch-robust O-PINI gadgets to get provable robustness against glitches and transitions is not directly possible since glitch-robust O-PINI gadgets from the literature [CS21, KM22] have two or three cycles of latency, while the construction of [SMG15] requires the gadgets in the carry-computation path to have a single cycle of latency.

Table 5.4: FPGA performance of 1st-order–secure 32-bit addition over Boolean shares

Design	Family	Area		Latency	Random	Freq.	Notes
		Flip-flops	LUTs	cycles	bit/cyc.	MHz	
Ripple carry adder							
Ours	Artix-7	146	441	33	69	200	Mod-q addition: 36 cycles
Ours (no opt)	Artix-7	146	380	36	69	250	No timing optimization
[SMG15]	Spartan-6	223	227	32	4	101	Refreshes only at first cycle
Fully pipelined Kogge-Stone carry-lookahead adder							
[SMG15]	Spartan-6	1330	937	6*	31	62	
[FBR+22]	Artix-7	1323	2464	6*		454	Contains additional logic
TI [BG22]	Spartan-6	1416	873	6*	32	228	
HPC2 [BG22]	Spartan-6	3981	2936	12*	249	176	
Fully pipelined Sklansky carry-lookahead adder							
TI [BG22]	Spartan-6	1416	579	6*	41	174	
HPC2 [BG22]	Spartan-6	3166	1801	12*	119	153	
Fully pipelined Brent-Kung carry-lookahead adder							
TI [BG22]	Spartan-6	2352	487	9*	31	280	
HPC2 [BG22]	Spartan-6	4317	1588	18*	74	173	

* Throughput of one addition per cycle.

clock cycles for our iterative ripple-carry adder and that of Schneider *et al.*. However, such architectures are only possible for the highest-performance applications considering the very large area occupied by fully pipelined adders. Our work, instead, definitely achieves its primary goal of low area utilization.

In terms of randomness usage, our proposed construction requires 8 fresh random bits per cycle for SecDualFullAdder and $2n-3$ for SecMux_{n-2}, that is, $2n+5$ bits per cycle in total. At 69 bits per cycle, the randomness requirements of our construction are consistent with the other listed designs, that range from 4 to 374 bits per cycle[6]. These relatively high randomness requirements are due to using an iterative design, since it mandates using O-PINI gadgets and sizing the amount of refresh bits based on the most costly iteration.[7]

5.4.2 Modular addition

As its primary objective, our architecture allows for performing secure modular addition at nearly no extra cost with respect to power-of-two addition: the necessary area is

[6]While our solution has a high cumulative randomness consumption per addition (2277 bits per 32-bit addition), we consider this figure of little importance: without expensive buffering, the random number generator must provide the required per-cycle randomness, and cumulative randomness is irrelevant.

[7]Analysing the area required for randomness generation is beyond our scope; however, extrapolating the research of Cassiers *et al.* [CMM+24] suggests that around 1.6 kGE would be required for a 138-bit linear-feedback shift register providing 69 bits per cycle, or around 4 kGE for an unrolled Trivium cipher.

already included, and the latency is increased by three cycles only. This is in contrast to previous works, which do not directly support modular addition. They must instead rely on two parallel or consecutive power-of-two additions, which uses either double the area (and double the per-cycle randomness), or double the latency with half the throughput. Consequently, in this setting, our construction achieves an even closer cost/performance ratio to high-performance designs from the literature: our design uses 5256 flip-flop × cycles per modular addition, and the efficient construction of [FBR+22] would spend at least 2646 flip-flop × cycles (in steady state) for the same task. Our work thus compares unexpectedly well with high-performance designs considering its focus on area minimization.

A similar comparison can be made with the ripple-carry adder of Schneider *et al.* [SMG15]. In this case, we will study two constructions for modular addition: either the raw and offsetted sums are computed in parallel, and secure selection between these two results accomplishes the modular reduction (section 5.3.1); or, using the solution of [FBR+22], the offsetted sum is computed directly, and this result is conditionally added with the modulus. The first solution will require the side-by-side instantiation of two adders and a secure n-bit multiplexer, and add at least one cycle of latency to the whole operation. The second one, instead, may reuse the same secure adder for both additions, but it will double the latency and halve the throughput with respect to power-of-two addition. Both solutions will thus use more than 14 272 flip-flop × cycles per 32-bit modular addition, which represents nearly three times the overall cost of our proposal.

5.5 Leakage assessment

We have proved in § 5.3.3.3 the security of our construction in a strong security model that is robust against glitches and transitions. However, since hardware implementations may have additional defects overlooked by this model, we give further assurance in the security of our design by performing a leakage assessment in simulation. Since this gate-level simulation is noiseless and it models the propagation delays within the cells for each input condition (not the routing delays, as simulation is before place-and-route), we expect better fidelity with this approach than by porting the design to an FPGA for validation.

We thus synthesize the secure adder for $n = 12$ bits with prime modulus $q = 3329$, for a 40 nm CMOS technology, and annotate it with gate-timing information. We then simulate the obtained netlist and derive power-consumption traces from the toggle count of the circuit, that is, the number of nets that change logic value at each time sample. This so-called *toggle-count* metric, introduced by Sadhukhan *et al.* [SMRM19], is suitable for the leakage assessment of a pre-silicon design. The simulation assumes a clock frequency of 200 MHz, easily achieved by the synthesized design, and uses a time resolution of 1 ps.

We analyze the simulated power traces in the test-vector leakage assessment (TVLA) methodology [GJJR11], which consists in collecting two separate sets of traces for different scenarios, and performing a t-test between the two sets to check whether they are statistically distinguishable. In all our experiments, the first set of traces corresponds to summing two all-zero operands, each masked with a uniformly random Boolean mask;

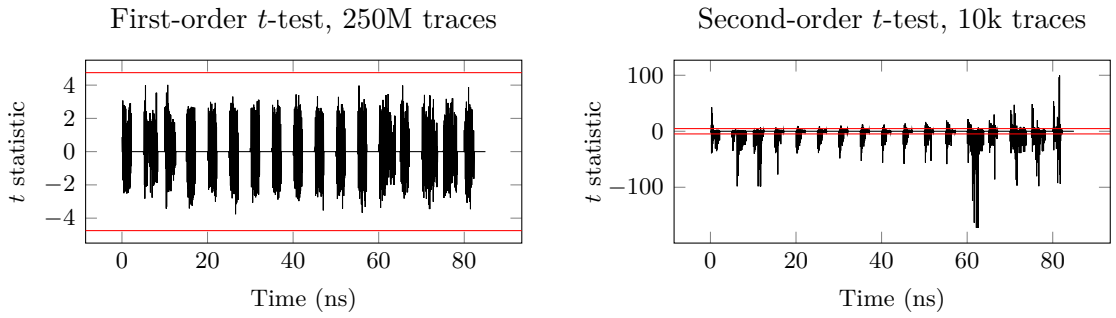


Figure 5.5: Leakage-assessment results of secure modular addition with first-order and second-order t-test at 200 MHz with $n = 12$. Modulus is $q = 3329$. A fixed-vs-random scenario is studied: for the first set of traces, both summands are always zero; for the second set, both summands are uniformly and independently sampled from $[\![0, q-1]\!]$. As expected for a secure first-order design, the first-order t-test shows no leakage ($|t| \leqslant 4.02 < 4.75$) with 250 million traces, while the second-order one already exhibits strong leakage ($\max|t| \gg 4.75$) at ten thousand traces. The HC statistic is respectively $1.6 < 4.8$ and ∞.

the second set of traces corresponds to sampling the two summands independently and uniformly at random from $[\![0, q-1]\!]$, again masking each of them with a random Boolean mask.

The results of the TVLA are reproduced in fig. 5.5. On the left are shown the t-test results with sets of 250 million traces each, at the first order. Since the power trace contains a large number of samples (85 000, of which 25 000 are nonzero), we choose a threshold of 4.75 for the t statistic, which corresponds to a false-positive probability of 5 % [DZD+18]. As expected from a secure design, the t statistic does not cross the ± 4.75 threshold anywhere in the trace (its maximum absolute value is 4.02), showing no statistically significant difference in the average power consumption between the two sets of traces.

To make the leakage detection more sensitive, we follow the more advanced methodology of [DZD+18] and compute the Higher Criticism (HC) statistic. Instead of only considering the extreme values of the t-test, this methodology checks the distribution of all t values against the joint null hypothesis of having no leakage at any trace point[8]. This statistic is equal to 1.6 in the first-order t-test, which is well below the 4.8 threshold for a significance level of 5 %. Furthermore, our choice of a false-positive probability of 5 % is more conservative (*i.e.* more sensitive to leakage) than the 1 % chosen by [DZD+18], which would result in threshold values of 5.1 for the t statistic and 10.1 for the HC statistic.

On the right of fig. 5.5, a second-order TVLA is conducted: as anticipated, strong leakage is shown since our design is only secure at the first order. In this situation, the small number of ten thousand traces per set is amply sufficient to detect the leakage, with t values exhibiting several strong peaks whose amplitude exceeds ± 100, well beyond

[8]The HC statistic implicitly assumes that all points of the trace have independently distributed noise; however, it is not significantly affected by local correlation in the noise [DZD+18].

the threshold for leakage detection. This is again confirmed with absolute certainty by the HC statistic, which is too large to even be represented as a floating-point value.

We run two additional experiments in fig. 5.6, by totally or selectively deactivating the random bits for refreshes. Both experiments involve ten thousand traces per set. In the left graph, we show the effect of deactivating all refreshes in the masked circuit, with the expected outcome of introducing strong leakage, since the security of the nonlinear masked operations in this chapter entirely depends on proper refreshing of the shares. In the rightmost graph, instead, we only deactivate part of the refreshes: specifically, the $2n-3$ refresh bits used by the SecMux gadget, while the refreshes inside SecDualFullAdder are kept. This partial disabling still introduces strong leakage, but this time, restricted to the last two clock cycles of secure modular addition. These are the cycles in which modular reduction is performed by selecting between the raw and offsetted sums. Since it is the only time at which a refreshing is required for security within SecMux, the leakage only occurs at these two cycles[9]. If the design were restricted to power-of-two operations, the randomness requirements would thus drop to 8 fresh bits per cycle without loss of security.

Overall, this leakage assessment helps confirm that the glitch+transition-robust probing model does not overlook glaring hardware defects, and that the synthesis flow does not introduce obvious vulnerabilities not present in the register-transfer level design.

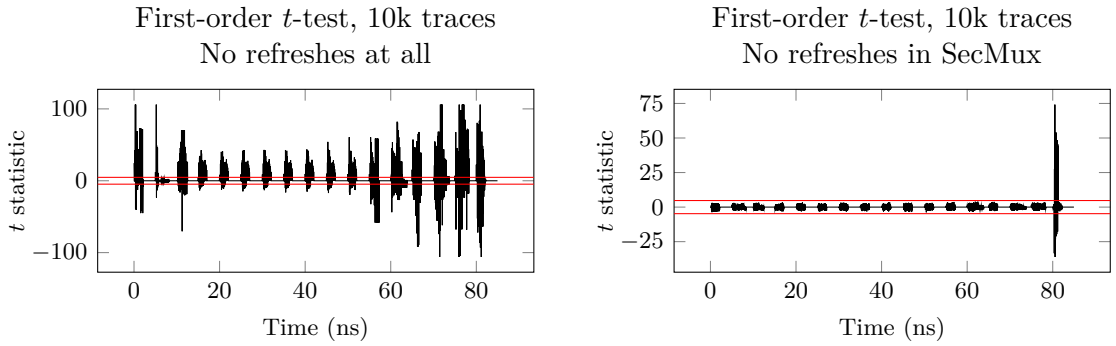


Figure 5.6: Leakage-assessment results of secure modular addition at the first order, with refreshes totally or partially disabled. Zero-vs-random scenario with ten thousand traces per set. With all refreshing disabled, leakage occurs at every clock cycle (left), while solely disabling the refreshing of SecMux contains the leakage to the last cycle (right).

[9]Since the secure dual register is also implemented by the SecMux gadget in its SDR configuration, refreshing also occurs in this other configuration, but is not required for security. Indeed, if its SMx configuration is removed, SecMux becomes share-isolating, so it remains O-PINI even without refreshes.

5.6 Conclusions

5.6.1 Summary

In this chapter, we have presented a new construction to compute modular addition securely over Boolean shares, with proven security against first-order probing attacks in the robust-probing model. To the best of our knowledge, this is both the first secure adder natively supporting modular arithmetic, and the first iterative adder benefiting from a proof of robust security in the presence of glitches and transitions. We furthermore demonstrated that these security claims firmly hold when performing leakage assessment in simulation.

Through careful design, our construction reaches an area efficiency that is significantly beyond the state of the art without compromising on security, and with better flexibility than any of the previous works from the literature, as it natively supports runtime configurability among either addition or subtraction, and either reduction modulo a publicly-known prime, or simple wraparound modulo a power of two.

Secure modular addition and subtraction, while being costly operations, are crucial to the protection of recent lattice-based cryptography algorithms. We expect that our work will help implement these algorithms securely on resource-constrained embedded devices.

5.6.2 Open problems

This study exclusively focuses on first-order security, which was deemed sufficient in the context of a low-cost implementation. An important extension of this work would consist in determining how the architecture of the solution would change when generalizing it to higher security orders. Indeed, our performance constraints forced us to design custom integrated gadgets, whose applicability to higher-order masking is unclear. The main step toward this goal would be to generalize SecDualFullAdder to high-order masking, while still containing to a single cycle the latency of its carry-chain paths $c \rightarrow z$ and $d \rightarrow \xi$.

We have shown in section 5.4 that, while our solution provides less throughput per unit of area than carry-lookahead adders from the literature, this difference remains extremely reasonable given the very large starting area of these high-performance adders. It would thus be interesting to explore whether intermediate approaches could lead to better efficiency than either architecture in moderate-performance applications. Besides, a strong assumption that was made in our work is that fresh random bits have low cost due to an already present pseudorandom number generator. For contexts in which this assumption does not hold, it would be highly beneficial to reduce the randomness requirements of our individual gadgets, or to reuse randomness between gadgets.

Finally, our main concern in this chapter was to reduce the area consumption of secure modular addition; yet, an equally important concern for low-cost implementations is their power consumption. Considering to the shift-register structure of our construction, about half of the flip-flops are expected to toggle at each clock cycle, which may cause a high dynamic power consumption compared to the small size of the circuit. Whether this consumption can be tolerated will be highly application-dependent.

5.A Security proofs

5.A.1 Glitch-robust O-PINI security of SecDualFullAdder

Proof of proposition 5.1. Table 5.5 lists the glitch-extended probes on output shares of SecDualFullAdder. Now, consider any of the internal probes listed in the first column of table 5.6. Without loss of generality, we will consider the probe in the last row: $\Xi'^{\mathbf{2}}$, that is, the result of refreshing $\Xi^{\mathbf{2}}$ with r_6. This probe is of particular interest as its extension involves cross-domain terms, which are the limiting factor to the security of nonlinear gadgets.

The probe glitch-extends to probes on $T'^{\mathbf{1}}$, $d^{\mathbf{0}}$, and r_6. We now build a simulator for the extended probe on $\Xi'^{\mathbf{2}}$ as well as extended probes on outputs having share index 0, from the knowledge of input shares with index 0: $a^{\mathbf{0}}$, $b^{\mathbf{0}}$, $c^{\mathbf{0}}$, $d^{\mathbf{0}}$, $s_{\mathrm{L}}^{\mathbf{0}}$, $z_{\mathrm{L}}^{\mathbf{0}}$.

The simulator first samples at random all the values listed in the second column, namely, $T'^{\mathbf{1}}$, r_6, $S'^{\mathbf{0}}$, $Z'^{\mathbf{0}}$, $Z'^{\mathbf{2}}$, $Z'^{\mathbf{4}}$ and $\Xi'^{\mathbf{0}}$. This sampling is indistinguishable from the actual gadget execution since each of these values is blinded with (or is a) fresh

Table 5.5: Glitch extension of probes on SecDualFullAdder outputs. Prime symbols represent the refreshed value of the corresponding quantity, e.g. $\Xi'^{\mathbf{2}} = \Xi^{\mathbf{2}} \oplus r_6$.

Share index	Glitch-extended probes on output shares			
0	$s^{\mathbf{0}} = S'^{\mathbf{0}}$,	$z^{\mathbf{0}} = Z'^{\mathbf{0}} \oplus Z'^{\mathbf{2}} \oplus Z'^{\mathbf{4}}$,	$\delta^{\mathbf{0}} = d^{\mathbf{0}} \oplus m$,	$\xi^{\mathbf{0}} = \Xi'^{\mathbf{0}} \oplus \Xi'^{\mathbf{2}}$
1	$s^{\mathbf{1}} = S'^{\mathbf{1}}$,	$z^{\mathbf{1}} = Z'^{\mathbf{1}} \oplus Z'^{\mathbf{3}} \oplus Z'^{\mathbf{5}}$,	$\delta^{\mathbf{1}} = d^{\mathbf{1}}$,	$\xi^{\mathbf{1}} = \Xi'^{\mathbf{1}} \oplus \Xi'^{\mathbf{3}}$

Table 5.6: Simulation of a glitch-extended internal probe of even index in SecDualFullAdder together with all extended probes on output shares having index 0. All input shares with index 0 ($a^{\mathbf{0}}$, $b^{\mathbf{0}}$, $c^{\mathbf{0}}$, $d^{\mathbf{0}}$, $s_{\mathrm{L}}^{\mathbf{0}}$, $z_{\mathrm{L}}^{\mathbf{0}}$) are known. Prime symbols represent the refreshed value of the corresponding quantity, e.g. $\Xi'^{\mathbf{2}} = \Xi^{\mathbf{2}} \oplus r_6$. Public parameters m and γ are known.

Glitch-extended internal probes	Values simulated by random sampling	Notes
$A^{\mathbf{0}} = a^{\mathbf{0}} \oplus r_0, B^{\mathbf{0}} = b^{\mathbf{0}} \oplus r_1$ $T'^{\mathbf{0}} = T^{\mathbf{0}} = A^{\mathbf{0}} \oplus B^{\mathbf{0}} \oplus c^{\mathbf{0}}$ $S'^{\mathbf{0}} = E^0(A^{\mathbf{0}}, T^{\mathbf{0}}, s_{\mathrm{L}}^{\mathbf{0}}) \oplus r_7$	r_0, r_1, r_7 $Z'^{\mathbf{0}}, Z'^{\mathbf{2}}, Z'^{\mathbf{4}}, \Xi'^{\mathbf{0}}, \Xi'^{\mathbf{2}}$	
$Z'^{\mathbf{0}} = F^0(A^{\mathbf{0}}, B^{\mathbf{0}}, c^{\mathbf{0}}, z_{\mathrm{L}}^{\mathbf{0}}) \oplus r_2$	r_0, r_1, r_2 $S'^{\mathbf{0}}, Z'^{\mathbf{2}}, Z'^{\mathbf{4}}, \Xi'^{\mathbf{0}}, \Xi'^{\mathbf{2}}$	Compute $A^{\mathbf{0}} = a^{\mathbf{0}} \oplus r_0$, $B^{\mathbf{0}} = b^{\mathbf{0}} \oplus r_1$
$Z'^{\mathbf{2}} = F^2(A^{\mathbf{1}}, B^{\mathbf{0}}, c^{\mathbf{0}}) \oplus r_3$	$A^{\mathbf{1}}, r_1, r_3$ $S'^{\mathbf{0}}, Z'^{\mathbf{0}}, Z'^{\mathbf{4}}, \Xi'^{\mathbf{0}}, \Xi'^{\mathbf{2}}$	Compute $B^{\mathbf{0}} = b^{\mathbf{0}} \oplus r_1$ $A^{\mathbf{1}}$ is blinded with r_0 (not probed)
$Z'^{\mathbf{4}} = F^4(B^{\mathbf{1}}, c^{\mathbf{0}}) \oplus r_4$	$B^{\mathbf{1}}, r_4, S'^{\mathbf{0}}, Z'^{\mathbf{0}}, Z'^{\mathbf{2}}, \Xi'^{\mathbf{0}}, \Xi'^{\mathbf{2}}$	$B^{\mathbf{1}}$ is blinded with r_1 (not probed)
$\Xi'^{\mathbf{0}} = G^0_{m,\gamma}(T'^{\mathbf{0}}, d^{\mathbf{0}}) \oplus r_5$	$T'^{\mathbf{0}}, r_5, S'^{\mathbf{0}}, Z'^{\mathbf{0}}, Z'^{\mathbf{2}}, Z'^{\mathbf{4}}, \Xi'^{\mathbf{2}}$	$T'^{\mathbf{0}}$ is blinded with r_0 and r_1 (not probed)
$\Xi'^{\mathbf{2}} = G^2_{m,\gamma}(T'^{\mathbf{1}}, d^{\mathbf{0}}) \oplus r_6$	$T'^{\mathbf{1}}, r_6, S'^{\mathbf{0}}, Z'^{\mathbf{0}}, Z'^{\mathbf{2}}, Z'^{\mathbf{4}}, \Xi'^{\mathbf{0}}$	$T'^{\mathbf{1}}$ is blinded with r_0 and r_1 (not probed)

random bits, respectively $r_0 \oplus r_1$, r_6, r_7, r_2, r_3, r_4, r_5. Then, $\Xi'^{\mathbf{2}}$ can be computed from these values and $d^{\mathbf{0}}$, which is a simulator input. In turn, it is clear from table 5.5 that these values are sufficient to compute the glitch-extended probes on outputs having share index 0.

The other lines of table 5.6 likewise indicate the values which must be sampled to simulate any other internal probe having even index. Similarly, any internal probe having odd index, together with extended probes on outputs with index 1, can be simulated from the input shares having index 1: the equivalent of table 5.6 for odd-index probes is derived by toggling the parity of the index of all quantities in the table except for random bits. □

5.A.2 Glitch-robust O-PINI security of SecMux

Proof of proposition 5.3. We highlight that the two configurations of the SecMux gadget use the same Boolean operators internally, only with different operands: where SMx operates on the shares of s, SDR instead uses public constants 0 and 1. Thus, they can be implemented using the same set of structural gates, with a public parameter to override the secret inputs with constants when implementing SDR. Proving the security of the gadget in the SMx mode is thus sufficient.

Let us consider an internal probe on the input of register $y^{\mathbf{0}}[j]$ for some arbitrary (but fixed) $j \in [\![0, n-1]\!]$. As before, this probe is of particular interest because it involves a crossing between share domains. We place the probe at the input of the corresponding register barrier, so that it glitch-extends through combinational logic according to the first cell in the last row of table 5.8. We can then simulate this extended probe, as well as extended output probes on share 0 of each output (their extension is shown in table 5.7), from the knowledge of inputs with share index 0, by sampling at random the values listed in the second cell of the same row: $S^{\mathbf{1}}$, r_{j+n+1} for the chosen j, all $x^{\mathbf{0}}[i]$ for $i \in [\![0, n-1]\!]$, and all $y^{\mathbf{0}}[i]$ for $i \in [\![0, n-1]\!] \setminus \{j\}$. Sampling all of these quantities independently at random makes the simulation indistinguishable from the actual gadget execution, since in the latter case each value is blinded with fresh random bits. Finally, the simulator can compute $y^{\mathbf{0}}[j]$ from its actual expression in algorithm 5.5 since all terms of the expression have been simulated. Likewise, outputs $(z^{\mathbf{0}}[i])_{0 \leqslant i < n}$ can be computed as $z^{\mathbf{0}} = x^{\mathbf{0}} \oplus y^{\mathbf{0}}$.

The proof proceeds likewise for any other internal probe having even index, and is easy to adapt to probes having odd index by toggling the parity of all share indexes. □

Table 5.7: Glitch extension of probes on output shares of SecMux_n.

Share index	Glitch-extended probes on output shares
0	$x^{\mathbf{0}}[i]$, $y^{\mathbf{0}}[i]$, $z^{\mathbf{0}}[i] = x^{\mathbf{0}}[i] \oplus y^{\mathbf{0}}[i]$
1	$x^{\mathbf{1}}[i]$, $y^{\mathbf{1}}[i]$, $z^{\mathbf{1}}[i] = x^{\mathbf{1}}[i] \oplus y^{\mathbf{1}}[i]$

Table 5.8: Simulation of a glitch-extended internal probe of even index in SecMux_n together with all extended probes on index-0 output shares. Input shares with index 0 ($(a^{\mathbf{0}}[i])_{0\leqslant i<n}$, $(b^{\mathbf{0}}[i])_{0\leqslant i<n}$, $c^{\mathbf{0}}$, $s^{\mathbf{0}}$) are known. The extended probes listed in the first column are placed at the input of the corresponding register barrier, and glitch-extend back to the inputs or to the previous register barrier.

Glitch-extended internal probes	Values simulated by random sampling	Notes
$S^{\mathbf{0}} = s^{\mathbf{0}} \oplus r_0$	$r_0, (x^{\mathbf{0}}[i])_{0\leqslant i<n}$, $(y^{\mathbf{0}}[i])_{0\leqslant i<n}$	Each $x^{\mathbf{0}}[i]$, $y^{\mathbf{0}}[i]$ is blinded with fresh randomness
$x^{\mathbf{0}}[j] = a^{\mathbf{0}}[j] \oplus b^{\mathbf{0}}[j]S^{\mathbf{0}} \oplus r_{j+1}$ (j is fixed)	$r_0, r_{j+1}, (x^{\mathbf{0}}[i])_{i\neq j}$, $(y^{\mathbf{0}}[i])_{0\leqslant i<n}$	Compute $S^{\mathbf{0}} = s^{\mathbf{0}} \oplus r_0$, $x^{\mathbf{0}}[j] = a^{\mathbf{0}}[j] \oplus b^{\mathbf{0}}[j] \circ S^{\mathbf{0}} \oplus r_{j+1}$; each other $x^{\mathbf{0}}[i]$, $y^{\mathbf{0}}[i]$ is blinded with fresh randomness
$y^{\mathbf{0}}[j] = b^{\mathbf{0}}[j]S^{\mathbf{1}} \oplus r_{j+n+1}$ (j is fixed)	$S^{\mathbf{1}}, r_{j+n+1}$, $(x^{\mathbf{0}}[i])_{0\leqslant i<n}$, $(y^{\mathbf{0}}[i])_{i\neq j}$	Compute $y^{\mathbf{0}}[j] = b^{\mathbf{0}}[j]S^{\mathbf{1}} \oplus r_{j+3}$; each other $x^{\mathbf{0}}[i]$, $y^{\mathbf{0}}[i]$ is blinded with fresh randomness; $S^{\mathbf{1}}$ is blinded with r_0, which is not probed

5.B Explicit definition of SecDualFullAdder configurations

Algorithms 5.6 to 5.9 below show the computations performed by the four configurations of SecDualFullAdder, without relying on black-box functions. These algorithms are obtained by specializing algorithm 5.4 using the equations from table 5.2.

Algorithm 5.6: SDFA function of SecDualFullAdder

Input: Shares $a^\star, b^\star, c^\star, d^\star \in \mathcal{B}^2(\mathbb{F}_2)$, public param. $m, \gamma \in \mathbb{F}_2$, randomness $r_0, \ldots, r_7 \in \mathbb{F}_2$
Output: Shares $s^\star, z^\star, \delta^\star, \xi^\star \in \mathcal{B}^2(\mathbb{F}_2)$ such that $(z \ll 1) \mid s = a + b + c$ and
$(\xi \ll 1) \mid (\delta \oplus s) = ((a \oplus b \oplus c) + d + m) \oplus (\gamma z \ll 1)$

$A^\star = \mathrm{Reg}[\mathrm{Refresh}(a^\star, r_0)] \qquad B^\star = \mathrm{Reg}[\mathrm{Refresh}(b^\star, r_1)]$
$S^\star = A^\star \oplus B^\star \oplus c^\star$
$Z^{\mathbf{0}}, Z^{\mathbf{1}} = A^{\mathbf{0}}B^{\mathbf{0}} \oplus A^{\mathbf{0}}c^{\mathbf{0}} \oplus B^{\mathbf{0}}c^{\mathbf{0}}, A^{\mathbf{1}}B^{\mathbf{1}} \oplus A^{\mathbf{1}}c^{\mathbf{1}} \oplus B^{\mathbf{1}}c^{\mathbf{1}}$
$Z^{\mathbf{2}}, Z^{\mathbf{3}} = A^{\mathbf{1}} \circ (B^{\mathbf{0}} \oplus c^{\mathbf{0}}), A^{\mathbf{0}} \circ (B^{\mathbf{1}} \oplus c^{\mathbf{1}})$
$Z^{\mathbf{4}}, Z^{\mathbf{5}} = B^{\mathbf{1}}c^{\mathbf{0}}, B^{\mathbf{0}}c^{\mathbf{1}}$
$T^\star = \mathrm{Reg}[A^\star \oplus B^\star \oplus c^\star]$
$s^\star = \mathrm{Reg}[\mathrm{Refresh}(S^\star, r_7)]$
$z^\star = \mathrm{Reg}[\mathrm{Refresh}((Z^{\mathbf{0}}, Z^{\mathbf{1}}), r_2)] \oplus \mathrm{Reg}[\mathrm{Refresh}((Z^{\mathbf{2}}, Z^{\mathbf{3}}), r_3)] \oplus \mathrm{Reg}[\mathrm{Refresh}((Z^{\mathbf{4}}, Z^{\mathbf{5}}), r_4)]$
$\delta^\star = d^\star \oplus (m, 0)$
$\Xi^{\mathbf{0}}, \Xi^{\mathbf{1}} = (T^{\mathbf{0}} \oplus d^{\mathbf{0}}) \circ m \oplus T^{\mathbf{0}}d^{\mathbf{0}} \oplus \gamma z^{\mathbf{0}}, (T^{\mathbf{1}} \oplus d^{\mathbf{1}}) \circ m \oplus T^{\mathbf{1}}d^{\mathbf{1}} \oplus \gamma z^{\mathbf{1}}$
$\Xi^{\mathbf{2}}, \Xi^{\mathbf{3}} = T^{\mathbf{1}}d^{\mathbf{0}}, T^{\mathbf{0}}d^{\mathbf{1}}$
$\xi^\star = \mathrm{Reg}[\mathrm{Refresh}((\Xi^{\mathbf{0}}, \Xi^{\mathbf{1}}), r_5)] \oplus \mathrm{Reg}[\mathrm{Refresh}((\Xi^{\mathbf{2}}, \Xi^{\mathbf{3}}), r_6)]$
return $s^\star, z^\star, \delta^\star, \xi^\star$

Algorithm 5.7: SDFAmx function of SecDualFullAdder

Input: Shares $a^\star, b^\star, c^\star, s_{\mathrm{L}}^\star \in \mathcal{B}^2(\mathbb{F}_2)$, randomness $r_0, \ldots, r_7 \in \mathbb{F}_2$
Output: Shares $s^\star, z^\star \in \mathcal{B}^2(\mathbb{F}_2)$ such that $z = a \oplus bs$ and $s = s_{\mathrm{L}}$

1 $A^\star = \mathrm{Reg}[\mathrm{Refresh}(a^\star, r_0)] \qquad B^\star = \mathrm{Reg}[\mathrm{Refresh}(b^\star, r_1)]$
2 $S^\star = s_{\mathrm{L}}$
3 $Z^{\mathbf{0}}, Z^{\mathbf{1}} = A^{\mathbf{0}}\overline{c^{\mathbf{0}}} \oplus B^{\mathbf{0}}c^{\mathbf{0}}, A^{\mathbf{1}}c^{\mathbf{1}} \oplus B^{\mathbf{1}}c^{\mathbf{1}}$
4 $Z^{\mathbf{2}}, Z^{\mathbf{3}} = A^{\mathbf{1}}\overline{c^{\mathbf{0}}}, A^{\mathbf{0}}c^{\mathbf{1}}$
5 $Z^{\mathbf{4}}, Z^{\mathbf{5}} = B^{\mathbf{1}}c^{\mathbf{0}}, B^{\mathbf{0}}c^{\mathbf{1}}$
6 $s^\star = \mathrm{Reg}[\mathrm{Refresh}(S^\star, r_7)]$
7 $z^\star = \mathrm{Reg}[\mathrm{Refresh}((Z^{\mathbf{0}}, Z^{\mathbf{1}}), r_2)] \oplus \mathrm{Reg}[\mathrm{Refresh}((Z^{\mathbf{2}}, Z^{\mathbf{3}}), r_3)] \oplus \mathrm{Reg}[\mathrm{Refresh}((Z^{\mathbf{4}}, Z^{\mathbf{5}}), r_4)]$
8 **return** $s^\star, z^\star$

Algorithm 5.8: SDFAcp function of SecDualFullAdder

Input: Shares $a^\star, b^\star, c^\star, d^\star \in \mathcal{B}^2(\mathbb{F}_2)$, randomness $r_0, \ldots, r_7 \in \mathbb{F}_2$
Output: Shares $s^\star, z^\star, \xi^\star \in \mathcal{B}^2(\mathbb{F}_2)$ such that $s = a$, $z = b$, $\xi = d$

1 $A^\star = \mathrm{Reg}[\mathrm{Refresh}(a^\star, r_0)] \qquad B^\star = \mathrm{Reg}[\mathrm{Refresh}(b^\star, r_1)]$
2 $S^\star = A^\star$
3 $Z^{\mathbf{0}}, \ldots, Z^{\mathbf{5}} = B^{\mathbf{0}}, B^{\mathbf{1}}, 0, \ldots, 0$
4 $s^\star = \mathrm{Reg}[\mathrm{Refresh}(S^\star, r_7)]$
5 $z^\star = \mathrm{Reg}[\mathrm{Refresh}((Z^{\mathbf{0}}, Z^{\mathbf{1}}), r_2)] \oplus \mathrm{Reg}[\mathrm{Refresh}((Z^{\mathbf{2}}, Z^{\mathbf{3}}), r_3)] \oplus \mathrm{Reg}[\mathrm{Refresh}((Z^{\mathbf{4}}, Z^{\mathbf{5}}), r_4)]$
6 $\Xi^{\mathbf{0}}, \ldots, \Xi^{\mathbf{3}} = d^{\mathbf{0}}, d^{\mathbf{1}}, 0, 0$
7 $\xi^\star = \mathrm{Reg}[\mathrm{Refresh}((\Xi^{\mathbf{0}}, \Xi^{\mathbf{1}}), r_5)] \oplus \mathrm{Reg}[\mathrm{Refresh}((\Xi^{\mathbf{2}}, \Xi^{\mathbf{3}}), r_6)]$
8 **return** $s^\star, z^\star, \xi^\star$

Algorithm 5.9: SDFAin function of SecDualFullAdder

Input: Shares $s_{\mathrm{L}}^\star, z_{\mathrm{L}}^\star \in \mathcal{B}^2(\mathbb{F}_2)$, randomness $r_0, \ldots, r_7 \in \mathbb{F}_2$
Output: Shares $s^\star, z^\star \in \mathcal{B}^2(\mathbb{F}_2)$ such that $s = s_{\mathrm{L}}$, $z = z_{\mathrm{L}}$

1 $S^\star = s_{\mathrm{L}}^\star$
2 $Z^{\mathbf{0}}, \ldots, Z^{\mathbf{5}} = z_{\mathrm{L}}^{\mathbf{0}}, z_{\mathrm{L}}^{\mathbf{1}}, 0, \ldots, 0$
3 $s^\star = \mathrm{Reg}[\mathrm{Refresh}(S^\star, r_7)]$
4 $z^\star = \mathrm{Reg}[\mathrm{Refresh}((Z^{\mathbf{0}}, Z^{\mathbf{1}}), r_2)] \oplus \mathrm{Reg}[\mathrm{Refresh}((Z^{\mathbf{2}}, Z^{\mathbf{3}}), r_3)] \oplus \mathrm{Reg}[\mathrm{Refresh}((Z^{\mathbf{4}}, Z^{\mathbf{5}}), r_4)]$
5 **return** $s^\star, z^\star$

6 Specialized countermeasures to physical attacks

In this chapter, we present three side-channel countermeasures that are mostly specialized to the two schemes of lattice-based cryptography under study. The first proposal, dedicated to the masking of Kyber, has been submitted as a patent application. The two other proposals, unpublished as of this writing, respectively relate to a masked sampling operation for Dilithium, and to a shuffling countermeasure applicable to both schemes.

6.1 Masked ciphertext compression

We saw in § 3.6.3.5 that masked implementations of Kyber need a secure way to compare the reencrypted ciphertext with the received one. Since the received ciphertext is compressed, this comparison requires either a masked ciphertext-compression operation, or an ad-hoc masked comparison operation that can handle one ciphertext being masked but uncompressed, while the other is compressed but unprotected. The second method, chosen for instance by Bos *et al.* [BGR+21], lacks in generality: in particular, it does not allow to mask encapsulation fully. The first method, on the other hand, has not received much attention: apparently, a single solution for its implementation, due to Fritzmann *et al.* [FBR+22], has been proposed in the literature. This solution has been better specified by Coron *et al.* [CGMZ23] for the case of high-order masking. We recalled its principle in § 3.6.3.5.

A shortcoming of this solution is its reliance on high-precision arithmetic and high bit-width masking conversions. The other masked operations of Kyber use 12-bit arithmetic and 12-bit masking conversions; in contrast, Fritzmann *et al.* call for integer division with a 24-bit quotient (constituted of 11 integral bits and 13 fractional bits), and a 24-bit arithmetic-to-Boolean (A2B) masking conversion.

Our contribution We present here a different solution for the masking of Kyber compression, which can be generalized to any similar rounded rescaling operation. Our solution does not rely on higher-precision arithmetic or on larger-width masking conversions than the rest of the masked scheme.

Patent application The solution described in this section, co-invented by Gilles Van Assche (STMicroelectronics, Diegem, Belgium) and Guilhèm Assael, has been submitted as a patent application, which is currently pending. The text of the patent application has not been published yet.

6.1.1 Preliminaries

We start by introducing some background: we first define a new notation for partial sharings, then recall the security notion of Probe-Isolating Non Interference (PINI) and its composability properties. We then reformulate Kyber ciphertext compression as a more generic operation that we call *integer rescaling*, and further extend this operation to make it essentially lossless.

6.1.1.1 Notation

We remind the reader that the shares of a quantity x are denoted by $x^{\mathbf{0}}, \ldots, x^{\boldsymbol{n-1}}$, and that the set of these n shares is abbreviated as $x^{\star}$. For $0 \leqslant \boldsymbol{i} \leqslant \boldsymbol{j} < n$, we will denote by $x^{\boldsymbol{i:j}} = (x^{\boldsymbol{i}}, x^{\boldsymbol{i+1}}, \ldots, x^{\boldsymbol{j}})$ the subset of the shares having indexes from $\boldsymbol{i}$ to $\boldsymbol{j}$, inclusively.

As before, for a set S, $\mathcal{B}^n(S)$ and $\mathcal{A}_q^n(S)$ will respectively denote the set of n-share Boolean sharings and of n-share arithmetic sharings modulo q of elements of S.

6.1.1.2 Security model

We will prove the security of our construction by expressing it as a composition of more elementary gadgets. In contrast with the previous chapter, we will not explicitly mention the robustness of our construction against glitches and transitions. This frees us from having to describe the physical implementation of our gadgets.

Consequently, we will not need the O-PINI security notion [CS21] used in chapter 5: absent glitches and transitions, the security notion of Probe-Isolating Non Interference (PINI), introduced earlier by Cassiers and Standaert [CS20], is sufficient for trivial composability.

Definition 6.1 (Probe-Isolating Non-Interference [CS20])**.** *Let G be a gadget over n shares and P a set of t_1 probes on wires of G (called internal probes). Let A be a set of t_2 share indices. G is t-Probe-Isolating Non-Interfering (t-PINI) if and only if for all P and A such that $t_1 + t_2 \leqslant t$, there exists a set B of at most t_1 share indices such that probes in P, together with probes on all output shares having share index in A, can be simulated assuming knowledge of the input shares having index in $A \cup B$.*

We furthermore say that a gadget with n shares is PINI if it is $(n-1)$-PINI, in which case it is also t-PINI for any t [CS21]. PINI security is compatible with gadget composition, in the sense of the following property.

Proposition 6.1 (PINI composability [CS20])**.** *Any composition of t-PINI gadgets is itself t-PINI.*

Like O-PINI, a valuable aspect of the PINI security notion is that it is trivially satisfied by share-isolating gadgets, that is, gadgets implemented by circuits which can be partitioned into subcircuits each involving a single share index [CS20, Proposition 4] [CS21, Proposition 1].

About robust-probing security Provided the base gadgets are glitch-robust PINI, robustness against glitches is immediately transferred to the gadget composition [CS21]. Furthermore, robustness against transitions can be added through either of two means: by upgrading the base gadgets to glitch+transition-robust O-PINI (Output-Probe Isolating Non-Interference) security, which is stable through trivial composition, or by using glitch+transition-robust PINI gadgets with a more careful composition strategy [CS21].

6.1.1.3 Ciphertext compression as an integer rescaling operation

To ensure that our proposal is described in a generic way, we recall here the principle of Kyber compression without relying on the specific parameters used by the scheme: in effect, what we describe here is a generic rounded integer rescaling.

We define this rescaling operation as

$$\begin{aligned} \mathrm{Rescale}_{q,q',r} : [\![0, q-1]\!] &\longrightarrow [\![0, q'-1]\!] \\ x &\longmapsto \left\lfloor \frac{xq'+r}{q} \right\rfloor \bmod q' , \end{aligned} \tag{6.1}$$

where q and q' are arbitrary integers greater than or equal to 2 (usually, $q' < q$), and r is an integer belonging to $[\![0, q-1]\!]$. Parameters q and q' of the rescaling respectively specify the input and output range, while r determines the rounding strategy: $r = 0$ results in truncation, while $r = \lfloor q/2 \rfloor$ results in rounding to the nearest neighbor (ties being rounded up). Comparing eq. (6.1) with eq. (2.7), we notice that Kyber compression is indeed a special case of this rescaling function: $\mathrm{Compress}_d = \mathrm{Rescale}_{q_\mathrm{K}, 2^d, \lfloor q_\mathrm{K}/2 \rfloor}$.

6.1.1.4 Fractional rescaling

In order to compute the rescaling efficiently, we define an extension of integer rescaling, that not only computes the Rescale function, but also the error introduced by it through rounding. We define this operation by

$$\begin{aligned} \mathrm{DivMod}_{q,q',r} : [\![0, q-1]\!] &\longrightarrow [\![0, q'-1]\!] \times [\![0, q-1]\!] \\ x &\longmapsto \left(\left\lfloor \frac{xq'+r}{q} \right\rfloor \bmod q', (xq'+r) \bmod q \right) . \end{aligned} \tag{6.2}$$

A very important aspect of this function is that it computes a single Euclidean division, $(xq'+r) \div q$, and employs both its quotient (further reduced modulo q') and its remainder. In general, computing both results of the Euclidean division is only marginally more costly than computing either.

We call the first output of DivMod its *integral* result, which corresponds to Rescale, and the second output its *fractional* result, because it represents the rescaling error in units of $1/q$. We note that DivMod satisfies the following property:

$$(y, f) = \mathrm{DivMod}_{q,q',r}(x) \quad \Longrightarrow \quad qy + f \equiv xq' + r \pmod{qq'} . \tag{6.3}$$

6.1.2 Constructing high-order gadgets by lateral union of low-order ones

We establish here a result that will be helpful in proving the high-order security of our construction: given two PINI gadgets g', g'' having n', n'' shares respectively, we can build from them a PINI gadget G with $n'+n''$ shares by mapping the n' first (respectively n'' last) share indexes of the input and output sharings of G to the input and output sharings of g' (respectively g''). We call gadget G the *lateral union* of g' and g''. We start by formally defining this concept.

Definition 6.2 (Lateral union of gadgets)**.** *Let g', g'' be two distinct gadgets each having u input sharings and v output sharings, such that all of the following hold:*

- *g' has n' shares,*
- *g' has inputs $(U'^{\boldsymbol{\delta}}_j)_{0\leqslant j<u,0\leqslant\boldsymbol{\delta}<n'}$, where $U'^{\star}_j$ is its j^{th} input sharing,*
- *g' has outputs $(V'^{\boldsymbol{\delta}}_j)_{0\leqslant j<v,0\leqslant\boldsymbol{\delta}<n'}$, where $V'^{\star}_j$ is its j^{th} output sharing,*

and the equivalent properties hold for g'', replacing all primes $'$ with double primes $''$.

Denote by G the gadget with $n'+n''$ shares, u input sharings and v output sharings, constructed from the union of g_0 and g_1 in the following way:

- *for $0\leqslant j<u$, the j^{th} input sharing of G is $(U^{\boldsymbol{\delta}}_j)_{0\leqslant\delta<n'+n''}$, where $U^{\boldsymbol{\delta}}_j = U'^{\boldsymbol{\delta}}_j$ if $\delta<n'$ and $U^{\boldsymbol{\delta}}_j = U''^{\boldsymbol{\delta}-n'}_j$ otherwise,*
- *for $0\leqslant j<v$, the j^{th} output sharing of G is $(V^{\boldsymbol{\delta}}_j)_{0\leqslant\delta<n'+n''}$, where $V^{\boldsymbol{\delta}}_j = V'^{\boldsymbol{\delta}}_j$ if $\delta<n'$ and $V^{\boldsymbol{\delta}}_j = V''^{\boldsymbol{\delta}-n'}_j$ otherwise.*

We call G the lateral union *of g' and g'', and inductively extend this notion to the lateral union of more than two gadgets.*

Proposition 6.2. *If G is the lateral union of two gadgets g' and g'', both of which are PINI, then G is likewise PINI. The same holds for lateral unions of more than two gadgets.*

Proof. Consider g_0, g_1 and G satisfying the hypotheses of the definition. Let $t = n'+n''-1$. Let I and O be sets of input and output probes on G, such that $|I| = t_1$, and when denoting by A the set of the share indexes in O, $|A| = t_2$, with $t_1+t_2 = t$.

We partition I into $I'\cup I''$ such that I' is the set of internal probes belonging to g' and I'' is the set of internal probes belonging to g''. We also partition $O = O'\cup O''$ in the same way. Let us denote by $A'\subset[\![0,n'-1]\!]$ (respectively, $A''\subset[\![0,n''-1]\!]$), the set of share indexes *relative to* g' (respectively, to g'') in O' (respectively, O'').

Since g' is t'-PINI for any t', there exists a set B' of at most $|I'|$ share indexes such that observations corresponding to I' and O' can be simulated using only the shares with indexes $A'\cup B'$ of each input sharing of g'.

We can determine the equivalent set B'' for g'' since it is PINI as well.

Let us now define $B = B'\cup\{i+n' \mid i\in B''\}$. We trivially have $|B| = |B'|+|B''|\leqslant |I'|+|I''| = t_1$. We can now simulate probes in I and O separately (depending on which member gadget they belong to) using the two above-mentioned simulators, since we know the input shares of g' with indexes in $A'\cup B'$ and the input shares of g'' with indexes in $A''\cup B''$. Since G has $n = n'+n''$ shares and is $(n-1)$-PINI, it is PINI. □

Note that the proposition requires that g' and g'' be secure at order $n'-1$ and $n''-1$ respectively (which is their maximum security order given their number of shares). This proposition can be seen as an extension of [CS21, Proposition 1], which states that share-isolating gadgets are (O-)PINI.

Extending this result to robust-probing security The above result can be extended to the O-PINI security notion [CS21] employed in chapter 5: *any lateral union of O-PINI gadgets is O-PINI*. We recall that this stronger security notion additionally requires simulating, in the same context as above, the output shares having index in B. The proof can be adapted to this context by noticing that the output shares of G with indexes in B exactly correspond to the output shares of g' with indexes in B' together with the output shares of g'' with indexes in B'', which can be simulated under the assumption that g' and g'' are O-PINI. The result also holds for robust probing, provided the probe expansion scheme is compatible with the partitioning of G into g' and g'', *i.e.* expanding probes in G is equivalent to expanding them in g' and in g'' separately. In particular, it holds for glitch+transition-robust probing if the members of the lateral union are implemented by disjoint structural gates and wires.

6.1.3 Supporting gadgets

We define a gadget that, despite containing essentially no logic, will be crucial to our security proof. Given $m, n, w \geqslant 1$, we define the SplitShares$_w^{m,n}$ gadget that takes two w-bit input sharings with m and n shares respectively, and constructs from them two sharings each having $m+n$ shares by adjoining null shares to the initial sharings. The definition of the gadget can be seen in algorithm 6.1. Since the definitions that we use need gadgets to have the same number of shares in all of their input and output sharings, we regard the two input sharings of SplitShares as a single one of order $m+n$. We highlight that the indexes that correspond to the null shares are essential to the security of the gadget, since they ensure that the input shares keep the same index at the output.

Note that the output sharings of SplitShares will be refreshed when entering a nonlinear PINI gadget, which will ensure that no shares are stuck at zero within said gadget.

Algorithm 6.1: SplitShares$_w^{m,n}$ gadget

Input: Sharing $(a^{\mathbf{0}}, \ldots, a^{\mathbf{m+n-1}}) \in (\mathbb{F}_2^w)^{m+n}$ (arbitrary masking scheme)
Output: Sharings $(x^{\mathbf{0}}, \ldots, x^{\mathbf{m+n-1}}) \in (\mathbb{F}_2^w)^{m+n}$ and $(y^{\mathbf{0}}, \ldots, y^{\mathbf{m+n-1}}) \in (\mathbb{F}_2^w)^{m+n}$ such that x (resp. y) corresponds to unmasking the first m (resp. the last n) shares of a

1 **for** $\boldsymbol{i} = 0$ to $m+n-1$ **do**
2 **if** $\boldsymbol{i} < m$ **then** $x^{\boldsymbol{i}} = a^{\boldsymbol{i}}$, $y^{\boldsymbol{i}} = 0 \in \mathbb{F}_2^w$
3 **else** $x^{\boldsymbol{i}} = 0 \in \mathbb{F}_2^w$, $y^{\boldsymbol{i}} = a^{\boldsymbol{i}}$
4 **end**

Proposition 6.3. *The* SplitShares *gadget is PINI.*

Proof. SplitShares being share-isolating, the result follows from [CS20, Prop. 4]. □

We furthermore define the $\text{SecDivMod}^n_{q,q',r}$ gadget as the share-wise application of the $\text{DivMod}_{q,q',r}$ function over an arithmetic sharing, according to algorithm 6.2.

Algorithm 6.2: $\text{SecDivMod}^n_{q,q',r}$ gadget: share-wise fractional rescaling

Parameter: Input and output moduli q, q', number of shares $n \geqslant 2$, rounding parameters $r_0, \ldots, r_{n-1}$
Input: Arithmetic sharing modulo q, $(x^{\mathbf{0}}, \ldots, x^{\boldsymbol{n-1}}) \in \mathcal{A}^n_q([\![0, q-1]\!])$
Output: Arithmetic sharings $(y^{\mathbf{0}}, \ldots, y^{\boldsymbol{n-1}}) \in \mathcal{A}^n_{q'}([\![0, q'-1]\!])$ and $(f^{\mathbf{0}}, \ldots, f^{\boldsymbol{n-1}}) \in \mathcal{A}^n_q([\![0, q-1]\!])$, respectively modulo q' and q, such that $\forall \boldsymbol{i}, (y^{\boldsymbol{i}}, f^{\boldsymbol{i}}) = \text{DivMod}_{q,q',r_{\boldsymbol{i}}}(x^{\boldsymbol{i}})$

1 **for** $\boldsymbol{i} = 0$ to $n-1$ **do** $(y^{\boldsymbol{i}}, f^{\boldsymbol{i}}) = \text{DivMod}_{q,q',r_{\boldsymbol{i}}}(x^{\boldsymbol{i}})$
2 **return** $y^\star, f^\star$

Proposition 6.4. *The* SecDivMod *gadget is PINI*

Proof. By definition, SecDivMod is share-isolating, from which the result follows. □

We will make use of the single-bit Boolean-to-Arithmetic conversion of Schneider *et al.* [SPOG19], which we denote by B2Abit^n_q. This gadget, already presented in § 3.6.2.1, gets as input a single bit expressed over n Boolean shares, and converts it to arithmetic shares modulo q (with no restriction on the value of the modulus q). An important property of this gadget is that it has much lower complexity than a conversion from Boolean to modulo-q arithmetic masking supporting any input between 0 and $q-1$ [SPOG19].

We now introduce a few other auxiliary gadgets, all working over Boolean sharings with $n \geqslant 2$ shares. We first assume the availability of a SecAddBin^n_w gadget performing secure arithmetic addition over w-bit Boolean shares. This gadget, whose description and coverage in previous works have been detailed in chapter 5, gets as inputs two w-bit Boolean sharings (each having n shares), and computes their sum modulo 2^w, as well as an output carry. Both results are encoded as n Boolean shares. We assume this gadget to satisfy PINI security. In particular, our proposal in chapter 5 gives such an implementation for first-order masking, if we disable modular reduction and make the carry available at the output of the gadget.

When the sum output of this gadget is not needed, and only the carry output is required, we can simplify the gadget into SecCarry^n_w, which receives two w-bit Boolean sharings $x^\star$ and $y^\star$ (each having n shares), and returns a Boolean sharing of a single bit indicating whether $x + y \geqslant 2^w$.

We also consider a $\text{SecAddOverflow}^n_q$ gadget, that performs secure addition modulo q and indicates modular overflow (that is, whether the sum of the two inputs is greater than or equal to q). More precisely, the gadget gets two Boolean input sharings $x^\star$ and $y^\star$, satisfying $x < q$ and $y < q$, and outputs Boolean sharings of $(x + y) \bmod q$ and of the truth value of $x + y \geqslant q$. If only first-order resistance is needed ($n = 2$), this is the full proposal of chapter 5, in which we also output the masked signal indicating whether modular reduction is performed (variable e in algorithm 5.3 and fig. 5.3). Previous works implementing this operation are also referenced in chapter 5.

By analogy with SecCarry, the SecOverflow^n_q gadget corresponds to $\text{SecAddOverflow}^n_q$ stripped of its sum output, with only the modular-overflow output kept.

Note that, while the q parameter of $\mathrm{SecAddOverflow}_q^n$ and $\mathrm{SecOverflow}_q^n$ indicates the modulus, the w parameter of $\mathrm{SecAddBin}_w^n$ and $\mathrm{SecCarry}_w^n$ denotes the number of bits of the operands, so that the corresponding modulus is 2^w. All the above gadgets are assumed to satisfy PINI security, which is notably possible at the first order through our proposal of chapter 5.

6.1.4 First-order masking of integer rescaling

6.1.4.1 Principle

Let us now examine what happens when rescaling an integer represented over two arithmetic shares. Let us fix parameters q, q' and r. Consider a first-order arithmetic sharing modulo q, denoted by $(x^{\mathbf{0}}, x^{\mathbf{1}}) \in \mathcal{A}_q^2(\llbracket 0, q-1 \rrbracket)$. We recall that the value represented by this sharing is $x = (x^{\mathbf{0}} + x^{\mathbf{1}}) \bmod q$. We have

$$\mathrm{Rescale}_{q,q',r}(x) \equiv \left\lfloor \frac{xq' + r}{q} \right\rfloor \equiv \frac{xq' + r - (xq' + r) \bmod q}{q} \pmod{q'}.$$

Let us choose $r_0, r_1 \in \mathbb{Z}$ such that $r_0 + r_1 = r$. If we apply the rescaling on each share separately with rounding parameters r_0 and r_1 respectively, the error with respect to rescaling x directly is

$$\begin{aligned}
\Delta_q^2(x^{\mathbf{0}}, x^{\mathbf{1}}) &\equiv \mathrm{Rescale}_{q,q',r}(x) - \mathrm{Rescale}_{q,q',r_0}(x^{\mathbf{0}}) - \mathrm{Rescale}_{q,q',r_1}(x^{\mathbf{1}}) \pmod{q'}, \\
&\equiv \frac{(x^{\mathbf{0}}q' + r_0) \bmod q + (x^{\mathbf{1}}q' + r_1) \bmod q - (xq' + r) \bmod q}{q} \pmod{q'}, \\
&\in \{0, 1\}.
\end{aligned}$$

We specifically have

$$\Delta_q^2(x^{\mathbf{0}}, x^{\mathbf{1}}) = \left[(x^{\mathbf{0}}q' + r_0) \bmod q + (x^{\mathbf{1}}q' + r_1) \bmod q \geqslant q \right], \tag{6.4}$$

where the Iverson bracket $[\cdots]$ associates 0 to false statements and 1 to true statements.

Assuming we can compute this correction function Δ in a secure way, integer rescaling can be performed over shared values by applying the rescaling share-wise with rounding parameters that sum to r, and adding the correction value to the rescaled shares.

6.1.4.2 Secure implementation

We show in fig. 6.1 the composition of gadgets through which we implement first-order masked rescaling from q to q', with rounding parameter r. In this diagram, each wire carries a first-order sharing, either in the Boolean domain (blue wires) or in the arithmetic domain (brown wires). We use three share-isolating gadgets, represented in light blue: $\mathrm{SecDivMod}_{q,q',r}^2$, $\mathrm{SplitShares}_w^{1,1}$, and addition modulo q'. Two nonlinear gadgets, $\mathrm{SecOverflow}_q^2$ and $\mathrm{B2A}_{q'}$ are also needed, and are represented in orange. The arithmetic shares modulo q of the input value $x^\star$ are first subjected to fractional rescaling through SecDivMod. The integral result $y^\star$ is represented over mod-q' arithmetic shares, while

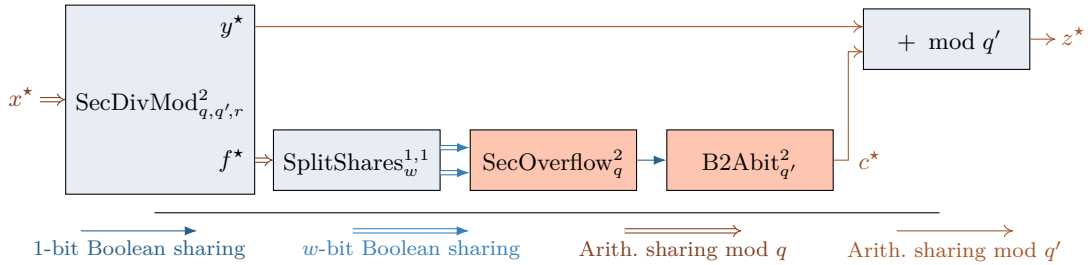


Figure 6.1: First-order masked computation of integer rescaling. Symbol w denotes $\lceil \log_2 q \rceil$. Each wire carries a 1st-order sharing among the types listed at the bottom.

the fractional result is still on mod-q arithmetic shares. This latter result is further split into a pair of Boolean shares (each having $w = \lceil \log_2 q \rceil$ bits), which are added together to check whether their sum is greater than q, thereby implementing the Δ^2_q function. This indication, output by SecOverflow over a single-bit Boolean sharing, is converted to arithmetic shares modulo q', denoted by $c^\star$. Finally, the integral result $y^\star$ of fractional rescaling is summed share-wise with the correction value $c^\star$, giving the expected result of rescaling $z^\star$ over arithmetic shares modulo q'.

The whole process is also formalized in algorithm 6.3, with two variants which differ in how the fractional correction is computed based on the available gadgets. In the first solution, we use SecOverflow^2_q to perform the comparison with q directly (lines 1.2 to 1.3), while in the second solution, we subtract q from $f^{\mathbf{0}}$ beforehand, and simply check whether summing this result with $f^{\mathbf{1}}$ generates an output carry (lines 2.2 to 2.3).

Algorithm 6.3: SecRescale$^2_{q,q',r}$ gadget: first-order secure integer rescaling

Parameter: Input and output moduli q and q', input-modulus width $w = \lceil \log_2 q \rceil$, rounding parameters r_0, r_1 such that $r_0 + r_1 = r$

Input: Arithmetic sharing modulo q, $(x^{\mathbf{0}}, x^{\mathbf{1}}) \in \mathcal{A}^2_q([\![0, q-1]\!])$

Output: Arithmetic sharing modulo q', $(z^{\mathbf{0}}, z^{\mathbf{1}}) \in \mathcal{A}^2_{q'}([\![0, q'-1]\!])$ such that $z = \text{Rescale}_{q,q',r}(x)$

Solution 1

1.1 $y^\star, f^\star = \text{SecDivMod}^2_{q,q',r}(x^\star)$

// Trivial Boolean sharings of f^0 and of f^1

1.2 $g^\star_0, g^\star_1 = \text{SplitShares}^{1,1}_w(f^\star)$

1.3 $b^\star = \text{SecOverflow}^2_q(g^\star_0, g^\star_1)$

1.4 $c^\star = \text{B2Abit}^2_{q'}(b^\star)$

1.5 $z^\star = (y^\star + c^\star) \bmod q'$

1.6 **return** $z^\star$

Solution 2

2.1 $y^\star, f^\star = \text{SecDivMod}^2_{q,q',r}(x^\star)$

// Trivial Boolean sharings of $f^0 + 2^w - q$ and of f^1

2.2 $g^\star_0, g^\star_1 = \text{SplitShares}^{1,1}_w((f^{\mathbf{0}} + 2^w - q, f^{\mathbf{1}}))$

2.3 $b^\star = \text{SecCarry}^2_w(g^\star_0, g^\star_1)$

2.4 $c^\star = \text{B2Abit}^2_{q'}(b^\star)$ . . *// 1-bit B2A conversion*

2.5 $z^\star = (y^\star + c^\star) \bmod q'$. *// Share-wise addition*

2.6 **return** $z^\star$

Since this first-order gadget is a special case of the high-order gadget described next, we do not yet prove its security, and will prove it generically for any order.

6.1.5 Extension to higher-order masking

The general principle used in the previous subsection, namely associating a share-wise fractional rescaling with a nonlinear correction step, can actually be applied at any order. However, the computation of the correction becomes more complex, because the amplitude of the rounding error is proportional to the masking order: with n shares, this error is comprised between 0 and $n-1$.

Using the same notation as before, the correction function for n shares becomes

$$\Delta_q^n(x^{\mathbf{0}}, \ldots, x^{\boldsymbol{n-1}}) = \left\lfloor \frac{\sum_{\boldsymbol{i}=0}^{n-1}((x^{\boldsymbol{i}}q' + r_{\boldsymbol{i}}) \bmod q)}{q} \right\rfloor, \tag{6.5}$$

which corresponds to summing the fractional results of share-wise rescaling, dividing the result by q, and only keeping the integral part of the quotient. While this truncated division could be performed by comparing with q in the first-order case, this is no longer possible at higher orders. Our solution, instead, is to add the shares progressively, checking each time if a reduction modulo q is needed, and to count how many reductions have been performed.

We describe in algorithm 6.4 the implementation of secure integer rescaling with n shares. As before, we start by applying fractional rescaling share-wise. This operation gives two sharings: $y^\star$, and $f^\star$. The former is an approximation of the rescaled result, which may underestimate it by up to $n-1$ units, while the latter contains the rescaling error. The key part of our algorithm, contained in lines 2 to 18, consists in progressively migrating this error from $f^\star$ to $y^\star$. A graphical representation of this process is also given in fig. 6.2 for seven shares.

We compute the correction by summing the shares $f^\star$ modulo q in a binary tree structure, and adding 1 to y every time a modular sum overflows from $[\![0, q-1]\!]$. The sum of the fractional parts is protected through Boolean masking, with a number of shares progressively increased to match the number of fractional shares involved in each partial sum[1]. In algorithm 6.4, the progress of the summation is represented by J, a composition of the number of shares n, that is, an ordered list of positive integers that sum to n. Throughout the algorithm, each term of J indicates how many fractional shares have been summed together, and the number of shares of the corresponding sum.

Practically, each fractional share $f^{\boldsymbol{i}}$ is initially a 0^{th} order (one-share) sharing: this is expressed by J being the sequence of n ones. Each iteration of the loop starting at line 3 applies one level of the summation tree. The first iteration of the inner loop at line 5 takes the first pair of Boolean sharings from $f^\star$, with J_0 and J_1 shares, and replaces them with a Boolean sharing of their sum modulo q, this time over $J_0 + J_1$ shares. The masked bit b indicating whether modular reduction occurred, represented over $J_0 + J_1$ shares as well, is converted into a modulo-q' arithmetic sharing, c. The following iterations proceed likewise on the rest of $f^\star$ and $c^\star$, merging every pair of Boolean sharings into a sharing of their sum while preserving the overall number of shares. If composition J

[1] This kind of recursive implementation over increasing numbers of shares has been previously used, in particular, by Coron *et al.* [CGV14].

Algorithm 6.4: $\mathrm{SecRescale}^n_{q,q',r}$ gadget: high-order secure integer rescaling

Parameter: Input and output moduli q and q'; number of shares $n \geqslant 2$; rounding parameters $r_0, \ldots, r_d$ such that $\sum_i r_i = r$; width $w = \lceil \log_2 q \rceil$ of the input modulus
Input: Arithmetic sharing modulo q: $x^\star \in \mathcal{A}^n_q(\llbracket 0, q-1 \rrbracket)$
Output: Arithmetic sharing modulo q': $z^\star \in \mathcal{A}^n_{q'}(\llbracket 0, q'-1 \rrbracket)$, such that $z = \mathrm{Rescale}_{q,q',r}(x)$

1 $y^\star, f^\star = \mathrm{SecDivMod}^n_{q,q',r}(x^\star)$
2 $J = (1)_{0 \leqslant i < n}$ *// J is a composition (ordered partition) of the number of shares; its i^{th} component is J_i*
3 **while** $|J| > 1$ **do** *// Loop until J has been reduced to the trivial composition (n)*
4 $\quad \boldsymbol{j} = 0$
5 $\quad$ **for** $i = 0$ to $\lfloor |J|/2 \rfloor$ **do**
6 $\quad\quad \boldsymbol{j'} = \boldsymbol{j} + J_{2i} + J_{2i+1} - 1$
7 $\quad\quad F_0^{\boldsymbol{j}:\boldsymbol{j'}}, F_1^{\boldsymbol{j}:\boldsymbol{j'}} = \mathrm{SplitShares}_w^{J_{2i}, J_{2i+1}}(f^{\boldsymbol{j}:\boldsymbol{j'}})$
8 $\quad\quad f^{\boldsymbol{j}:\boldsymbol{j'}}, b^{\boldsymbol{j}:\boldsymbol{j'}} = \mathrm{SecAddOverflow}_q^{J_{2i}+J_{2i+1}}(F_0^{\boldsymbol{j}:\boldsymbol{j'}}, F_1^{\boldsymbol{j}:\boldsymbol{j'}})$ *// Sum the Boolean sharings pair-wise*
9 $\quad\quad c^{\boldsymbol{j}:\boldsymbol{j'}} = \mathrm{B2Abit}_{q'}^{J_{2i}+J_{2i+1}}(b^{\boldsymbol{j}:\boldsymbol{j'}})$ *// One-bit* B2A *conversion*
10 $\quad\quad \boldsymbol{j} = \boldsymbol{j'} + 1$
11 $\quad$ **end**
12 $\quad$ **if** $|J| \equiv 1 \pmod 2$ **then**
13 $\quad\quad \boldsymbol{j'} = n - 1$
14 $\quad\quad c^{\boldsymbol{j}:\boldsymbol{j'}} = 0, \ldots, 0$
15 $\quad$ **end**
16 $\quad y^\star = (y^\star + c^\star) \bmod q'$ *// Share-wise modular addition of current correction value*
17 $\quad J = (J_{2i} + J_{2i+1})_{0 \leqslant i < \lceil |J|/2 \rceil}$ *// Merge pairs of terms of J; for notational convenience, $J_{|J|} = 0$*
18 **end**
19 $z^\star = y^\star$
20 **return** $z^\star$

had an odd number of terms at this level of the tree (line 12), the lone last sharing is kept unchanged in $f^\star$, and the corresponding shares of $c^\star$ are set to zero[2].

At this point, $c^\star$ contains a modular arithmetic sharing of the number of modular reduction that have occurred at this level of the summation tree: indeed, notice that concatenating arithmetic sharings gives a higher-order sharing of their sum. $c^\star$ can then be summed share-wise into $y^\star$ to apply the corresponding correction. Finally, the composition J of the number of shares is updated by summing its terms in pairs (leaving the last lone term unchanged if applicable) to reflect how the pairs of sharings have been merged. The next iteration of the outer loop then computes the following level of the summation tree, until J has been reduced to a single term. This condition indeed means that all fractional shares have been summed, and all required corrections have been applied to $y^\star$.

Comparing algorithm 6.4 with fig. 6.2, one can notice a difference that is small but sometimes important for performance. On one hand, the algorithm involves the same gadgets

[2]When the number of shares is not a power of two, different shapes are possible for the correction tree. The optimal tree will depend on how exactly the runtime of the base gadgets evolves with their masking order, and may not be the one we choose here.

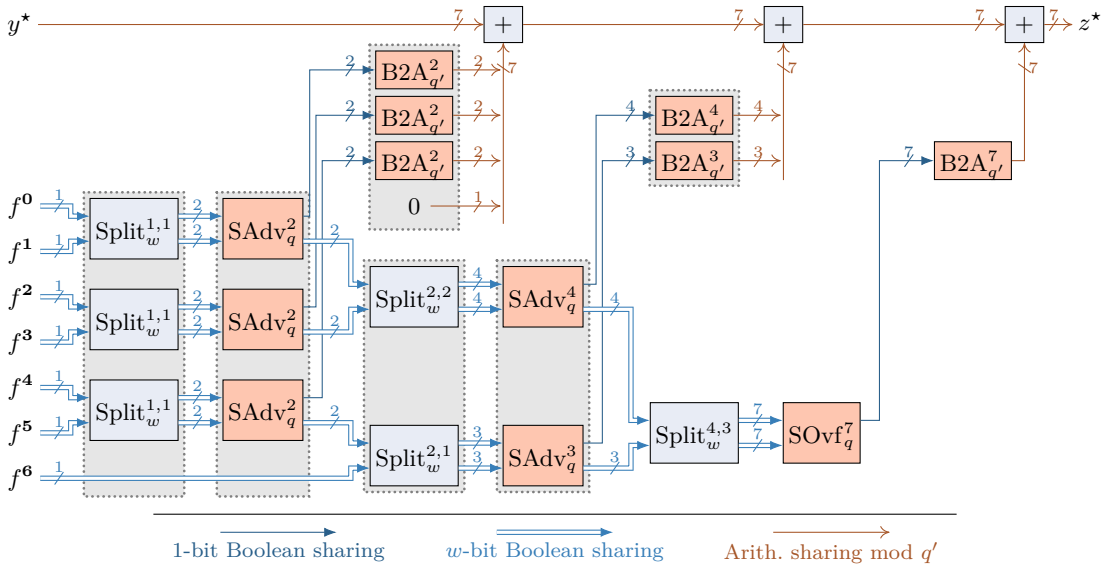


Figure 6.2: Correction of rescaling from q to q' with seven shares. Symbol w denotes $\lceil \log_2 q \rceil$. Each wire carries a sharing among the types listed at the bottom, with the number of shares indicated next to every wire. Gadgets grouped within gray regions constitute 7-share gadgets through lateral union. Abbreviations Split, SAdv, SOvf, and B2A stand for SplitShares, SecAddOverflow, SecOverflow, and B2Abit respectively. The top and bottom output sharings of SAdv respectively correspond to its overflow and sum outputs.

SplitShares, SecAddOverflow and B2A at each level of the tree for clarity. On the other hand, fig. 6.2 uses SecOverflow instead of SecAddOverflow at the last level of the tree: only the modular-overflow indication is needed, and the sum can be discarded. Depending on the implementation of SecAddOverflow and SecOverflow, this optimization can bring a significant performance gain since this gadget is masked at the highest order, n.

We have not discussed yet how to choose the set of rounding parameters, $r_0, \ldots, r_{n-1}$, aside from the constraint that their sum is r. In fact, this choice does not matter security-wise, and has negligible influence performance-wise, apart from saving an arithmetic addition and a modular reduction by q' within DivMod if the corresponding rounding parameter is zero. We thus suggest two strategies to distribute the rounding parameter.

- If the number of shares n divides r, all rounding parameters can be made equal: $\forall i \in [\![0, n-1]\!], r_i = r/n$. This is notably the case for Kyber ($r = \lfloor q_\mathrm{K}/2 \rfloor = 1664$) when the number of shares is a power of two, up to $n = 2^7$.

- In other cases, the simplest choice is to set $r_0 = r$ and to null the other rounding parameters, $\forall i \in [\![1, n-1]\!], r_i = 0$.

6.1.6 Correctness and security

To prove the correctness and security of algorithm 6.4, we start by analyzing the properties of the outer loop. We first state in lemma 6.1 an invariant satisfied by J.

Lemma 6.1 (First invariant of the loop at line 3 of algorithm 6.4)**.** *J is a composition of n at all times, that is, $\sum_i J_i = n$, and for all i, $J_i \geqslant 1$.*

Proof. J is initialized to n copies of 1. It is then only updated at line 17, where its terms are summed in nonoverlapping pairs (if there are an odd number of terms, the last one is left unchanged), preserving the overall sum. □

For the following, we introduce an auxiliary function, HybridUnmask. This function takes two tuples (finite ordered sets) J and f of positive integers such that $\sum_i J_i = |f|$, and is recursively defined for the empty tuples by

$$\text{HybridUnmask}(\varnothing, \varnothing) = 0\,,$$

and for nonempty J and f,

$$\text{HybridUnmask}(J, f) = \left(\bigoplus_{i=0}^{J_0-1} f_i\right) + \text{HybridUnmask}((J_1, \ldots, J_{|J|-1}), (f_{J_0}, \ldots, f_{|f|}))\,.$$

This function thus splits f in groups of size given by the elements of J, reduces each group to a single element through a bitwise XOR, and sums over all groups.

Lemma 6.2. *The computation in lines 7 to 9 of algorithm 6.4 satisfies*

$$\bigoplus_{\boldsymbol{k}=\boldsymbol{j}}^{\boldsymbol{j'}} f^{\boldsymbol{k}} + q \sum_{\boldsymbol{k}=\boldsymbol{j}}^{\boldsymbol{j'}} c^{\boldsymbol{k}} \equiv \text{HybridUnmask}((J_{\mathit{before},2i}, J_{\mathit{before},2i+1}), f_{\mathit{before}}^{\boldsymbol{j}:\boldsymbol{j'}}) \pmod{qq'}\,,$$

where symbols subscripted with $_{\mathit{before}}$ refer to the value of the corresponding variables before the execution of said lines, and symbols without this subscript refer to their value after execution.

Proof. We have

$$\begin{aligned}\bigoplus_{\boldsymbol{k}=\boldsymbol{j}}^{\boldsymbol{j'}} f^{\boldsymbol{k}} + q \sum_{\boldsymbol{k}=\boldsymbol{j}}^{\boldsymbol{j'}} c^{\boldsymbol{k}} &\equiv \bigoplus_{\boldsymbol{k}=\boldsymbol{j}}^{\boldsymbol{j'}} f^{\boldsymbol{k}} + q \bigoplus_{\boldsymbol{k}=\boldsymbol{j}}^{\boldsymbol{j'}} b^{\boldsymbol{k}} \equiv \left(\bigoplus_{\boldsymbol{k}=\boldsymbol{j}}^{\boldsymbol{j'}} F_0^{\boldsymbol{k}}\right) + \left(\bigoplus_{\boldsymbol{k}=\boldsymbol{j}}^{\boldsymbol{j'}} F_1^{\boldsymbol{k}}\right) \pmod{qq'} \\ &\equiv \text{HybridUnmask}((J_{\text{before},2i}, J_{\text{before},2i+1}), f_{\text{before}}^{\boldsymbol{j}:\boldsymbol{j'}}) \pmod{qq'}\end{aligned}$$ □

Lemma 6.3 (Second invariant of the loop at line 3 of algorithm 6.4)**.** *Considered at line 3, the following equivalence holds across all iterations of the outer loop:*

$$\text{HybridUnmask}(J, f^\star) + q \sum_{\boldsymbol{i}=0}^{n-1} y^{\boldsymbol{i}} \equiv \sum_{\boldsymbol{i}=0}^{n-1} (x^{\boldsymbol{i}} q' + r_{\boldsymbol{i}}) \pmod{qq'}\,.$$

Proof. First note that it is legal to compute HybridUnmask$(J, f^\star)$ since, by lemma 6.1, J always sums to n, and $f^\star$ is an n-tuple.

Before the first iteration of the outer loop, the property is satisfied since by eq. (6.3),

$$\text{HybridUnmask}(J, f^\star) + q\sum_{\boldsymbol{k}=0}^{n-1} y^{\boldsymbol{k}} = \sum_{\boldsymbol{k}=0}^{n-1}(f^{\boldsymbol{k}} + qy^{\boldsymbol{k}}) \equiv \sum_{\boldsymbol{k}=0}^{n-1}(x^{\boldsymbol{k}}q' + r_{\boldsymbol{k}}) \pmod{qq'}.$$

At an arbitrary iteration of the loop, denote with subscript $_{\text{before}}$ the value of variables immediately before this iteration, and without subscript, the values immediately after this iteration. Assume the property holds before the iteration. By summing both sides of the equivalence in lemma 6.2 over all successive values of $(\boldsymbol{j}, \boldsymbol{j'})$ at line 7, we get

$$\text{HybridUnmask}(J, f^\star) + q\sum_{\boldsymbol{k}=0}^{n-1} c^{\boldsymbol{k}} \equiv \text{HybridUnmask}(J_{\text{before}}, f^\star_{\text{before}}) \pmod{qq'}.$$

Adding $q\sum_{\boldsymbol{k}=0}^{n-1} y^{\boldsymbol{k}}_{\text{before}}$ to each side, and applying the induction hypothesis, we get

$$\text{HybridUnmask}(J, f^\star) + q\left(\sum_{\boldsymbol{k}=0}^{n-1} c^{\boldsymbol{k}} + \sum_{\boldsymbol{k}=0}^{n-1} y^{\boldsymbol{k}}_{\text{before}}\right) \equiv \sum_{\boldsymbol{i}=0}^{n-1}(x^{\boldsymbol{i}}q' + r_{\boldsymbol{i}}) \pmod{qq'},$$

from which we get the expected equivalence since $y^\star = c^\star + y^\star_{\text{before}}$. □

We can now prove that the proposed algorithm terminates and that it correctly computes the Rescale function (theorem 6.1).

Theorem 6.1 (Correctness and termination of algorithm 6.4)**.** *Algorithm 6.4 terminates after* $\lceil \log_2 n \rceil$ *iterations of the loop at line 3. It correctly computes* $z^\star$ *such that* $(\sum_{\boldsymbol{i}=0}^{n-1} z^{\boldsymbol{i}}) \bmod q' = \text{Rescale}_{q,q',r}((\sum_{\boldsymbol{i}=0}^{n-1} x^{\boldsymbol{i}}) \bmod q)$.

Proof. Before the first iteration, $|J| = n$. At each iteration, line 17 halves the length of J, rounding up. Thus, the loop stops after k iterations, k being the least integer such that $\lceil n/2^k \rceil = 1$.

Due to the condition of the outer loop, $|J| \leqslant 1$ once the algorithm has terminated. Since by lemma 6.1, J is a composition of n, we conclude that it is the singleton (n) at this point. Applying lemma 6.3 there, we deduce that

$$\text{HybridUnmask}((n), f^\star) + q\sum_{\boldsymbol{i}=0}^{n-1} y^{\boldsymbol{i}} \equiv \sum_{\boldsymbol{i}=0}^{n-1}(x^{\boldsymbol{i}}q' + r_{\boldsymbol{i}}) \pmod{qq'},$$

or equivalently,

$$\bigoplus_{\boldsymbol{i}=0}^{n-1} f^{\boldsymbol{i}} + q\sum_{\boldsymbol{i}=0}^{n-1} z^{\boldsymbol{i}} \equiv q'\left(\sum_{\boldsymbol{i}=0}^{n-1} x^{\boldsymbol{i}}\right) + r \pmod{qq'}.$$

As $f^\star$ is the sum output of the $\text{SecAddOverflow}_q^n$ gadget, $\bigoplus_{\boldsymbol{i}=0}^{n-1} f^{\boldsymbol{i}} < q$. Thus, the floor division by q of each side of the previous equation gives the sought result,

$$\sum_{\boldsymbol{i}=0}^{n-1} z^{\boldsymbol{i}} \equiv \left\lfloor \frac{q'(\sum_{\boldsymbol{i}=0}^{n-1} x^{\boldsymbol{i}}) + r}{q} \right\rfloor \pmod{q'}\,. \qquad \square$$

We finally prove the security of our construction (theorem 6.2) through a compositional approach: low-order gadgets are merged into n-share gadgets through lateral union, and the resulting gadgets are composed under our working assumption of PINI base gadgets.

Theorem 6.2 (Security of algorithm 6.4)**.** *Algorithm 6.4 implements a PINI gadget over n shares.*

Proof. Consider an arbitrary iteration of the outer loop of algorithm 6.4, and iteration i of the inner loop starting at line 5. $\text{SplitShares}_w^{J_{2i},J_{2i+1}}$ is a PINI gadget with $J_{2i} + J_{2i+1}$ shares by proposition 6.3. By assumption, the same holds for SecAddOverflow and B2Abit (see section 6.1.3). Through the composition of PINI gadgets, each iteration of the inner loop is thus itself a PINI gadget over $J_{2i} + J_{2i+1}$ shares. Furthermore, if $|J|$ is odd, lines 12 to 15 implement a gadget over $J_{|J|-1}$ shares, that ignores its $f^{\boldsymbol{j}:\boldsymbol{j}'}$ input sharing and outputs an all-zero sharing as $c^{\boldsymbol{j}:\boldsymbol{j}'}$. Being share-isolating, it is similarly PINI.

Therefore, lines 5 to 12, in which the inner loop has been unrolled, implement an n-share lateral union of PINI gadgets, which is itself PINI by proposition 6.2.

The share-wise addition at line 16 and the $\text{SecDivMod}_{q,q',r}^n$ gadget at line 1 satisfy the same security property (by linearity and by proposition 6.4). The whole algorithm is thus a composition of n-share PINI gadgets. $\square$

6.1.7 Optimization of the upper level of the correction tree

For simplicity, our description of the gadget in algorithm 6.4 uses a homogeneous correction tree, having the same types of gadgets at each level. However, as already shown in fig. 6.2, we can actually simplify the upper level of the tree by replacing SecAddOverflow_q with SecOverflow_q since only the overflow output of the former gadget is needed.

Furthermore, SecOverflow_q itself can be replaced with a composition of gadgets of possibly lower complexity (depending on their exact implementation). We recall that Fritzmann *et al.* [FBR+22] proposed, in the context of computing the secure addition modulo q of two quantities a and b, to determine whether modular overflow occurs by instead calculating $a + (b - q)$, and checking the sign of the result, which is directly obtained by checking the output carry of the sum.

We can use the same technique in the upper level of the reduction tree: instead of checking whether the last sum of fractional shares is greater than q, we securely add $2^w - q$ (we recall that $w = \lceil \log_2 q \rceil$) to one of the fractional shares before summing them, and simply check with SecCarry whether the subsequent sum has a nonzero output carry, as represented in fig. 6.3. This transformation is advantageous, since the addition of $2^w - q$ now needs a lower-order protection: still referring to fig. 6.3, computing this operation over three shares is sufficient for security. In contrast, the SecOverflow_q gadget had to compute it internally over seven shares.

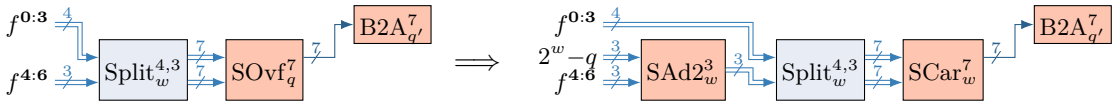


Figure 6.3: Optimization of the upper level of the masked correction tree. The left diagram, extracted from fig. 6.2, is optimized into the right-hand one. Abbreviations Split, SOvf, SCar and SAd2 stand for SplitShares, SecOverflow, SecCarry and SecAddBin respectively.

6.1.8 Comparison with previous works

In this subsection, we compare our solution based on secure integer rescaling with state-of-the art proposals for the masking of Kyber compression. We thus assume that $q' = 2^d$ for some integer $d \leqslant 11$. We start by analyzing the running time of our construction. For simplicity, we focus on a power-of-two number of shares, $n = 2^N$, so that the correction is applied along a well-balanced binary tree. If we optimize the upper level of the tree as discussed in section 6.1.7, and neglect the small running time of the linear gadgets (in particular SecDivMod), SecRescale has a time complexity of

$$T(\mathrm{SecRescale}_{q,q',r}^{2^N}) \approx T(\mathrm{SecAddBin}_w^{2^{N-1}}) + T(\mathrm{SecCarry}_w^{2^N}) + \sum_{i=1}^{N-1} 2^{N-i} T(\mathrm{SecAddOverflow}_q^{2^i}) + \sum_{i=1}^{N} 2^{N-i} T(\mathrm{B2Abit}_{q'}^{2^i})\,.$$

Using $T(\mathrm{SecAddOverflow}_q^{n'}) \lesssim 2T(\mathrm{SecAddBin}_w^{n'})$ [BBE+18, FBR+22], we get

$$T(\mathrm{SecRescale}_{q,q',r}^{2^N}) \lesssim T(\mathrm{SecAddBin}_w^{2^{N-1}}) + T(\mathrm{SecAddBin}_w^{2^N}) + \sum_{i=1}^{N-1} 2^{N-i+1} T(\mathrm{SecAddBin}_w^{2^i}) + \sum_{i=1}^{N} 2^{N-i} T(\mathrm{B2Abit}_{q'}^{2^i})\,,$$

We now determine the asymptotic complexity of our gadget. We use the complexities reported in the literature for the base gadgets we rely upon: $T(\mathrm{SecAddBin}_{2^w}^{n'}) = O(wn'^2)$, as shown by Coron *et al.* [CGV14], and $T(\mathrm{B2Abit}_{q'}^{n'}) = O(n'^2)$ using the gadget of Schneider *et al.* [SPOG19]. We thus obtain

$$\begin{aligned} T(\mathrm{SecRescale}_{q,q',r}^{2^N}) &= O\left(w(2^{N-1})^2 + w(2^N)^2 + \sum_{i=1}^{N-1} 2^{N-i+1} w(2^i)^2 + \sum_{i=1}^{N} 2^{N-i}(2^i)^2\right), \\ &= O\left(wn^2 + wn\sum_{i=1}^{N-1} 2^i + n\sum_{i=1}^{N} 2^i\right), \\ &= O(n^2 \log q)\,. \end{aligned}$$

In comparison, the proposal of Coron *et al.* for high-order masked compression to d bits has a time complexity of $T(\mathrm{HOCompress}_{q,d}) = O(n^2(\log q + \log n))$ [CGMZ23],

which is slightly worse than our result. A refined comparison would need to consider the running time on an actual device.

As an additional advantage, our solution is generic: while Kyber compression, and the masked gadget of Coron *et al.*, always compress to a power-of-two range $q' = 2^d$, this restriction does not exist for SecRescale. Any integer $q' \geqslant 2$ can be chosen, including $q' > q$, in which case the rescaling is more akin to Kyber decompression (see § 2.3.1.3).

A notable difference between SecRescale and HOCompress is that, while our gadget outputs its result as an arithmetic sharing modulo q', that of Coron *et al.* instead outputs a Boolean sharing. The choice between the two gadgets may thus be additionally determined by the type of operations to be performed on the result of compression.

6.1.9 Summing up our contribution

In this section, we have described a novel way of computing Kyber ciphertext compression securely, by viewing it as a more generic integer rescaling. Implementing this rescaling through a share-wise approximation step, and a nonlinear correction step, we obtain a secure algorithm whose data width is independent of the masking order. This allows us to reuse the secure-addition and masking-conversion gadgets that are already employed in other parts of the secure implementation of Kyber.

While our proposal has slightly better asymptotic complexity than achieved in previous works, it would be beneficial to refine this comparison by analyzing the actual runtime of these solutions on a defined software or hardware platform.

6.2 Masked uniform rejection sampling

The learning-with-errors framework requires sampling secrets and errors in a small range centered around zero. While Kyber and its learning-with-rounding equivalent, Saber, use binomial sampling for this purpose, Dilithium instead relies on uniform sampling.

Since this operation is used for the sampling of the private key, it is desirable to protect it thoroughly against side-channel attacks. Indeed, many application need key generation to be performed in the field (where the device performing the operation is vulnerable to side-channel attacks). Furthermore, due to the relatively large size of a private key for Dilithium, this key is sometimes stored as a short (32-byte) seed, which is expanded in memory before every use, thereby running key generation on the same seed each time. This behavior, explicitly allowed by the FIPS standard of ML-DSA [FIP24b], puts a strong constraint on the side-channel security of key generation.

However, as of today, there seems to be no published implementation of masked secret-key sampling for Dilithium and ML-DSA. As a matter of fact, we showed in § 3.6.4.2 that the few implementations of masked uniform sampling that have been proposed do not comply with the specifications of Dilithium (version 3.1) and ML-DSA. Although they sample secret-key polynomials having the appropriate uniform distribution, they do not respect the mapping from the private-key seed specified by the scheme and by the standard.

In this section, we thus describe an implementation of masked uniform sampling that complies with these specifications, while providing provable security at an arbitrary order.

6.2.1 Reminder on Dilithium uniform rejection sampling

The sampling of Dilithium secret-key polynomials, parameterized by $\eta \in \{2, 4\}$ (which we call the half-width of the uniform sampling), consists in drawing a polynomial with coefficients distributed independently and uniformly at random in $[\![-\eta, \eta]\!]$; we called this function ExpandSi in section 2.4. This process is accomplished by repeatedly executing the Sample_η function shown in algorithm 6.5, over a contiguous stream of four-bit words, until all coefficients of the polynomial have been generated in order. One execution of Sample_η samples a single coefficient having the required distribution: to do so, the words of input randomness are successively sent to the $\text{RejectMod}_{2\eta+1}$ function, defined through

$$\text{RejectMod}_5 \colon x \mapsto \begin{cases} x \bmod 5 & \text{if } x < 15\,, \\ \perp & \text{otherwise}\,, \end{cases} \tag{6.6}$$

$$\text{RejectMod}_9 \colon x \mapsto \begin{cases} x & \text{if } x < 9\,, \\ \perp & \text{otherwise}\,, \end{cases} \tag{6.7}$$

until it returns a valid sample. Configured by parameter $H = 2\eta+1$, RejectMod_H checks whether its input x is lower than a bound, and returns the residue of its input modulo H if the bound is satisfied, and a rejection symbol $\perp$ otherwise. The bound is defined as the largest multiple of H strictly below 2^4. When a valid sample z is found, as indicated by RejectMod returning an integer instead of $\perp$, this sample is subtracted from η to get a coefficient within the appropriate range: this is the value returned by Sample_η.

Algorithm 6.5: Sample_η: Dilithium uniform rejection sampling in $[\![-\eta, \eta]\!]$

Parameter: Half-width $\eta \in \{2, 4\}$ of the uniform-sampling range
Input: Stream of 4-bit words $(w_i) \in [\![0, 15]\!]^*$
Output: Coefficient $s \in \mathbb{F}_{q_\text{D}}$, sampled uniformly at random from $[\![-\eta, \eta]\!]$

```
do
    z = RejectMod_{2η+1}(w_i)        // Defined in eqs. (6.6) and (6.7)
    i = i + 1
while z = ⊥
s = η − z
return s
```

We will split the analysis of securely implementing Sample_η in two parts. We first provide a masked implementation of RejectMod_H, which we call SecRejectMod_H. We describe this first protected operation as a dedicated hardware block. This view influences our technical choices in two ways: in terms of security, it raises the need for protection against the physical defects of hardware implementations, namely glitches and transitions; in terms of efficiency, it makes hardware area a primary concern and encourages exploiting hardware parallelism within the computation. We then detail how to proceed with the masking of Sample_η, using $\text{SecRejectMod}_{2\eta+1}$ among other building blocks. For this step, we do not explicitly address hardware over software: we simply presuppose the availability of some elementary gadgets without mandating their specific implementation, and merely show how to assemble them into the desired secure function.

6.2.2 Security model

Since we propose a hardware implementation of a masked operation, we will reuse the framework of Cassiers and Standaert [CS21], already recalled in section 5.2. This framework describes secure digital hardware in terms of *structural circuits*, partitioned into a hierarchy of *structural gadgets* performing secure operations over sharings.

In contrast to our work of chapter 5, the hardware circuit we propose here is not iterative, being instead arranged as a straight pipeline. We will therefore not need the security notion of Output-Probe Isolating Non-Interference (O-PINI) [CS21], and will instead rely on the weaker Probe-Isolating Non-Interference (PINI) notion [CS20], which is sufficient for our purpose. This notion has been previously recalled in definition 6.1 in the standard probing model.

Hardware circuits generally need to consider the robust probing model introduced by Faust *et al.* [FGP+18], since it can suitably describe the hardware defects of *glitches* and *transitions*. Cassiers and Standaert showed how to extend their PINI notion to take these defects into account, by considering the simulatability of *glitch+transition-extended* probes [CS21]. Such probes span glitches, by revealing the values of all the wires which contribute to the value of a net through combinational logic, and transitions, by exposing on each probed wire both its current value and its value at the previous clock cycle.

6.2.3 Logic optimization of reduction modulo five

When the uniform sampling is parameterized by $H = 5$, every accepted sample must be reduced modulo 5 by subtracting from it 0, 5 or 10. Since performing a modular reduction over a Boolean sharing is inconvenient, we first reformulate it in terms of logic operations, and optimize them as much as possible. As a first step, we express both the comparison with the rejection bound and the reduction modulo 5 through two consecutive conditional subtractions, per algorithm 6.6. The input value is first compared with 10, and decreased by this amount if it was greater than 10. A similar conditional subtraction of 5 is then performed, giving the residue of x modulo 5. This final result is returned, unless both conditional subtractions had to be performed, in which case the sample is rejected. As a second step, we will rework the logic equations to minimize the number of multiplications in $\mathbb{F}_2$, since these are nonlinear over a Boolean sharing.

Algorithm 6.6: RejectMod_5: rejection sampling in $[\![0, 14]\!]$ with reduction modulo 5

Input: $x \in [\![0, 15]\!]$
Output: $s \in [\![0, 4]\!] \cup \{\perp\}$ such that $s = \begin{cases} x \bmod 5 & \text{if } x < 15 \\ \perp & \text{otherwise} \end{cases}$

1 **if** $x \geqslant 10$ **then** $g_{10} = 1$ **else** $g_{10} = 0$
2 $y = x - 10 \cdot g_{10}$
3 **if** $y \geqslant 5$ **then** $g_5 = 1$ **else** $g_5 = 0$
4 $z = y - 5 \cdot g_5$
5 **if** $g_{10} g_5 = 1$ **then return** $\perp$ **else return** z

The computation of g_{10} can be expressed in terms of logic operations on the bits of x, denoted $x_0, \ldots, x_3$ (from least to most significant), through

$$g_{10} = x_3 \cdot (\overline{x_2}\,\overline{x_1} \oplus 1)\,,$$

and the bits of $y = x - 10 \cdot g_{10}$ can be determined by writing down the subtraction

		$\overline{x_2}\,\overline{x_1} g_{10}$	$\overline{x_1} g_{10}$		
		x_3	x_2	x_1	x_0
$-$		g_{10}	0	g_{10}	0
$=$		$x_3 \oplus g_{10} \oplus \overline{x_2}\,\overline{x_1} g_{10}$	$x_2 \oplus \overline{x_1} g_{10}$	$x_1 \oplus g_{10}$	x_0
$=$		y_3	y_2	y_1	y_0 .

The computation of g_5 is then

$$g_5 = y_3 \mid y_2(\overline{y_1}\,\overline{y_0} \oplus 1) = x_3\overline{x_1} \oplus x_2 \oplus x_2 x_1 \overline{x_0} \oplus \overline{x_3} x_2 \overline{x_0}\,.$$

We can now compute z through the following subtraction, where ? is a don't-care value:

	?	$\overline{x_0} g_5 (\overline{x_1} \oplus g_{10}) \oplus \overline{x_1} g_{10}$	$\overline{x_0} g_5$	
	x_3	x_2	x_1	x_0
$-$	g_{10}	g_5	g_{10}	g_5
$=$	?	$x_2 \oplus g_5 \oplus \overline{x_0} g_5 (\overline{x_1} \oplus g_{10}) \oplus \overline{x_1} g_{10}$	$x_1 \oplus \overline{x_0} g_5 \oplus g_{10}$	$x_0 \oplus g_5$
$=$		z_2	z_1	z_0 .

In the above, we can simplify $\overline{x_0} g_5 = g_5 \oplus x_0 g_5$ through

$$x_0 g_5 = x_3 \overline{x_1} x_0 \oplus x_2 x_0\,,$$

and since the value of z is unimportant when both g_{10} and g_5 are set, we can assume that $g_{10} g_5 = 0$, *i.e.* $x_3 x_2 x_1 x_0 = 0$, which allows us to write

$$\begin{aligned} z_0 &= x_0 \oplus g_5\,, \\ z_1 &= x_1 \oplus g_5 \oplus x_0 g_5 \oplus g_{10}\,, \\ z_2 &= x_2 \oplus x_0 g_5 \oplus x_2 x_1 \overline{x_0} \oplus x_3 x_2\,. \end{aligned}$$

Furthermore, the rejection condition for $H = 5$ can be expressed as $\mathit{Reject} = x_3 x_2 x_1 x_0$. The computation of the three sample bits y_2, y_1, y_0, as well as the rejection indication, is represented in fig. 6.4. This circuit uses eight two-input AND gates arranged in two layers, a few XOR gates (some of which are merged into the AND gates), and some noninverting buffers to match the latency of the AND gates.

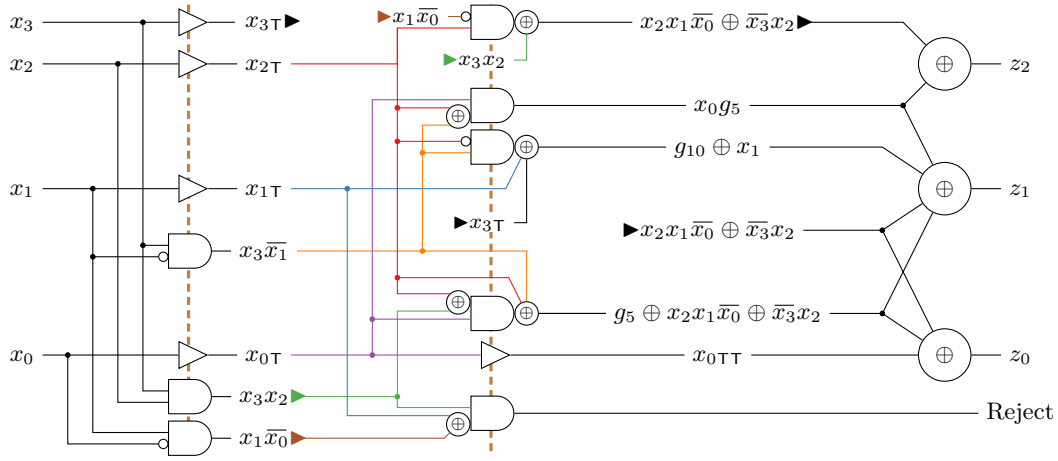


Figure 6.4: Reduction modulo 5 and rejection modulo 15 over four bits. Solid triangles ▶ connect wires having the same label. Wire colors are for better legibility and have no specific meaning. Buffers (shown as $\triangleright$) have the same latency as AND gates; XOR gates have zero latency. Vertical dashed lines delimit pipeline stages. The $_\mathsf{T}$ notation indicates delayed copies of input values.

6.2.4 Secure implementation of rejection and modular reduction

Implementing the circuit of fig. 6.4 using masked gadgets over Boolean shares yields a secure implementation of the $\mathrm{RejectMod}_5$ function, provided the employed gadgets are appropriately composable. In particular, a circuit implemented as the composition of glitch-robust PINI gadgets, whose gates are arranged in a straight pipeline, is provably PINI in the glitch+transition-robust probing model, by [CS21, Lemma 2 and Corollary 2].

6.2.4.1 Supporting gadgets

To achieve this result, we need to introduce a few auxiliary gadgets and prove their security. We start by showing that the HPC3 AND gate of Knichel and Moradi [KM22] can be easily extended to also perform additions after the multiplication in $\mathbb{F}_2$, with the same security and essentially the same cost as the initial gadget. The interest of this new gadget lies in all its inputs having the same latency, which allows us minimize the number of synchronization registers needed for a pipelined implementation of our circuit.

We name this gadget $\mathrm{SecAndXor}_m^n$, n being the number of shares, and m the number of summands to be added to the output of the multiplication. Its definition is shown in algorithm 6.7, where the changes with respect to HPC3 multiplication [KM22] are highlighted in blue. In terms of circuit structure, the gadget is implemented as a straight pipeline, by unrolling all loops and instantiating distinct structural gates for each logic operation.

Proposition 6.5. $\mathrm{SecAndXor}_m^n$ *is a circuit correctly implementing* $x, y, s_0, \ldots, s_{m-1} \mapsto x \circ y \oplus \bigoplus_i s_i$ *over* n *shares, and satisfying PINI security in the glitch+transition-robust probing model.*

Algorithm 6.7: $\mathrm{SecAndXor}_m^n$: combined HPC3 multiplication [KM22] and addition

Input: Boolean sharing $x^\star \in \mathcal{B}^n(\mathbb{F}_2)$ of $x \in \mathbb{F}_2$
Input: Boolean sharing $y^\star \in \mathcal{B}^n(\mathbb{F}_2)$ of $y \in \mathbb{F}_2$
Input: Boolean sharings $s_0^\star, \ldots, s_{m-1}^\star \in \mathcal{B}^n(\mathbb{F}_2)$ of $s_0, \ldots, s_{m-1} \in \mathbb{F}_2$
Output: Boolean sharing $z^\star \in \mathcal{B}^n(\mathbb{F}_2)$ of $z = x \circ y \oplus \bigoplus_{k=0}^{m-1} s_k$

for $\boldsymbol{i} = 0$ to $n-2$ **do**
  for $\boldsymbol{j} = \boldsymbol{i}+1$ to $n-1$ **do**
    $R'_{\boldsymbol{i},\boldsymbol{j}} \xleftarrow{\$} \mathbb{F}_2$; $R''_{\boldsymbol{i},\boldsymbol{j}} \xleftarrow{\$} \mathbb{F}_2$
    $R'_{\boldsymbol{j},\boldsymbol{i}} = R'_{\boldsymbol{i},\boldsymbol{j}}$
    $R''_{\boldsymbol{j},\boldsymbol{i}} = R''_{\boldsymbol{i},\boldsymbol{j}}$
  end
end
for $\boldsymbol{i} = 0$ to $n-1$ **do**
  for $\boldsymbol{j} = 0$ to $n-1$ **do**
    if $\boldsymbol{i} \neq \boldsymbol{j}$ **then**
      $U'_{\boldsymbol{i},\boldsymbol{j}} = \mathrm{Reg}[y^{\boldsymbol{j}} \oplus R'_{\boldsymbol{i},\boldsymbol{j}}]$
      $U''_{\boldsymbol{i},\boldsymbol{j}} = \mathrm{Reg}[\overline{x^{\boldsymbol{i}}} \circ R'_{\boldsymbol{i},\boldsymbol{j}} \oplus R''_{\boldsymbol{i},\boldsymbol{j}}]$
      $C_{\boldsymbol{i},\boldsymbol{j}} = \mathrm{Reg}[x^{\boldsymbol{i}}] \circ U'_{\boldsymbol{i},\boldsymbol{j}} \oplus U''_{\boldsymbol{i},\boldsymbol{j}}$
    end
  end
end
for $\boldsymbol{i} = 0$ to $n-1$ **do**
  $z^{\boldsymbol{i}} = \mathrm{Reg}[x^{\boldsymbol{i}} \circ y^{\boldsymbol{i}} \oplus \bigoplus_{k=0}^{m-1} s_k^{\boldsymbol{i}}] \oplus \bigoplus_{\boldsymbol{j}=0}^{n-1} C_{\boldsymbol{i},\boldsymbol{j}}$ *// The first $\bigoplus$ is the only difference from [KM22]*
end

Proof. Correctness. Following [KM22], we can discard all registers in algorithm 6.7, thereby obtaining $z = \bigoplus_{\boldsymbol{i}} z^{\boldsymbol{i}} = (\bigoplus_{\boldsymbol{i}} x^{\boldsymbol{i}}) \circ (\bigoplus_{\boldsymbol{i}} y^{\boldsymbol{i}}) \oplus \bigoplus_k \bigoplus_{\boldsymbol{i}} s_k^{\boldsymbol{i}} = x \circ y \oplus \bigoplus_k s_k$.

Security. We adapt the proof of [KM22, Theorem 3.1] by adjusting the notation of cases I–III and modifying case IV from

> *A glitch-extended output probe*
>
> $$P_{Z_i} = [X^i, Y^i] \cup \{ \bigcup_{0 \leqslant j \leqslant d, i \neq j} C_{ij} \}$$
>
> *can be simulated by X^i and Y^i and drawing d times $R'_{ij} \xleftarrow{\$} \mathbb{F}_2$ and $R''_{ij} \xleftarrow{\$} \mathbb{F}_2$ (to simulate C_{ij}).*

into

> A glitch-extended output probe
>
> $$P_{z_{\boldsymbol{i}}} = \{x^{\boldsymbol{i}}, y^{\boldsymbol{i}}, s_0^{\boldsymbol{i}}, \ldots, s_{k-1}^{\boldsymbol{i}}\} \cup \bigcup_{0 \leqslant \boldsymbol{j} < n,\ \boldsymbol{i} \neq \boldsymbol{j}} \{C_{\boldsymbol{i},\boldsymbol{j}}\}$$
>
> can be simulated by $x^{\boldsymbol{i}}$, $y^{\boldsymbol{i}}$ and all $s_k^{\boldsymbol{i}}$ for $0 \leqslant k < m$, and drawing d times $R'_{\boldsymbol{i},\boldsymbol{j}} \xleftarrow{\$} \mathbb{F}_2$ and $R''_{\boldsymbol{i},\boldsymbol{j}} \xleftarrow{\$} \mathbb{F}_2$ (to simulate $C_{\boldsymbol{i},\boldsymbol{j}}$).

Applying this adjustment allows us to prove that SecAndXor_m^n is glitch-robust PINI. Since the structural circuit implementing it is pipeline, by applying [CS21, Lemma 2], we can conclude that the gadget is glitch+transition-robust PINI. □

We note that the SecAndXor_m^n has the same cost as the HPC3 multiplication gate, in terms of the number of registers ($2n^2$ for n shares) and the number of refresh bits required ($n^2 - n$ for n shares).

In addition to this, we use the unmodified HPC3 AND gate, denoted SecAnd, that is likewise glitch+transition-robust PINI. Finally, share-wise addition over $\mathbb{F}_2$ and share-wise multiplexing (configured by a public parameter), which are glitch+transition-robust PINI by [CS21, Proposition 1], will be employed.

6.2.4.2 Composition

Using the aforementioned gadgets, we can implement the RejectMod_5 function securely over Boolean shares, as expressed in algorithm 6.8. This algorithm reproduces the circuit diagram of fig. 6.4 using PINI gadgets. The computation introduces a latency of two cycles, but since it is fully pipelined, one input sample can be processed at each clock cycle. In the algorithm listing, we denote by $1^\star$ a fixed sharing of 1, *e.g.* $(1, 0, \ldots, 0)$.

Algorithm 6.8: SecRejectMod_H^n: Dilithium secret-key sampling over Boolean shares

Parameter: Half-width $H \in \{5, 9\}$ of the uniform-sampling range, number of shares n
Input: Boolean shares of four-bit input word, $x^\star \in \mathcal{B}^n(\mathbb{F}_2^4)$, having bits $x_0^\star, \ldots, x_3^\star$
Output: Rejection indication, $\mathit{Reject} \in \mathcal{B}(\mathbb{b})$ such that $\bigoplus_{\boldsymbol{i}} \mathit{Reject}^{\boldsymbol{i}} = [\text{RejectMod}_H(\bigoplus_{\boldsymbol{i}} w^{\boldsymbol{i}}) = \perp]$
Output: Boolean shares of the output sample, $s^\star \in \mathcal{B}^n(\mathbb{F}_2^4)$, having bits $z_0^\star, \ldots, z_3^\star$, such that
$\bigoplus_{\boldsymbol{i}} \mathit{Reject}^{\boldsymbol{i}} = 0 \implies \bigoplus_{\boldsymbol{i}} s^{\boldsymbol{i}} = \text{RejectMod}_H(\bigoplus_{\boldsymbol{i}} w^{\boldsymbol{i}})$

	// Value of variables (? is don't care) when	$H = 5$	$H = 9$
	// First stage of the pipeline — note that all gadgets have n shares		
1	$a^\star = \text{SecAnd}(x_3^\star, x_1^\star \oplus 1^\star)$ . . . . . *//*	$x_3\overline{x_1}$	$x_3\overline{x_1}$
2	$b^\star = \text{SecAnd}(x_3^\star, x_2^\star \oplus (1^\star \textbf{ if } H \neq 5))$ . . . . . *//*	x_3x_2	$x_3\overline{x_2}$
3	$c^\star = \text{SecAnd}(x_1^\star \oplus (1^\star \textbf{ if } H \neq 5), x_0^\star \oplus 1^\star)$ . . . . . *//*	$x_1\overline{x_0}$	$\overline{x_1}\overline{x_0}$
4	$x_{0\mathsf{T}}^\star = \text{Reg}[x_0^\star];\quad x_{1\mathsf{T}}^\star = \text{Reg}[x_1^\star];\quad x_{2\mathsf{T}}^\star = \text{Reg}[x_2^\star];\quad x_{3\mathsf{T}}^\star = \text{Reg}[x_3^\star]$. *//*	*Pipeline registers*	
	// Second stage		
5	$d^\star = \text{SecAndXor}_1(1^\star \oplus (c^\star \textbf{ if } H = 2),\ x_{2\mathsf{T}}^\star,\ (b^\star \textbf{ if } H = 5))$ . . *//*	$x_2x_1\overline{x_0} \oplus \overline{x_3}x_2$	x_2
6	$e^\star = \text{SecAnd}(x_{0\mathsf{T}}^\star,\ x_{2\mathsf{T}}^\star \oplus a^\star)$ . . . . . *//*	x_0g_5	?
7	$f^\star = \text{SecAndXor}_2(x_{2\mathsf{T}}^\star \oplus 1^\star,\ (a^\star \textbf{ if } H = 5),\ x_{1\mathsf{T}}^\star,\ (x_{3\mathsf{T}}^\star \textbf{ if } H = 5))$ *//*	$g_{10} \oplus x_1$	x_1
8	$g^\star = \text{SecAndXor}_2((x_{0\mathsf{T}}^\star \textbf{ if } H{=}2 \textbf{ else } 1^\star),\ x_{2\mathsf{T}}^\star \oplus b^\star,\ (a^\star \textbf{ if } H{=}2),\ x_{2\mathsf{T}}^\star)$ *//*	$g_5 \oplus x_2x_1\overline{x_0} \oplus \overline{x_3}x_2$	$x_3\overline{x_2}$
9	$\mathit{Reject}^\star = \text{SecAndXor}_1(c^\star \oplus (x_{1\mathsf{T}}^\star \textbf{ if } H = 5),\ b^\star,\ (x_{3\mathsf{T}}^\star \textbf{ if } H \neq 5))$ *//*	$x_3x_2x_1x_0$	$x_3\overline{x_2}\overline{x_1}\overline{x_0} \oplus x_3$
10	$x_{0\mathsf{TT}}^\star = \text{Reg}[x_{0\mathsf{T}}^\star]$ . . . . . *//*	*Pipeline register*	
	// Third stage (not closed by a register barrier)		
11	$z_3^\star = (g^\star \textbf{ if } H \neq 5)$ . . . . . *//*	0	$x_3\overline{x_2}$ (= x_3 when $\neg \mathit{Reject}$)
12	$z_2^\star = d^\star \oplus (e^\star \textbf{ if } H = 5)$ . . . . . *//*	$\overline{x_3}x_2 \oplus x_0g_5 \oplus x_2x_1\overline{x_0}$	x_2
13	$z_1^\star = f^\star \oplus (e^\star \oplus g^\star \oplus d^\star \textbf{ if } H = 5)$ . . . . . *//*	$x_1 \oplus \overline{x_0}g_5 \oplus g_{10}$	x_1
14	$z_0^\star = x_{0\mathsf{TT}}^\star \oplus (g^\star \oplus d^\star \textbf{ if } H = 5)$ . . . . . *//*	$x_0 \oplus g_5$	x_0
15	**return** $\mathit{Reject}^\star, z^\star$		

In its other parameterization ($H = 9$), the RejectMod_9 function performs a single logic computation, to check the rejection condition $x \geqslant 9$, which we can also express as $x_3\overline{x_2}\overline{x_1}\overline{x_0} \oplus x_3 = 1$. The input sample x does not need to be altered in this case. We can thus additionally implement rejection sampling for this other value of the parameter with no additional nonlinear gadget, with the help of a few share-wise multiplexers that slightly change the operands of the SecAnd and SecAndXor gadgets. These multiplexers are represented in algorithm 6.8 in the form ($T^\star$ **if** *predicate* **else** $F^\star$), which equals $T^\star$ if *predicate* is true and $F^\star$ otherwise. The abbreviated notation ($T^\star$ **if** *predicate*) stands for ($T^\star$ **if** *predicate* **else** $0^\star$) where $0^\star$ is the all-zero sharing.

The correctness of the gadget in both configurations can be checked with the help of the right-hand comments in algorithm 6.8, which keep track of the value of the variables at each line, depending on the value of parameter H. Regarding output z_3 when $H = 9$, note that accepted samples notably satisfy $x_3x_2 = 0$, so $x_3 = x_3\overline{x_2}$ is correct as their most significant bit, and the value of this bit does not matter for rejected samples. This slight optimization does not thus preclude correctness. It is therefore clear that *Reject* always computes the appropriate rejection criterion, and that when *Reject* $= 0$, the output sharings z_0 to z_3 correspond to the output value of RejectMod_H.

6.2.4.3 Security

In terms of circuit structure, we implement SecRejectMod by instantiating distinct structural gadgets for each operation in algorithm 6.8, and connecting them accordingly. We thus obtain a straight pipeline without any loop, in which each structural gadget is used exactly once for each successive input sharing. In the terminology of Cassiers and Standaert, the composing gadgets of SecRejectMod have no *adjacent executions* [CS21]. PINI gadgets being securely composable in this situation, we can effortlessly prove the security of our construction, as witnessed by the following theorem.

Theorem 6.3. *For any $n \geqslant 2$, the SecRejectMod_H^n gadget over n shares satisties glitch+transition-robust PINI security.*

Proof. We first highlight that SecRejectMod_H^n is the composition of four SecAnd, three SecAndXor_2, and one SecAndXor_1 gadget, as well as share-wise multiplexing and share-wise addition gadgets. Each of the composing structural gadgets has no adjacent executions. Since the composing gadgets are glitch+transition-robust PINI by [KM22, Theorem 3.1], by proposition 6.5, and by the share-isolating nature of multiplexing and addition, we conclude from [CS21, Corollary 2] that SecRejectMod_H^n is glitch+transition-robust PINI. □

6.2.4.4 Performance and cost

As discussed earlier, SecRejectMod has a latency of two clock cycles, but can process one input sample per cycle thanks to its pipelining (the output rate, however, is lower due to the rejection of 1 out of 16 samples for $H = 5$, and 7 out of 16 for $H = 9$).

We can estimate the area usage of the whole composite gadget by considering the area occupied by the nonlinear composing gadgets (whose size grows quadratically with the

number of shares), and the number of additional flip-flops (proportional to the number of shares). As argued in § 6.2.4.1, the $\mathrm{SecAndXor}_m$ uses the same number of registers as the HPC3 multiplication gate [KM22] over the same number of shares, and occupies negligibly more area than it. The area of SecRejectMod is thus approximately

$$\begin{aligned}\mathrm{Area}(\mathrm{SecRejectMod}_H^n) &\approx 8\,\mathrm{Area}(\text{HPC3 SecAnd}^n) + 5n\,\mathrm{Area}(\text{flip-flop})\,,\\ &\approx (16n^2 + 5n)\,\mathrm{Area}(\text{flip-flop})\,,\end{aligned}$$

where the second line assumes that flip-flops occupy the major part of the overall area[3].

Thanks to the use of PINI gadgets, no refreshing is needed beyond that which is already included in the nonlinear gadgets. Consequently, our proposal needs $8 \times n(n-1)$ fresh random bits for each input sample.

6.2.5 Conversion to arithmetic shares and offsetting of the uniform sample

The $\mathrm{SecRejectMod}_H$ gadget we just introduced outputs an integer uniformly distributed in $[\![0, H-1]\!]$ (or a rejection symbol). We now need to shift this sample into the range $[\![-\eta, \eta]\!]$ (with $2\eta + 1 = H$) in accordance with algorithm 6.5. This offsetting is a linear operation over arithmetic shares, and the sampled polynomial will undergo further arithmetic operations throughout the Dilithium scheme. Consequently, we start by converting the output of SecRejectMod from Boolean to arithmetic shares, and apply the offsetting in the arithmetic domain. This process is formalized in algorithm 6.9, which starts by repeatedly attempting secure rejection sampling SecRejectMod, until a valid sample is found: this event is signaled by obtaining a zero value when unmasking the *Reject* signal. We perform this check through the SecUnmask gadget of Azouaoui *et al.*, which consists in refreshing its input sharing then recombining the shares, and which was shown to satisfy PINI security [ABC+23]. We rely on the rejection or acceptance of the input samples not being a sensitive information, in accordance with the specification of Dilithium and with many other works [BDK+21, MGTF19, ABC+23, CGTZ23][4].

Having obtained a valid sample $z^\star$, we convert its shares from the Boolean to the arithmetic domain, modulo Dilithium's modulus q_D. For this purpose, the few-bit masking conversion of Schneider *et al.* [SPOG19], recalled in § 3.6.2.2, is especially appropriate since $z^\star$ is expressed over three or four bits (respectively when $\eta = 2$ and $\eta = 4$). The offsetting $z \mapsto (\eta - z) \bmod q_\mathrm{D}$ is then applied as a share-isolating operation, before returning the obtained sample as arithmetic shares modulo q_D.

We highlight that it may be appropriate to split algorithm 6.9 in two separate steps: the rejection sampling in lines 1 to 4 on one hand, and the conversion and offsetting in lines 5 to 6 on the other hand. Indeed, the sampling of the secret polynomials happens during Dilithium key generation. Unless the private key is expanded from its seed

[3] This is a reasonable (albeit rough) approximation for an application-specific integrated circuit (ASIC), in which each flip-flop typically occupies the same area as 3 to 10 two-input logic gates.

[4] The input samples are indeed generated by a secure pseudorandom extendable output function, which guarantees that successive input samples are not correlated in any distinguishable way. Learning that a sample is rejected does not thus provide any information on the values of the accepted samples.

Algorithm 6.9: $\mathrm{SecSample}_{\eta}^{n}$: Dilithium masked uniform rejection sampling in $[\![-\eta, \eta]\!]$

Parameter: Half-width $\eta \in \{2, 4\}$ of the uniform-sampling range, number of shares n
Input: Stream of Boolean sharings of 4-bit words $(w_i^\star) \in \mathcal{B}^n([\![0, 15]\!])^*$
Output: Masked coefficient $s^\star \in \mathcal{A}_{q_\mathrm{D}}^n(\mathbb{F}_{q_\mathrm{D}})$, sampled uniformly at random from $[\![-\eta, \eta]\!]$, as arithmetic shares modulo q_D

1 **do**
2 $\quad$ $\mathit{Reject}^\star, z^\star = \mathrm{SecRejectMod}_{2\eta+1}^{n}(w_i^\star)$
3 $\quad$ $i = i + 1$
4 **while** $\mathrm{SecBoolUnmask}^n(\mathit{Reject}^\star) = 1$
5 $t^\star = \mathrm{B2A}_{q_\mathrm{D}}^{n}(z^\star)$
6 $s^\star = (\eta - t^{\mathbf{0}},\ -t^{\mathbf{1}},\ \ldots,\ -t^{\boldsymbol{n-1}}) \bmod q_\mathrm{D}$
7 **return** $s^\star$

for each signature generation, the key-generation operation must then store the secret polynomials in the packed private key. This would waste a large amount of storage space if the arithmetic shares of the secret polynomials were stored, since arithmetic shares modulo q_D occupy 23 bits per polynomial coefficient and per share, although each coefficient only has 5 or 9 possible values when unmasked. It is thus more opportune to store the coefficients as Boolean shares, since the required storage thus decreases to 3 or 4 bits per coefficient and per share[5]. In this case, key generation will pack the Boolean shares obtained after the rejection-sampling step, and proceed with the step of conversion and offsetting before securely computing the noisy product $\mathbf{A} \times \mathbf{s}_1 + \mathbf{s}_2$ at line 15 of algorithm 2.16. Then, signature generation will unpack the Boolean shares of $\mathbf{s}_1$ and $\mathbf{s}_2$ from the packed private key, and solely apply the step of conversion and offsetting before proceeding with the rest of the signature-generation process (see algorithm 2.17).

6.2.6 Summary

The construction described in this section provides a secure implementation of the sampling of Dilithium secret polynomials. This masked gadget is fully compliant with the specifications of Dilithium and ML-DSA: this is in contrast to previous works, all proposing to mask this operation in a way that is, at best, interoperable with the specification, but that fails to satisfy the mapping from a specific seed to the corresponding private key.

For the first part of our proposal, which implements the rejection sampling and optional modular reduction giving secret coefficients with the appropriate span, we provide a compact hardware implementation employing as little as eight secure AND gates (together with some linear logic), and accomplishing its function in a fully pipelined way with two cycles of latency. The second part mainly consists of an efficient masking conversion from the state of the art.

[5] As an illustration, storing the two secret polynomials $\mathbf{s}_1$ and $\mathbf{s}_2$ over two shares, for the lowest security level of Dilithium, would require 11.5 kB with arithmetic shares, but only 1.5 kB with Boolean shares.

As demonstrated in section 3.6.4, this was the last operation of Dilithium for which an appropriate masked implementation lacked in the literature: by providing a conclusive solution to this problem, we thus close the gap and make the full protection of Dilithium against vertical attacks achievable from the state of the art.

6.3 Shuffling

We highlighted in chapter 3 that masking is not an appropriate countermeasure against the single-trace attacks that have a high success rate, since attacking each share of a secret separately will still lead to a successful attack in this situation. Consequently, additional countermeasures are needed to defeat these specific attacks. This is the case, in particular, for Soft Analytical Side-Channel Attacks (SASCA) [VGS14], developed in section 3.2, for which *shuffling* has been frequently suggested as an effective countermeasure [VGS14, PPM17, PP19b, KPP20]. The principle of this protection is to randomly change the order in which independent computations are performed by the scheme. Already alluded to in the seminal paper of Kocher *et al.* on differential power analysis [KJJ99], it was then explicitly described by Herbst *et al.* [HOM06] and Rivain *et al.* [RPD09], among others. Intuitively, shuffling makes the task of the attackers harder by obscuring the association between points of interest in the side-channel leakage and the corresponding secret datum. While this effect, taken on its own, is unlikely to defeat a determined attacker, it can enable the use of other countermeasures such as masking. Indeed, the strong theoretical and practical security arguments for masking are only applicable to implementations that are already sufficiently noisy [PR13].

6.3.1 Theoretical and practical bounds of effectiveness

An attack that was proposed early on against shuffled implementations, or implementations protected through other kinds of temporal randomization, is the *integrated* Differential Power Analysis (DPA) technique, which consists in averaging the leakage over several time samples before performing a usual DPA attack [CCD00, TH08, RPD09]. The behavior noted in these works is that spreading the leakage related to a sensitive variable over t time samples can, at best, decrease the correlation between the secret and the side-channel measurements by a factor of $\sqrt{t}$.

This statement has been confirmed both theoretically and practically by Veyrat-Charvillon *et al.* [VMKS12]. However, they also note that this is only an upper bound on the effectiveness of the countermeasure: indeed, attacks that exploit the specifics of the shuffling method in use can obtain much better performance. They give as an example the shuffled computation of an AES round, involving 16 S-box computations that can be shuffled in any order. They show that the averaging performed by the integrated DPA attack decreases by more than an order of magnitude the mutual information between the secrets and their side-channel leakage, with respect to the case of a standard DPA attack on an unshuffled implementation. However, they also demonstrate that depending on the shuffling method and on the leakage of the permutation itself, several strategies allow the attacker to obtain much finer results up to moderately high noise conditions.

In particular, they highlight that shuffling based on *random start index*, which cyclically shifts the order of operations by a random amount, has negligible effect on the attack cost in low to moderate noise conditions (for noise having a standard deviation of up to 2 when measuring the Hamming weight of 8-bit secret bytes and of 4-bit permuted indexes). When, instead, a full permutation of the 16 S-boxes is enforced, the attacker can no longer enumerate all possible permutations and has to use heuristic methods to only retain the most likely ones, thereby being significantly more affected by noise [VMKS12].

6.3.2 Previous implementations and attacks of shuffling

Since lattice-based cryptography involves operating on large sequences of integers (either in the form of integer vectors or of polynomials over a finite field), in an order that does not matter for most operations, it constitutes a convenient opportunity to implement shuffling as a side-channel mitigation.

We listed in section 3.8 a few of these instances of shuffling in the literature on lattice-based cryptography. Jati *et al.* [JGCS24] use shuffling and other kinds of temporal and spatial randomization as sole side-channel countermeasures. In contrast, Zijlstra *et al.* [ZBT19] employ shuffling in combination with masking, blinding and shifting, to complement the DPA protection of the last three with the effectivenes of the first against single-trace attacks. Their security analysis, however, merely counts the additional entropy added by shuffling, with no consideration of how the countermeasure would fare against a dedicated attack.

As a matter of fact, while the authors of several side-channel attacks against lattice-based cryptography have cited shuffling as a countermeasure to their attacks (notably for the above-mentioned SASCA [VGS14, PPM17, PP19b, KPP20]), it has been shown that this protection can be defeated, at least in some cases, by adapting the attack.

As a first example, Hermelink *et al.* [HSST23] show how to adapt single-trace attacks based on belief-propagation to cope with shuffling, at the cost of reduced noise tolerance and increased runtime. They attack a simulated implementation of the Number-Theoretic Transform (NTT, see § 2.3.1.1) for Kyber, in which they shuffle the computation of the butterfly operations within each layer. When the butterflies are computed in order but each has its two inputs randomly swapped (the authors call this *fine shuffling*), their attack is nearly as effective as without shuffling, although its noise tolerance is slightly decreased. When the butterflies are shuffled within groups sharing an identical *twiddle factor* (a situation called *coarse in-block shuffling*), they are still able to recover the secret polynomial perfectly provided the measurement noise is sufficiently low.

In a second setting, Ngo *et al.* [NDJ21] demonstrate how to make a message-recovery attack agnostic to shuffling, by only recovering the Hamming weight of the messages, and exploiting the ciphertext malleability of schemes based on LWE or LWR (see section 3.5.1) to determine the position of their set bits. When converting their attack into key recovery, they need 62 k traces to successfully recover a private-key in the intermediate security level of Saber. In a followup work, Backlund *et al.* [BNGD23] improve the attack by directly recovering the value of two message bits at known positions, namely 0 and 255,

and rotating the chosen ciphertexts they employ to reveal all private coefficients through these two indexes. To do so, they explicitly attack the shuffling permutation (generated through the Fisher-Yates algorithm) to determine when the coefficients corresponding to these two indexes are processed. Through this improvement, they are able to divide by 13 the number of traces required for successful key recovery.

6.3.3 Problem statement

We want to put in place a shuffling countermeasure to complement masking by increasing the amount of noise in the side-channel leakage, and by disabling some single-trace attacks that are little affected by masking alone, but whose effectiveness deteriorates in the presence of shuffling. This shuffling should be applicable to all the operations of Kyber and Dilithium that are not inherently monotonic. In particular, it should be applicable to polynomial arithmetic and to masking operations. Sampling operations, instead, are out of scope since they need to be performed sequentially, and in the case of rejection sampling, do not even consume a predefined amount of data.

The computation of the NTT as it is described in algorithm 2.1 requires to operate, for each layer, on $128 = 2^7$ pairs, which can be processed in any order. On the other hand, the addition of two polynomials operates on $256 = 2^8$ pairs of coefficients, where each pair is formed of a coefficient from the first polynomial, and the corresponding coefficient of the second polynomial. Furthermore, when these operations are performed on vectors of polynomials, the processing of each polynomial does not need to be contiguous: intermingling the processing of all the coefficients over the whole polynomial vector increases the potential for noise amplification. The maximum vector size is reached by the highest security level of Dilithium, which employs vectors of size 8. The number of operations to be shuffled for the two schemes of interest thus ranges from 2^7 to 2^{11}.

Due to some constraints on the memory layout of our hardware implementation, we need to process every group or four consecutive polynomial coefficients as an atomic block, which cannot be separated by shuffling. Consequently, the shuffling must operate on a quarter of the above-mentioned ranges to satisfy this property. To simplify our analysis, we will also not consider vectors whose size is not a power of two. We therefore need a way to shuffle a power-of-two number of steps between 2^5 and 2^9, and we will also extend our analysis to 2^{10}.

We note that when performing linear operations on masked data, the same operation is applied to each share. It may thus seem reasonable to consider the k shares of a polynomial vector as a single vector, k times larger, and to shuffle all the coefficients together, irrespective of the share they belong to. However, unrestricted shuffling in this situation may induce two shares of a single coefficient to be processed sequentially. When this happens, the security order may be decreased since the transition from one share to the other reveals the Hamming distance between them, while processing two unrelated shares sequentially gives a random Hamming distance. Therefore, in the case of first-order masking, the average Hamming distance over all transitions will be slightly biased toward the average Hamming weight of the unmasked coefficients. In particular, this would probably allow an attacker to distinguish a polynomial having all or most

coefficients null, from a polynomial having uniformly random coefficients (as involved in decryption-failure attacks, described in section 3.3). We therefore refrain from employing such a simultaneous shuffling of polynomial shares.

We thus seek a method to efficiently shuffle ranges of the form $[\![0, 2^N - 1]\!] \cong \mathbb{b}^N$, where $5 \leqslant N \leqslant 10$. In this section, symbol N will consistently define in this way the range to be shuffled. We also denote by $N_{\max} = 10$ its maximum value of interest.

6.3.4 Desired properties of shuffling for lattice-based cryptography

It seems that the attack of Ngo *et al.* [NDJ21] that we mentioned is intrinsically immune to shuffling, since it is only concerned with recovering the distribution of the message bits, rather than determining the position of these values. Therefore, improving the shuffling method is unlikely to have any positive effect against the attack.

In contrast, the attack of Hermelink *et al.* [HSST23] and that of Backlund *et al.* [BNGD23] explicitly work around the flaws of the specific shuffling in use. In the first case, the authors use that the shuffling operates independently on small groups of butterfly operations, both in the case of *fine shuffling*, and in the case of computing the outermost NTT layers for *coarse in-block shuffling*. Consequently, they can reasonably enumerate permutations, average the measurements over the blocks of permuted butterflies, or match corresponding values between NTT layers. In the second case, Backlund *et al.* rely on the permutation being computed in a pre-processing step, and then being accessed a second time during the computation of the polynomial operation: the leakage on the permutation itself allows them to reliably recover two coefficient indexes.

In order to avoid these weaknesses and similar ones, we seek a shuffling method that performs well on all the following security aspects.

Entropy The generation of the permutation should have sufficient entropy to make enumeration impractical during the analysis phase of a side-channel attack. There is no need to add a very large entropy in this way, since enumeration is likely to be a worst-case handling of the countermeasure by the attacker: we consider 30 bits of entropy as a reasonable starting point. As a reference, the best performance figures for a belief-propagation attack against Kyber NTT mention a running time of at least four minutes over a single CPU core [PP19b]. If the attack had to be run 2^{30} times to enumerate the shuffling permutations of a single layer, even parallelizing it over 2^{10} cores would bring the running time to two years, which seems impractical unless optimizations by more than an order of magnitude are possible.

Marginal uniformity Each index, taken independently, should be permuted almost uniformly over the whole range of output indexes, so that the countermeasure has the best possible effect against integrated attacks.

Joint uniformity Subsets of the permuted indexes should have a distribution close to uniform among the arrangements of the output indexes, up to sufficiently large cardinalities.

Low leakage The generation and use of the permutation should leak as little as possible to ensure that an attacker cannot target the permutation itself (as exemplified by the above-mentioned attack of Backlund *et al.* [BNGD23]). In particular, generating the permutation on the fly during its use is desirable: it avoids accessing it twice, and it should complicate the separation of the side-channel leakages of the permutation itself and of the permuted values.

Randomness efficiency The generation of the permutation should use its input randomness efficiently to facilitate a reseeding after each shuffled operation. It is desirable, for instance, to shuffle each layer independently during an NTT computation, and to shuffle independently the linear computations on different shares of a masked polynomial.

Furthermore, some practical characteristics are desired to facilitate the integration of the solution.

Small area and low latency When implemented in hardware, the generation of the permutation should occupy a small area. Computing the permutation should not require more than a few cycles to avoid increasing the overall latency of shuffled operations by a significant amount.

Runtime configurability The solution should be able to generate permutations of various ranges at runtime, to best support different polynomial operations and operations on polynomial vectors of various sizes. We restrict our analysis to shuffling power-of-two ranges from 2^5 to 2^{10}.

6.3.5 Pseudorandom permutation based on ad-hoc block cipher

It is remarkable that the above-mentioned security aspects are reminiscent of some of the security properties of block ciphers, which exactly solve our problem. Given a block cipher E_K, indexed by key K, that operates on blocks of N bits, and having chosen K at random, $(E_K(i))_{i\in\mathbb{b}^N}$ is a pseudorandom permutation of $\mathbb{b}^N$. K can be considered as the seed of the permutation, and different K will give unrelated permutations. Not all the security properties of a cryptographically secure block cipher are required however. For instance, a strong block cipher should not allow an adversary to determine K from a large number of $(i, E_K(i))$ pairs, but for the purpose of shuffling this property is moot since determining many $(i, E_K(i))$ pairs is precisely the objective of the attacker.

We thus propose a pseudorandom-permutation generator based on a scaled-down block cipher built specifically for our purpose. Since we are working on a small block size, the block cipher simply iterates a substitution box (we will detail the requirements on this substitution function in the next subsection), as illustrated in fig. 6.5. This structure is called a key-alternating block cipher [DR02, Can16]. Given as input a list of *round keys* $K_0, \ldots, K_m$, a monotonic input range is transformed into a permuted range by applying the substitution box m times and adding a round key between two successive applications of the S-box, as well as at the beginning and at the end.

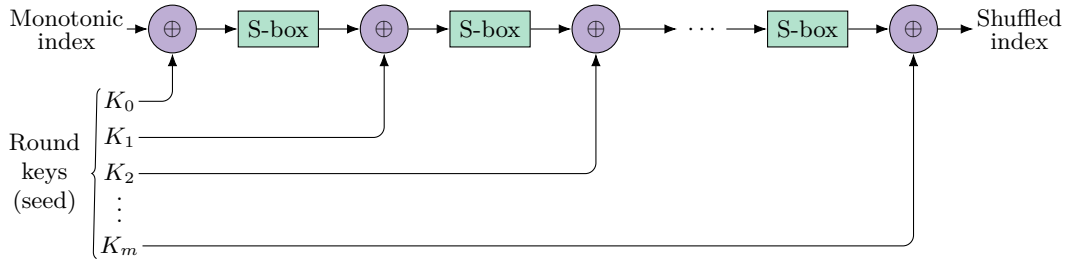


Figure 6.5: Pseudorandom permutation based on a key-alternating block cipher.

This key-alternating construction is used, for instance, in AES [AES01, DR02], with a major difference with respect to our proposal: the round permutation of AES is defined as the parallel application of a substitution function over 8-bit words, followed by two linear transformations (termed ShiftRows and MixColumns), which provide *diffusion* among these words. However, in our proposal, the block size between 5 and 10 bits is too small to be split in several words; it would not even be possible to split the block into equal-size words for prime block sizes. This explains why our round function will be comprised solely of the substitution function and of the round-key addition, without an additional linear diffusion layer.

Formally, our block cipher can be defined inductively through the following equations, in which m is the number of rounds, $K_0, \ldots, K_m \in \mathbb{b}^N$ are the round keys, and $S_N \colon \mathbb{b}^N \to \mathbb{b}^N$ is an N-bit substitution box.

$$E_{K_0} \colon x \mapsto x \oplus K_0$$
$$\text{For } m > 0,\ E_{K_0,\ldots,K_m} \colon x \mapsto S_N(E_{K_0,\ldots,K_{m-1}}(x)) \oplus K_m$$

Correctness For any integer $m \geqslant 0$ and $K_0, \ldots, K_m \in \mathbb{b}^N$, $E_{K_0,\ldots,K_m}$ is a permutation of $\mathbb{b}^N$ if and only if S_N is a permutation.

We note that when adapting this construction for different values of N, the only difficulty lies in finding a family of good substitution functions (S_N). The key-addition steps can trivially handle different index widths by supporting wider round keys (having $N_{\max}$ bits) and masking out their superfluous bits (above index N).

6.3.6 Variable-size substitution box

To get a substitution box providing good security properties, we loosely follow the design decisions having led to the Rijndael S-box [DR02], in turn based on the work of Nyberg [Nyb94] on differentially uniform mappings. In addition to efficient computability, Nyberg lists the following three desired properties for a good substitution box S over $\mathbb{F}_{2^N}$:

(*i*) large distance from linear functions, characterized by a high nonlinearity[6] $\mathcal{N}(S)$;

(*ii*) high nonlinear order, *i.e.* the degrees of the output bit functions are large; and

[6] Canteaut [Can16] instead consider the complementary value, linearity $\mathcal{L}(S) = 2^{N-1} - \frac{1}{2}\mathcal{N}(S)$.

(*iii*) resistance against differential cryptanalysis, which is provided by a low differential-uniformity value $\delta(S)$.

Canteaut [Can16] provides some additional information on two of these criteria. Criterion (*i*) is best satisfied by *almost bent* (AB) functions, which reach the optimal nonlinearity value $\mathcal{N}(S) = 2^{N-1} - 2^{(N-1)/2}$. However, AB functions only exist for odd N: for even N, the highest known nonlinearity value is $\mathcal{N}(S) = 2^{N-1} - 2^{N/2}$. Criterion (*iii*) is best met by *almost perfectly nonlinear* (APN) functions, whose differential uniformity is 2. Since such functions do not exist on every binary field, the goal is again to get as close to them as possible. Canteaut notes that AB permutations are also APN.

Nyberg [Nyb94] shows that all three properties are well satisfied by inversion in $\mathbb{F}_{2^N}$ (additionally mapping 0 to 0), for which:

(*i*) $\mathcal{N}(S) \geqslant 2^{N-1} - 2^{N/2}$, with equality for even N;

(*ii*) S has degree $N - 1$; and

(*iii*) $\delta(S) = 2$ for odd N, $\delta(S) = 4$ for even N.

Canteaut [Can16] furthermore notes that for even $N \geqslant 8$, there is no known function achieving better nonlinearity or differential uniformity (for the latter, it is even proved impossible), and when $N \equiv 0 \pmod 4$, we do not even know of other functions than inversion (up to affine equivalence) reaching these values.

Since inversion over $\mathbb{F}_{2^N}$ is close to optimal in all three above-listed properties (and no better function is known for even n), we use it as the basis of our S-box, similarly to the Rijndael one. As an aside, a strong argument in favor of this choice, which did not apply for Rijndael, is that inversion provides the basis for a strong S-box *for any block size*: as already mentioned, a major goal in designing our ad-hoc block cipher is to achieve run-time configurability of the block size. With this objective in mind, using a class on substitution functions that perform well for any N is more important than finding the best such function for an individual block size.

6.3.6.1 Tower-field inversion

Finite-field inversion is complex to implement as an arithmetic operation. Inversion based on Fermat's Little Theorem or on the Extended Euclidean Algorithm [Col69], while applicable to any finite field, is difficult to compute with a very short latency. On the other hand, if separate lookup tables were used for the inversion in $\mathbb{F}_{2^N}$ for each $N \in [\![5, 10]\!]$, it would not be possible to share any part of these lookup tables, and the overall area would be unacceptably large.

A more efficient approach, when possible, is to consider a finite field as a *tower field*, through the following transformation: $\mathbb{F}_{p^k} \cong \mathbb{F}_p[X]/f_k(X)$, where $f_k(X)$ is an irreducible polynomial of degree k over $\mathbb{F}_p$, and p is a prime or prime power. In this way, finding an inverse in $\mathbb{F}_{p^k}$ reduces to finding an inverse in $\mathbb{F}_p$, and computing a few multiplications and additions over this subfield. This technique has notably been used for efficient masked implementations of the AES S-box (see *e.g.* [BP10]).

Concretely, we consider $\mathbb{F}_{2^6}$, $\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^{10}}$ as quadratic extension fields of $\mathbb{F}_{2^3}$, $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^5}$ respectively. On the other hand, $\mathbb{F}_{2^9}$ will be treated as a cubic extension field of $\mathbb{F}_{2^3}$. Since $\mathbb{F}_{2^7}$ cannot be decomposed in this way, inversion over it will be computed with a lookup table. We use the following representations for our finite fields:

$$\begin{aligned}
&F_3 = \mathbb{F}_2[x]/(x^3+x+1) && F_6 = F_3[Y]/(Y^2+Y+1) && F_9 = F_3[Y]/(Y^3+Y+x)\,,\\
&F_4 = \mathbb{F}_2[x]/(x^4+x^3+1) && F_8 = F_4[Y]/(Y^2+Y+x)\,,\\
&F_5 = \mathbb{F}_2[x]/(x^5+x^3+1) && F_{10} = F_5[Y]/(Y^2+Y+1)\,,\\
&F_7 = \mathbb{F}_2[x]/(x^7+x+1)\,.
\end{aligned}$$

These representations have been chosen so as to facilitate resource sharing when implementing inversion over these fields: in particular, the reducing polynomials of the three quadratic extension fields are as similar as possible.

Let us illustrate how inversion is computed by taking the example of F_6 (which is isomorphic to $\mathbb{F}_{2^6}$). Let us choose an arbitrary $a = a_1Y + a_0 \in F_6$, and let $b = b_1Y + b_0 \in F_6$ be its inverse. Then, we have

$$\begin{aligned}
1 &= (a_1Y + a_0) \times (b_1Y + b_0) = a_1b_1Y^2 + (a_1b_0 + a_1b_0)Y + a_0b_0\,,\\
&= (a_1b_0 + a_0b_1 + a_1b_1)Y + (a_0b_0 + a_1b_1) \quad \text{since } Y^2 = Y + 1\,,
\end{aligned}$$

which implies $a_1b_0 + a_0b_1 + a_1b_1 = 1$ and $a_0b_0 + a_1b_1 = 0$. Solving for (b_0, b_1) gives

$$\begin{aligned}
b_0 &= (a_0(a_0 + a_1) + a_1^2)^{-1} \times (a_0 + a_1)\,,\\
b_1 &= (a_0(a_0 + a_1) + a_1^2)^{-1} \times a_1\,,
\end{aligned}$$

which allows us to compute inversion in F_6 with a few operations in the base field F_3: an inversion, three multiplications, a squaring and two additions.

6.3.6.2 Affine mapping

Since finite-field inversion is its own inverse function, and has 0 as a fixed point, it is not sufficient on its own for a secure S-box, and should be composed with a well-chosen affine mapping[7] [Nyb94, DR02]. It is noteworthy that composition with an affine function does not change the nonlinearity and differential uniformity of the S-box. In the case of AES, this affine function is crucial to break the strong algebraic structure of inversion: without it, it would be possible to derive the secret key from plaintext–ciphertext pairs, by solving low-degree polynomial equations. More practically for our shuffling, if the S-box is self-inverse, then two consecutive rounds having a null round key have a null

[7] We insist that this affine mapping is meant to be part of the S-box, in the same way that the Rijndael S-box is the composition of inversion in $\mathbb{F}_{2^8}$ with an invertible affine transformation [DR02]. We are not alluding to a linear permutation layer alike to the ShiftRows and MixColumns operations of AES: as already mentioned, since our block-cipher construction operates on a single word, there is no need for such a diffusion layer.

effect on the index. For instance, with two rounds and an arbitrary round-seed $x \in \mathbb{b}^N$, seeds $(x, 0, 0)$ and $(0, 0, x)$ generate the same permutation.

Unfortunately, we did not have the opportunity to design this affine layer. We will show below, through experimental evaluation, that our shuffling still has good properties, but they can likely be improved with a careful choice for this affine mapping.

6.3.7 Hardware implementation

We implement finite-field inversion over fields $F_5, \ldots, F_{10}$ with a single inversion unit comprised of a configurable square-field inverter operating over base field F_3, F_4 or F_5, a cube-field inverter operating over base field F_3, and a dedicated lookup table for inversion over F_7. A block diagram of this inverter is given in fig. 6.6.

When implemented in hardware in a 40 nm technology, this architecture performs Galois-Field inversion in a single clock cycle with a critical-path delay that can easily reach down to 3 ns, while using an area of 1.1 kGE. The square-field inverter, cube-field inverter and inverter over F_7 respectively take up 55 %, 19 % and 23 % of this overall area. By pipelining the permutation rounds, it is then possible to compute an m-round shuffling with $m + 1$ cycles of latency (including the addition of the last round seed), using m separate S-boxes.

6.3.8 Experimental evaluation

We try to evaluate how our shuffling fares in terms of the properties listed in section 6.3.4. The first property, *marginal uniformity*, is easy to prove. Considering shuffling over N-bit indexes with m full rounds and $m + 1$ round seeds, for any fixed input index $i \in \mathbb{b}^N$ and any choice of seeds for the m first rounds, each choice of the $(m + 1)^{\text{th}}$ seed maps i to a different shuffled index $E_{K_0,\ldots,K_m}(i)$. Marginal uniformity is thus achieved.

We can experimentally evaluate the entropy of the permutation up to about 40 bits. We proceed in the following way. We enumerate all possible choices for the first m seeds, generating the corresponding permutations until before the last round-key addition. As previously mentioned, the 2^N possible choices for the last seed will generate 2^N different

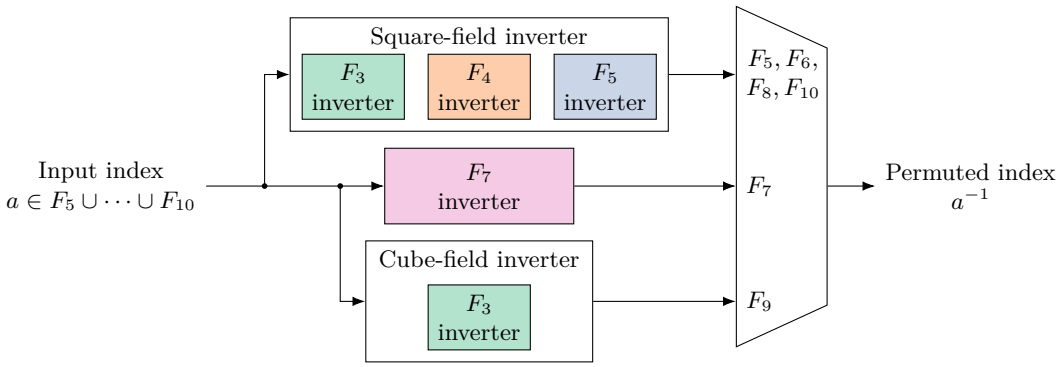


Figure 6.6: Structure of the configurable S-box based on Galois-Field inversion

permutations for a fixed choice of the other seeds; however, these permutations may collide with others having different values for the first seeds. To avoid this possibility of miscounting, we choose the last seed so as to *normalize* the permutation by ensuring that it has 0 as a fixed point. Concretely, for each choice of $K_0, \ldots, K_{m-1}$, we choose

$$K_m = E_{K_0,\ldots,K_{m-1},0}(0)\,.$$

The normalized permutations generated in this way are counted with the help of the HyperLogLog algorithm[8] [FFGM07], and this count is multiplied by 2^N to account for the choice of the last-round seed. We use 2^{20} registers for HyperLogLog, which allows for counting with a relative standard error of 2^{-10}, using 1 MB of memory.

The obtained results are reported in fig. 6.7 in the following way: the horizontal axis of the plot measures the number of bits of entropy available for choosing the seed, equal to $(m+1)N$, while the vertical axis represents how much less entropy is present at the output of the permutation generator. We plot the efficiency of shuffling over N-bit indexes, from one round of shuffling to the maximum that we were able to compute in a reasonable time (8 rounds for $N = 5$, 3 rounds for $N = 10$). We notice that, even for permutations over $N = 5$ bits, the shuffling has a very good randomness efficiency: with 8 rounds and 45 bits of input entropy for seeding, we lose less than one bit of entropy and obtain approximately $2^{44.4}$ distinct permutations. Understandably, better efficiency is achieved for larger N: for instance, the lost output entropy for $N = 10$ stays within twice the standard error of counting. We can thus consider the property of *randomness efficiency* to be satisfied. To reach our target output entropy of at least 30 bits, we need six rounds of shuffling for $N = 5$ and five rounds for $N = 6$; four rounds are amply sufficient for $N > 7$.

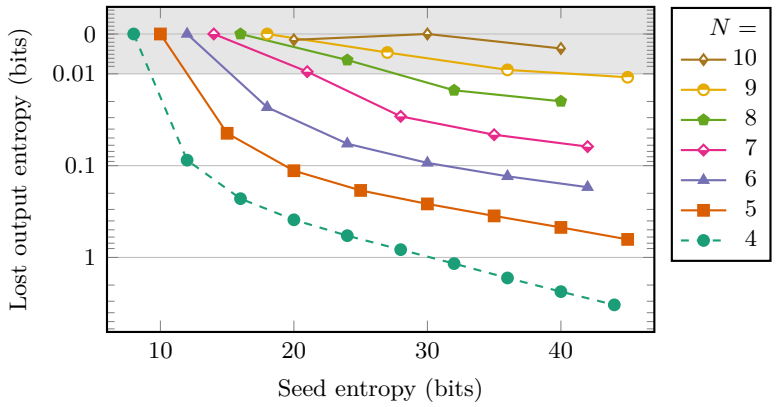


Figure 6.7: Randomness efficiency of our shuffling when permuting N-bit indexes. The vertical axis is logarithmic for values greater than 0.01, but linear around 0.

[8] HyperLogLog is a probabilistic counting algorithm that approximates the number of distinct elements among extremely large lists, while using comparatively very little memory.

We did not attempt to measure *joint uniformity* (notably since we are not sure which measure of uniformity is relevant, and how to evaluate it efficiently), but given the good results on entropy and entropy efficiency, we conjecture that this property should be satisfied to an acceptable level.

Although we do not have a model accurately describing the number of distinct permutations obtained with our method, the following inequality holds (up to the counting error) for all our experiments:

$$\text{for small } m, \quad 2^N \frac{2^N!}{(2^N - m)!} \leqslant \mathcal{P}(N, m) \leqslant 2^{N(m+1)}, \tag{6.8}$$

where $\mathcal{P}(N, m)$ is the number of distinct N-bit permutations obtained with m rounds. More specifically, the middle term is close to the geometric mean of the outer terms. The right-hand side is the number of possible seeds, so this side of the inequality is obviously satisfied. The left-hand side is 2^N times the number of size-m permutations from a set of 2^N items, so this side of the inequality cannot be satisfied for m close to 2^N (for which it becomes larger than the number of permutations of 2^N elements). However, the fact that this lower bound holds for small m seems a promising sign of the maximum number of permutations that can be achieved with this method. It should be noted that at most half of the permutations of $\mathbb{b}^N$ can be obtained in this way, since each round has even parity[9], but this limitation is unlikely to matter.

For a better understanding of how exhaustive this shuffling method is, we also plot in fig. 6.7 the results we would obtain if we were to shuffle 4-bit indexes, for which all possible permutations ($16! \approx 2^{44.25}$) can be enumerated in a reasonable time. The behavior of shuffling over this range is also represented in a different way in fig. 6.8, which

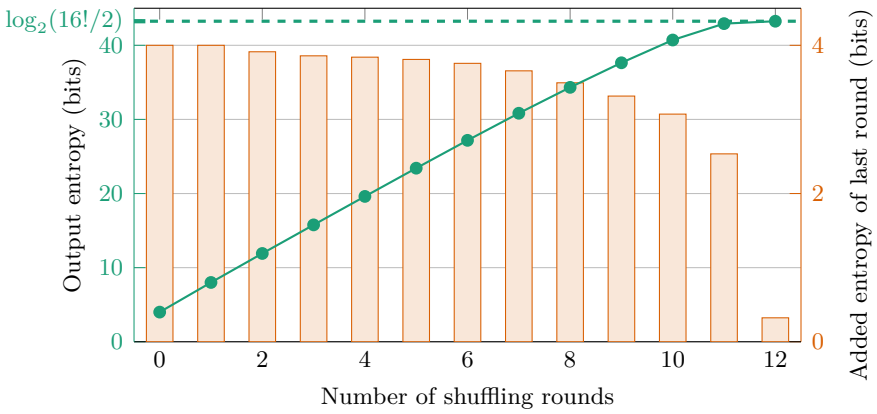


Figure 6.8: Evolution of shuffling entropy when permuting 4-bit indexes. The solid line shows the total output entropy with a specific number of rounds, while the bars show how much output entropy is added by each round.

[9]The parity of a permutation is the parity of the number of transpositions (exchange of two elements) needed to describe it.

illustrates that the output entropy of shuffling has an almost linear increase with the number of rounds, quickly approaching the total number of permutations having even parity, 16!/2. After 11 rounds and 48 bits for seeding, the number of distinct permutations is $0.4 \times 16!$, which amounts to 42.9 bits of output entropy. An additional round achieves within the 2^{-10} relative counting error of 16!/2, with 83 % entropy efficiency. For larger N, we suspect that we can similarly approach $2^N!/2$ distinct permutations after approximately $\log_2(2^N!)/N$ rounds. Unfortunately, checking this hypothesis for $N \geqslant 5$ is computationally infeasible, but the very similar behavior exhibited for different values of N in fig. 6.7, together with the fact that all our experiments satisfy eq. (6.8), give credence to this conjecture.

6.3.9 Summary and open problems

We have described in this section a method to shuffle indexes from a power-of-two range. Our proposal, based on a block-cipher–like construction, does not need precomputations, instead transforming indexes on the fly. This feature is first an advantage in terms of efficiency, since it only delays shuffled operations by a small number of cycles, which can easily be hidden by pipelining when several shuffled operations are performed sequentially. It is also, and most importantly, a strong added value in terms of security, as it prevents attackers from easily separating the side-channel leakage of the permutation itself from that of the shuffled operation.

The experimental evaluation of our proposal seems to confirm that this construction satisfies the informal security properties we targeted. However, a more formal definition and evaluation of these properties would be strongly beneficial. Adding an appropriate affine mapping to the S-box would likely improve its properties: this analysis is left as future work.

One of our key objectives in terms of applicability was to be able to generate permutations of different-sized ranges at runtime. As a matter of fact, we showed that power-of-two ranges from 2^5 to 2^{10} can all be supported with a small hardware area. Smaller ranges can be handled at nearly no additional cost; however, for 2^3 or fewer values, the maximum number of permutations becomes very small, so shuffling may bring very little additional security. A very intriguing question is whether a construction of this kind can be efficiently generalized to arbitrary, non–power-of-two ranges, to support other schemes such as NTRU, which uses polynomials whose degree is not a power of two. In this case, the algebraic and computational properties of binary fields can no longer be used, and an extension of our solution is needed.

7 Conclusion

7.1 Closing remarks

Migrating from the current cryptographic systems to ones resisting to quantum computers is an urgent task. Nonetheless, such a global transformation of public-key cryptosystems must be rooted in an extensive understanding of the security and efficiency of the proposed post-quantum cryptographic schemes, despite their relative novelty. Among other concerns, the security of implementations of these schemes against side-channel attacks remains insufficiently studied.

This work constitutes our contribution to the research in this field. Through its focus on embedded devices, it complements the many other facets of the state of the art, too few of which acknowledge the simultaneous need for strong protections against side-channel attacks, and for efficient implementations whose cost can be borne by resource-constrained devices.

After recalling the construction of two schemes of lattice-based cryptography, Kyber and Dilithium, we have conducted a literature review of the major kinds of physical attacks against them. In our view, a key takeaway from this review is the importance of side-channel and fault-injection techniques implementing oracle-based attacks against key-encapsulation mechanisms. Indeed, we saw how wide the attack surface is for realizing such oracles, and how difficult it is to exhaustively protect against them. Of particular importance also, are soft-analytical side-channel attacks, since they are able to efficiently cope with noisy leakage. Furthermore, as they rely on a single side-channel trace, they are not easily defeated by the usual masking countermeasure.

Precisely, building upon one of these soft-analytical attacks, we have shown that fine-tuning the model assumed by a belief-propagation attack so that it closely matches the targeted cryptographic implementation allows it to perform reliably even in relatively high noise conditions.

On the other hand, new or improved countermeasures against side-channel attacks have been exhibited. We have first shown an area-efficient implementation of masked arithmetic addition over Boolean shares. While this protected operation has been implemented in several ways in the literature, only one earlier work proposes an architecture that can reasonably be used in low-cost devices. Our proposal further decreases the hardware area occupied by the operation, and very importantly, formally proves its security against side-channel attacks. Moreover, our construction is unique in that it natively supports modular addition with an arbitrary modulus: this capability is required for schemes such as Kyber and Dilithium, and supporting it at the root of the solution is much more efficient than adding it on top of power-of-two addition as proposed in the literature.

Countermeasures that are dedicated to the two cryptographic schemes of interest have also been proposed. Our solution for the masking of Kyber ciphertext compression proposes a new take on the problem with respect to the state of the art. Thereby, it achieves slightly better asymptotic efficiency, while being similarly applicable to high-order masking. In the case of Dilithium, we closed a gap in the literature: to the best of our knowledge, no masked implementation of small-norm uniform sampling had been published. Our proposal corrects this unfortunate situation with an efficient masked gadget capable of implementing the operation at any order. Last but not least considering the previously mentioned concerns about single-trace attacks, we provided a new implementation of shuffling, attempting to mitigate the flaws of previous realizations of this countermeasure. Indeed, the lack of balance between cost and security in the literature on this topic, and structural weaknesses of previously proposed solutions, have led to effective attacks against them. While we do not pretend that our shuffling counters them all, we argued that it should significantly reduce the attack surface.

A crucial topic which has not been addressed by this work is the protection of lattice-based cryptography against fault-injection attacks. Very importantly, such protection is not straightforward to combine with the protection against side-channel attacks: some countermeasures against faults tend to facilitate side-channel analysis, and conversely. Combined protection against both thus remains a major opportunity for future work.

7.2 List of contributions

Publications and patents Along the course of this thesis, two peer-reviewed articles have been published. The first one [AEVR23], published in the proceedings of the IEEE HOST conference, describes an incremental improvement to an existing side-channel attack against Kyber. The second article [AEV24], published in the IACR journal of *Communications in Cryptology*, exhibits an efficient implementation of a generic operation protected against side-channel attacks, having particular interest for lattice-based cryptography. Besides its very small hardware area with respect to the state of the art, it benefits from a formal proof of robustness in a well-established security model. Both research articles have been reproduced in this document.

[AEVR23] Guilhèm Assael, Philippe Elbaz-Vincent, and Guillaume Reymond. Improving single-trace attacks on the number-theoretic transform for Cortex-M4. In *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 111–121, 2023. doi:10.1109/HOST55118.2023.10133270.

[AEV24] Guilhèm Assael and Philippe Elbaz-Vincent. Provably secure and area-efficient modular addition over Boolean shares. *IACR Communications in Cryptology*, 1(2), July 2024. doi:10.62056/aee0zoja5.

In addition to these, a new technique for the protection of a component of Kyber against side-channel attacks, specifically a masked ciphertext-compression operation, has been co-invented with Gilles Van Assche (STMicroelectronics, Diegem, Belgium), and

has been submitted as a patent application by STMicroelectronics Rousset. Having been submitted in 2024, this application has not yet been published, and is thus not explicitly referenced here. The pertaining solution has nonetheless been developed in this document.

Communications in seminars and conferences Our work has also given rise to communications in research seminars and conventions, both in an academic and in a corporate context.

PhD days of Institut Fourier, Grenoble (France), October 2022. This event, during which the doctoral students of Institut Fourier (mathematics laboratory of the university of Grenoble) shared their research, was the occasion to give a somewhat vulgarized introduction to post-quantum cryptography, with its promises and challenges in terms of efficiency and security.

Journées Codage et Cryptographie, Najac (France), October 2023. During this gathering of the French research community at the intersection between coding theory and cryptography, we presented our work covered by [AEVR23].

Journée des doctorants de STMicroelectronics, Rousset (France), May 2024. Like the other PhD students of STMicroelectronics Rousset, we took advantage of this day to present our research results to a wide corporate audience.

Unpublished contributions Two additional contributions have not yet been published due to being very recent and needing some additional examination to stand by themselves. First, a solution to mask a sampling operation of Dilithium has been developed, protecting it against vertical side-channel attacks. Second, an implementation of the shuffling countermeasure, randomizing the order of independent operations in a scheme, has been proposed, with a qualitative and quantitative review of its potential benefits with respect to existing solutions. The status of these contributions has been detailed in sections 6.2 and 6.3.

Bibliography

[AAB+19a] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, and Gilles Zémor. HQC. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-2-submissions`.

[AAB+19b] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillippe Gaborit, Gilles Zémor, Alain Couvreur, and Adrien Hauteville. RQC. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-2-submissions`.

[AAB+19c] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[AAB+22] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, Jurjen Bos, Arnaud Dion, Jerome Lacan, Jean-Marc Robert, and Pascal Veron. HQC. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions`.

[AAC+22] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, and Yi-Kai Liu. Status report on the third round of the NIST post-quantum cryptography standardization process. Technical report, National Institute of Standards and Technology, September 2022. NIST IR 8413. `doi:10.6028/NIST.IR.8413-upd1`.

[AB74] R. Agarwal and C. Burrus. Fast convolution using Fermat number transforms with applications to digital filtering. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 22(2):87–97, 1974. `doi:10.1109/TASSP.1974.1162555`.

[ABB+19] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar-Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zémor, and Valentin Vasseur. BIKE. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-2-submissions`.

[ABC+22] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions`.

[ABC+23] Melissa Azouaoui, Olivier Bronchain, Gaëtan Cassiers, Clément Hoffmann, Yulia Kuzovkova, Joost Renes, Tobias Schneider, Markus Schönauer, François-Xavier Standaert, and Christine van Vredendaal. Protecting Dilithium against leakage revisited sensitivity analysis and improved implementations. *IACR TCHES*, 2023(4):58–79, 2023. `doi:10.46586/tches.v2023.i4.58-79`.

[ABD+19] Nicolas Aragon, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, Olivier Ruatta, Jean-Pierre Tillich, Gilles Zémor, Carlos Aguilar-Melchor, Slim Bettaieb, Loic Bidoux, Magali Bardet, and Ayoub Otmani. ROLLO. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-2-submissions`.

[ABD+21] Roberto Avanzi, Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber – Algorithm specifications and supporting documentation. Version 3.01, January 2021. URL: `https://pq-crystals.org/kyber/data/kyber-specification-round3-20210131.pdf`.

[AD17] Martin R. Albrecht and Amit Deo. Large modulus ring-LWE $\geqslant$ module-LWE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 267–296. Springer, Cham, December 2017. `doi:10.1007/978-3-319-70694-8_10`.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016.

[ADR02] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 83–107. Springer, Berlin, Heidelberg, April / May 2002. doi:10.1007/3-540-46035-7_6.

[AES01] Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce, November 2001.

[AEV24] Guilhèm Assael and Philippe Elbaz-Vincent. Provably secure and area-efficient modular addition over Boolean shares. *IACR Communications in Cryptology*, 1(2), July 2024. doi:10.62056/aee0zoja5.

[AEVR23] Guilhèm Assael, Philippe Elbaz-Vincent, and Guillaume Reymond. Improving single-trace attacks on the number-theoretic transform for Cortex-M4. In *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 111–121, 2023. doi:10.1109/HOST55118.2023.10133270.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996. doi:10.1145/237814.237838.

[Ajt98] Miklós Ajtai. The shortest vector problem in L2 is NP-hard for randomized reductions (extended abstract). In *30th ACM STOC*, pages 10–19. ACM Press, May 1998. doi:10.1145/276698.276705.

[AMD+21] Abubakr Abdulgadir, Kamyar Mohajerani, Viet Ba Dang, Jens-Peter Kaps, and Kris Gaj. A lightweight implementation of saber resistant against side-channel attacks. In Avishek Adhikari, Ralf Küsters, and Bart Preneel, editors, *INDOCRYPT 2021*, volume 13143 of *LNCS*, pages 224–245. Springer, Cham, December 2021. doi:10.1007/978-3-030-92518-5_11.

[ANS22] ANSSI. ANSSI views on the post-quantum cryptography transition. Technical position paper, March 2022. URL: https://cyber.gouv.fr/sites/default/files/document/EN_Position.pdf.

[ANS23] ANSSI. ANSSI views on the post-quantum cryptography transition (2023 follow up). Technical position paper, December 2023. URL: https://cyber.gouv.fr/sites/default/files/document/follow_up_position_paper_on_post_quantum_cryptography.pdf.

[BB03] David Brumley and Dan Boneh. Remote timing attacks are practical. In *USENIX Security 2003*. USENIX Association, August 2003.

[BBE+18] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the GLP lattice-based signature scheme at any order. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 354–384. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78375-8_12.

[BCDP96] David Beckman, Amalavoyal N Chari, Srikrishna Devabhaktuni, and John Preskill. Efficient networks for quantum factoring. *Physical Review A*, 54(2):1034, 1996.

[BDH+21] Shivam Bhasin, Jan-Pieter D'Anvers, Daniel Heinz, Thomas Pöppelmann, and Michiel Van Beirendonck. Attacking and defending masked polynomial comparison. *IACR TCHES*, 2021(3):334–359, 2021. URL: https://tches.iacr.org/index.php/TCHES/article/view/8977, doi:10.46586/tches.v2021.i3.334-359.

[BDK+21] Shi Bai, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium – Algorithm specifications and supporting documentation. Version 3.1, February 2021. URL: https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf.

[BDK+23] Uri Blumenthal, Quynh H. Dang, Panos Kampanakis, Rafael Mattsson, John Misoczki, Ruben Niederhagen, Markku-Juhani O. Saarinen, Sophie Schmieg, Peter Schwabe, and Bas Westerbaan. Discussion about Kyber's tweaked FO transform. Public posts on NIST pqc-forum Google Group, May 2023. URL: https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/WFRDl8DqYQ4/m/-T0qpR2WAwAJ.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 37–51. Springer, Berlin, Heidelberg, May 1997. doi:10.1007/3-540-69053-0_4.

[BDPA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 313–314. Springer, Berlin, Heidelberg, May 2013. doi:10.1007/978-3-642-38348-9_19.

[BDS23] Sven Bauer and Fabrizio De Santis. Forging Dilithium and Falcon signatures by single fault injection. In *2023 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, pages 81–88, 2023. doi:10.1109/FDTC60478.2023.00017.

[Bec21] Alison Becker. Re: [lamps] Hybrid-for-the-sake-of-security, and Composite vs Non-Composite. IETF mail archive, November 2021. Message on the *spasm* mailing list. URL: https://mailarchive.ietf.org/arch/msg/spasm/McksDhejGgJJ6xG617FEWLbBq0k/.

[Bed62] O. J. Bedrij. Carry-select adder. *IRE Transactions on Electronic Computers*, EC-11(3):340–346, 1962. doi:10.1109/IRETELC.1962.5407919.

[Ber23a] Daniel J. Bernstein. Kyber decisions, part 2: FO transform. Public posts on NIST pqc-forum Google Group, January 2023. URL: https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/C0D3W1KoINY/m/hNIyCYVlCAAJ.

[Ber23b] Daniel J. Bernstein. variable-time Kyber ref software. Public post on NIST pqc-forum Google Group, December 2023. URL: https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/hWqFJCucuj4/m/-Z-jm_k9AAAJ.

[Ber24] Daniel J. Bernstein. KyberSlash: division timings depending on secrets in Kyber software. Personal webpage, 2024. URL: https://kyberslash.cr.yp.to/.

[Beu21] Ward Beullens. Improved cryptanalysis of UOV and Rainbow. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 348–373. Springer, Cham, October 2021. doi:10.1007/978-3-030-77870-5_13.

[Beu22] Ward Beullens. Breaking rainbow takes a weekend on a laptop. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 464–479. Springer, Cham, August 2022. doi:10.1007/978-3-031-15979-4_16.

[BG22] Florian Bache and Tim Güneysu. Boolean masking for arithmetic additions at arbitrary order in hardware. *Applied Sciences*, 12(5), 2022. URL: https://www.mdpi.com/2076-3417/12/5/2274, doi:10.3390/app12052274.

[BGR+21] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Masking Kyber: First- and higher-order implementations. *IACR TCHES*, 2021(4):173–214, 2021. URL: https://tches.iacr.org/index.php/TCHES/article/view/9064, doi:10.46586/tches.v2021.i4.173-214.

[BGRR19] Aurélie Bauer, Henri Gilbert, Guénaël Renault, and Mélissa Rossi. Assessment of the key-reuse resilience of NewHope. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 272–292. Springer, Cham, March 2019. doi:10.1007/978-3-030-12612-4_14.

[BJRW23] Katharina Boudgoust, Corentin Jeudy, Adeline Roux-Langlois, and Weiqiang Wen. On the hardness of module learning with errors with short distributions. *Journal of Cryptology*, 36(1):1, January 2023. doi:10.1007/s00145-022-09441-3.

[Bla79] G. R. Blakley. Safeguarding cryptographic keys. *Proceedings of AFIPS 1979 National Computer Conference*, 48:313–317, 1979.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 575–584. ACM Press, June 2013. doi:10.1145/2488608.2488680.

[BMR21] Luk Bettale, Simon Montoya, and Guénaël Renault. Safe-error analysis of post-quantum cryptography mechanisms. In *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, pages 39–44, September 2021. doi:10.1109/FDTC53659.2021.00015.

[BMRT22] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. IronMask: Versatile verification of masking security. In *2022 IEEE Symposium on Security and Privacy*, pages 142–160. IEEE Computer Society Press, May 2022. doi:10.1109/SP46214.2022.9833600.

[BNGD23] Linus Backlund, Kalle Ngo, Joel Gärtner, and Elena Dubrova. Secret key recovery attack on masked and shuffled implementations of CRYSTALS-Kyber and Saber. In Jianying Zhou, Lejla Batina, Zengpeng Li, Jingqiang Lin, Eleonora Losiouk, Suryadipta Majumdar, Daisuke Mashima, Weizhi Meng, Stjepan Picek, Mohammad Ashiqur Rahman, Jun Shao, Masaki Shimaoka, Ezekiel Soremekun, Chunhua Su, Je Sen Teh, Aleksei Udovenko, Cong Wang, Leo Zhang, and Yury Zhauniarovich, editors, *Applied Cryptography and Network Security Workshops*, pages 159–177, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-41181-6_9.

[BP10] Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In Paola Festa, editor, *Experimental Algorithms*, pages 178–189, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-13193-6_16.

[BPO+20] Florian Bache, Clara Paglialonga, Tobias Oder, Tobias Schneider, and Tim Güneysu. High-speed masking for polynomial comparison in lattice-based kems. *IACR TCHES*, 2020(3):483–507, 2020. URL: https://tches.iacr.org/index.php/TCHES/article/view/8598, doi:10.13154/tches.v2020.i3.483-507.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors,

EUROCRYPT 2012, volume 7237 of *LNCS*, pages 719–737. Springer, Berlin, Heidelberg, April 2012. doi:10.1007/978-3-642-29011-4_42.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 513–525. Springer, Berlin, Heidelberg, August 1997. doi:10.1007/BFb0052259.

[BS23] Manuel Barbosa and Peter Schwabe. Kyber terminates. Cryptology ePrint Archive, Report 2023/708, 2023. URL: https://eprint.iacr.org/2023/708.

[BSI24] BSI: German federal office for Information Security. Cryptographic mechanisms: Recommendations and key lengths, version 2024-01. Technical guideline BSI TR-02102-1, January 2024. URL: https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr02102/tr02102_node.html.

[BUC19] Utsav Banerjee, Tenzin S. Ukyab, and Anantha P. Chandrakasan. Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols. *IACR TCHES*, 2019(4):17–61, 2019. URL: https://tches.iacr.org/index.php/TCHES/article/view/8344, doi:10.13154/tches.v2019.i4.17-61.

[CAD+20] David Cooper, Daniel Apon, Quynh Dang, Michael Davidson, Morris Dworkin, and Carl Miller. Recommendation for stateful hash-based signature schemes. National Institute of Standards and Technology, NIST SP 800-208, U.S. Department of Commerce, October 2020. doi:10.6028/NIST.SP.800-208.

[Cam17] Thomas Camus. *Méthodes algorithmiques pour les réseaux algébriques*. PhD thesis, Université Grenoble Alpes, Grenoble, France, July 2017. URL: https://theses.hal.science/tel-01563081.

[Can16] Anne Canteaut. Lecture notes on cryptographic Boolean functions, March 2016. URL: https://www.rocq.inria.fr/secret/Anne.Canteaut/poly.pdf.

[CCD00] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In Çetin Kaya Koç and Christof Paar, editors, *CHES 2000*, volume 1965 of *LNCS*, pages 252–263. Springer, Berlin, Heidelberg, August 2000. doi:10.1007/3-540-44499-8_20.

[CD23] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023,*

Part V, volume 14008 of *LNCS*, pages 423–447. Springer, Cham, April 2023. doi:10.1007/978-3-031-30589-4_15.

[CFOS21] Gaëtan Cassiers, Sebastian Faust, Maximilian Orlt, and François-Xavier Standaert. Towards tight random probing security. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 185–214, Virtual Event, August 2021. Springer, Cham. doi:10.1007/978-3-030-84252-9_7.

[CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Transactions on Computers*, 70(10):1677–1690, 2021. doi:10.1109/TC.2020.3022979.

[CGM+23] Gaëtan Cassiers, Barbara Gigerl, Stefan Mangard, Charles Momin, and Rishub Nagpal. Compress: Reducing area and latency of masked pipelined circuits. Cryptology ePrint Archive, Report 2023/1600, 2023. https://eprint.iacr.org/2023/1600.

[CGMZ22] Jean-Sébastien Coron, François Gérard, Simon Montoya, and Rina Zeitoun. High-order table-based conversion algorithms and masking lattice-based encryption. *IACR TCHES*, 2022(2):1–40, 2022. doi:10.46586/tches.v2022.i2.1-40.

[CGMZ23] Jean-Sébastien Coron, François Gérard, Simon Montoya, and Rina Zeitoun. High-order polynomial comparison and masking lattice-based encryption. *IACR TCHES*, 2023(1):153–192, 2023. doi:10.46586/tches.v2023.i1.153-192.

[CGTZ23] Jean-Sébastien Coron, François Gérard, Matthias Trannoy, and Rina Zeitoun. Improved gadgets for the high-order masking of Dilithium. *IACR TCHES*, 2023(4):110–145, 2023. doi:10.46586/tches.v2023.i4.110-145.

[CGV14] Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala. Secure conversion between Boolean and arithmetic masking of any order. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 188–205. Springer, Berlin, Heidelberg, September 2014. doi:10.1007/978-3-662-44709-3_11.

[Chi21] Andrew M. Childs. Lecture notes on quantum algorithms. Department of Computer Science, Institute for Advanced Computer Studies, and Joint Center for Quantum Information and Computer Science, University of Maryland, April 2021. URL: http://www.cs.umd.edu/~amchilds/qa/qa.pdf [cited 2021-04-28].

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In

Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 398–412. Springer, Berlin, Heidelberg, August 1999. doi:10.1007/3-540-48405-1_26.

[CKMS17] Sanjit Chatterjee, Neal Koblitz, Alfred Menezes, and Palash Sarkar. Another look at tightness II: Practical issues in cryptography. In Raphaël C.-W. Phan and Moti Yung, editors, *Paradigms in Cryptology – Mycrypt 2016. Malicious and Exploratory Cryptology*, pages 21–55, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-61273-7_3.

[Cla88] Andrew J. Clark. Physical protection of cryptographic devices. In David Chaum and Wyn L. Price, editors, *EUROCRYPT'87*, volume 304 of *LNCS*, pages 83–93. Springer, Berlin, Heidelberg, April 1988. doi:10.1007/3-540-39118-5_9.

[Cla07] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 181–194. Springer, Berlin, Heidelberg, September 2007. doi:10.1007/978-3-540-74735-2_13.

[CMM+24] Gaëtan Cassiers, Loïc Masure, Charles Momin, Thorben Moos, Amir Moradi, and François-Xavier Standaert. Randomness generation for secure hardware masking – unrolled Trivium to the rescue. *IACR Communications in Cryptology*, 1(2), July 2024. doi:10.62056/akdkp2fgx.

[CNE+14] Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, Hovav Shacham, and Matthew Fredrikson. On the practical exploitability of dual EC in TLS implementations. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 319–335. USENIX Association, August 2014.

[Col69] George E. Collins. Computing multiplicative inverses in GF(p). *Mathematics of Computation*, 23(105):197–200, 1969. doi:10.1090/S0025-5718-1969-0242345-5.

[Cor99] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Çetin Kaya Koç and Christof Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 292–302. Springer, Berlin, Heidelberg, August 1999. doi:10.1007/3-540-48059-5_25.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with Probe Isolating Non-Interference. *IEEE Transactions on Information Forensics and Security*, 15:2542–2555, 2020. doi:10.1109/TIFS.2020.2971153.

[CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably secure hardware masking in the transition- and glitch-robust probing model: Better safe than sorry. *IACR TCHES*, 2021(2):136–158, 2021. URL: https://tches.iacr.org/index.php/TCHES/article/view/8790, doi:10.46586/tches.v2021.i2.136-158.

[CT03] Jean-Sébastien Coron and Alexei Tchulkine. A new algorithm for switching from arithmetic to Boolean masking. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 89–97. Springer, Berlin, Heidelberg, September 2003. doi:10.1007/978-3-540-45238-6_8.

[CVBH24] Andersson Calle Viera, Alexandre Berzati, and Karine Heydemann. Fault attacks sensitivity of public parameters in the Dilithium verification. In Shivam Bhasin and Thomas Roche, editors, *Smart Card Research and Advanced Applications*, pages 62–83, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-54409-5_4.

[DB22] Jan-Pieter D'Anvers and Senne Batsleer. Multitarget decryption failure attacks and their application to Saber and Kyber. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 3–33. Springer, Cham, March 2022. doi:10.1007/978-3-030-97121-2_1.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 423–440. Springer, Berlin, Heidelberg, May 2014. doi:10.1007/978-3-642-55220-5_24.

[DDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 329–358. Springer, Cham, August 2020. doi:10.1007/978-3-030-56880-1_12.

[Deb12] Blandine Debraize. Efficient and provably secure methods for switching from arithmetic to Boolean masking. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 107–121. Springer, Berlin, Heidelberg, September 2012. doi:10.1007/978-3-642-33027-8_7.

[DFP+24] Julien Devevey, Pouria Fallahpour, Alain Passelègue, Damien Stehlé, and Keita Xagawa. A detailed analysis of Fiat-Shamir with aborts. Cryptology ePrint Archive, Paper 2023/245, 2024. Major revision of an IACR publication in CRYPTO 2023. URL: https://eprint.iacr.org/2023/245.

[dG15] W. de Groot. A performance study of X25519 on Cortex-M3 and M4. Master's thesis, Eindhoven University of Technology, Eindhoven, Netherlands, 2015. URL: `https://research.tue.nl/en/studentTheses/a-performance-study-of-x25519-on-cortex-m3-and-m4`.

[DGJ+19] Jan-Pieter D'Anvers, Qian Guo, Thomas Johansson, Alexander Nilsson, Frederik Vercauteren, and Ingrid Verbauwhede. Decryption failure attacks on IND-CCA secure lattice-based schemes. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 565–598. Springer, Cham, April 2019. `doi:10.1007/978-3-030-17259-6_19`.

[DHP+22] Jan-Pieter D'Anvers, Daniel Heinz, Peter Pessl, Michiel Van Beirendonck, and Ingrid Verbauwhede. Higher-order masked ciphertext comparison for lattice-based cryptography. *IACR TCHES*, 2022(2):115–139, 2022. `doi:10.46586/tches.v2022.i2.115-139`.

[DKR+20] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel Van Beirendonck, and Andrea Basso. SABER. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions`.

[DR02] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002.

[DRV20] Jan-Pieter D'Anvers, Mélissa Rossi, and Fernando Virdia. (One) failure is not an option: Bootstrapping the search for failures in lattice-based encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 3–33. Springer, Cham, May 2020. `doi:10.1007/978-3-030-45727-3_1`.

[DS21] Léo Ducas and John Schanck. Security estimation scripts for Kyber and Dilithium. Public Github repository pq-crystals/security-estimates, 2021. Commit 75c2694. URL: `https://github.com/pq-crystals/security-estimates/tree/master`.

[DVV19] Jan-Pieter D'Anvers, Frederik Vercauteren, and Ingrid Verbauwhede. The impact of error dependencies on ring/mod-LWE/LWR based schemes. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*, pages 103–115. Springer, Cham, 2019. `doi:10.1007/978-3-030-25510-7_6`.

[DZD+18] A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsi Fei. Towards sound and optimal leakage detection procedure. In *Smart Card Research and Advanced Applications: 16th International*

Conference, CARDIS 2017, Lugano, Switzerland, November 13–15, 2017, Revised Selected Papers, pages 105–122, Cham, 2018. Springer.

[Esp19] Thomas Espitau. *Algebraic Lattices: Algorithmic Aspects*. PhD thesis, Sorbonne Université – Laboratoire d'Informatique de Paris 6, Paris, France, 2019.

[FBR+22] Tim Fritzmann, Michiel Van Beirendonck, Debapriya Basu Roy, Patrick Karl, Thomas Schamberger, Ingrid Verbauwhede, and Georg Sigl. Masked accelerators and instruction set extensions for post-quantum cryptography. *IACR TCHES*, 2022(1):414–460, 2022. doi:10.46586/tches.v2022.i1.414-460.

[FFGM07] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In Philippe Jacquet, editor, *2007 Conference on Analysis of Algorithms (AofA 07)*, volume AH of *DMTCS Proceedings*, pages 137–156, Juan les Pins, France, June 2007. Discrete Mathematics and Theoretical Computer Science. doi:10.46298/dmtcs.3545.

[FG05] Wieland Fischer and Berndt M. Gammel. Masking at gate level in the presence of glitches. In Josyula R. Rao and Berk Sunar, editors, *CHES 2005*, volume 3659 of *LNCS*, pages 187–200. Springer, Berlin, Heidelberg, August / September 2005. doi:10.1007/11545262_14.

[FGP+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR TCHES*, 2018(3):89–120, 2018. URL: https://tches.iacr.org/index.php/TCHES/article/view/7270, doi:10.13154/tches.v2018.i3.89-120.

[FIP24a] Module-lattice-based key-encapsulation mechanism standard. National Institute of Standards and Technology, NIST FIPS PUB 203, U.S. Department of Commerce, August 2024. doi:10.6028/NIST.FIPS.203.

[FIP24b] Module-lattice-based digital signature standard. National Institute of Standards and Technology, NIST FIPS PUB 204, U.S. Department of Commerce, August 2024.

[FIP24c] Stateless hash-based digital signature standard. National Institute of Standards and Technology, NIST FIPS PUB 205, U.S. Department of Commerce, August 2024.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Berlin, Heidelberg, August 1999. doi:10.1007/3-540-48405-1_34.

[FO13] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Journal of Cryptology*, 26(1):80–101, January 2013. doi:10.1007/s00145-011-9114-1.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987. doi:10.1007/3-540-47721-7_12.

[FWZ22] Boyue Fang, Weize Wang, and Yunlei Zhao. Tight analysis of decryption failure probability of Kyber in reality. In Cristina Alcaraz, Liqun Chen, Shujun Li, and Pierangela Samarati, editors, *ICICS 22*, volume 13407 of *LNCS*, pages 148–160. Springer, Cham, September 2022. doi:10.1007/978-3-031-15777-6_9.

[FY48] Ronald Aylmer Fisher and Frank Yates. *Statistical tables for biological, agricultural and medical research (3rd ed.)*. Oliver and Boyd, London, 1948.

[Gat18] Fletcher Gates. Reduction-respecting parameters for lattice-based cryptosystems. Master's thesis, McMaster University, Hamilton, Ontario, 2018. URL: http://hdl.handle.net/11375/24466.

[GC24] Google Quantum AI and Collaborators. Quantum error correction below the surface code threshold. arXiv preprint, paper 2408.13687, August 2024. URL: https://arxiv.org/abs/2408.13687.

[GE21] Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, April 2021. doi:10.22331/q-2021-04-15-433.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136. Cryptography Research Inc., 2011.

[GJN20] Qian Guo, Thomas Johansson, and Alexander Nilsson. A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 359–386. Springer, Cham, August 2020. doi:10.1007/978-3-030-56880-1_13.

[GMGL24] Guillaume Goy, Julien Maillard, Philippe Gaborit, and Antoine Loiseau. Single trace HQC shared key recovery with SASCA. *IACR TCHES*, 2024(2):64–87, 2024. doi:10.46586/tches.v2024.i2.64-87.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer, Berlin, Heidelberg, May 2001. doi:10.1007/3-540-44709-1_21.

[GMP22] Paul Grubbs, Varun Maram, and Kenneth G. Paterson. Anonymous, robust post-quantum public key encryption. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 402–432. Springer, Cham, May / June 2022. doi:10.1007/978-3-031-07082-2_15.

[GN07] Nicolas Gama and Phong Q. Nguyen. New chosen-ciphertext attacks on NTRU. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 89–106. Springer, Berlin, Heidelberg, April 2007. doi:10.1007/978-3-540-71677-8_7.

[Gou01] Louis Goubin. A sound method for switching between Boolean and arithmetic masking. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 3–15. Springer, Berlin, Heidelberg, May 2001. doi:10.1007/3-540-44709-1_2.

[Gra21] Heather Gray. Introduction to quantum computing. CERN Academic Training Lecture Regular Programme, March 2021. URL: https://indico.cern.ch/event/870515/ [cited 2021-04-28].

[GS21] Élie Gouzien and Nicolas Sangouard. Factoring 2048-bit RSA integers in 177 days with 13 436 qubits and a multimode memory. *Phys. Rev. Lett.*, 127:140503, Sep 2021. doi:10.1103/PhysRevLett.127.140503.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Cham, November 2017. doi:10.1007/978-3-319-70500-2_12.

[HHP+21] Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. Chosen ciphertext k-trace attacks on masked CCA2 secure Kyber. *IACR TCHES*, 2021(4):88–113, 2021. URL: https://tches.iacr.org/index.php/TCHES/article/view/9061, doi:10.46586/tches.v2021.i4.88-113.

[HMH+12] Johann Heyszl, Stefan Mangard, Benedikt Heinz, Frederic Stumpf, and Georg Sigl. Localized electromagnetic analysis of cryptographic implementations. In Orr Dunkelman, editor, *CT-RSA 2012*, volume 7178 of *LNCS*, pages 231–244. Springer, Berlin, Heidelberg, February / March 2012. doi:10.1007/978-3-642-27954-6_15.

[HNP+03] Nick Howgrave-Graham, Phong Q. Nguyen, David Pointcheval, John Proos, Joseph H. Silverman, Ari Singer, and William Whyte. The impact of decryption failures on the security of NTRU encryption. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 226–246. Springer, Berlin, Heidelberg, August 2003. doi:10.1007/978-3-540-45146-4_14.

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *ACNS 06International Conference on Applied Cryptography and Network Security*, volume 3989 of *LNCS*, pages 239–252. Springer, Berlin, Heidelberg, June 2006. doi:10.1007/11767480_16.

[HP23] Daniel Heinz and Thomas Pöppelmann. Combined fault and DPA protection for lattice-based cryptography. *IEEE Transactions on Computers*, 72(4):1055–1066, 2023. doi:10.1109/TC.2022.3197073.

[HPA21] James Howe, Thomas Prest, and Daniel Apon. SoK: How (not) to design and implement post-quantum cryptography. In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 444–477. Springer, Cham, May 2021. doi:10.1007/978-3-030-75539-3_19.

[HPP21] Julius Hermelink, Peter Pessl, and Thomas Pöppelmann. Fault-enabled chosen-ciphertext attacks on kyber. In Avishek Adhikari, Ralf Küsters, and Bart Preneel, editors, *INDOCRYPT 2021*, volume 13143 of *LNCS*, pages 311–334. Springer, Cham, December 2021. doi:10.1007/978-3-030-92518-5_15.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423 of *LNCS*, pages 267–288. Springer, June 1998.

[HSST23] Julius Hermelink, Silvan Streit, Emanuele Strieder, and Katharina Thieme. Adapting belief propagation to counter shuffling of NTTs. *IACR TCHES*, 2023(1):60–88, 2023. doi:10.46586/tches.v2023.i1.60-88.

[HZZ+22] Junhao Huang, Jipeng Zhang, Haosong Zhao, Zhe Liu, Ray C. C. Cheung, Çetin Kaya Koç, and Donglong Chen. Improved plantard arithmetic for lattice-based cryptography. *IACR TCHES*, 2022(4):614–636, 2022. doi:10.46586/tches.v2022.i4.614-636.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Berlin, Heidelberg, August 2003. doi:10.1007/978-3-540-45146-4_27.

[Jc24] Stephen Jordan and contributors. Quantum algorithm zoo, August 2024. URL: https://quantumalgorithmzoo.org [cited 2024-10-07].

[JFB+22] Jan Jancar, Marcel Fourné, Daniel De Almeida Braga, Mohamed Sabt, Peter Schwabe, Gilles Barthe, Pierre-Alain Fouque, and Yasemin Acar. "They're not that hard to mitigate": What cryptographic library developers think about timing attacks. In *2022 IEEE Symposium on Security and Privacy*, pages 632–649. IEEE Computer Society Press, May 2022. doi:10.1109/SP46214.2022.9833713.

[JGCS24] Arpan Jati, Naina Gupta, Anupam Chattopadhyay, and Somitra Kumar Sanadhya. A configurable CRYSTALS-Kyber hardware implementation with side-channel protection. *ACM Trans. Embed. Comput. Syst.*, 23(2), March 2024. doi:10.1145/3587037.

[JJ00] Éliane Jaulmes and Antoine Joux. A chosen-ciphertext attack against NTRU. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 20–35. Springer, Berlin, Heidelberg, August 2000. doi:10.1007/3-540-44598-6_2.

[Joz19] Richard Jozsa. Lecture notes on quantum information and computation. Department of Applied Mathematics and Theoretical Physics, University of Cambridge, 2019. URL: http://www.qi.damtp.cam.ac.uk/files/PartIIIQC/Part%202%20QIC%20lecturenotes.pdf [cited 2021-04-27].

[KF15] Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 43–62. Springer, Berlin, Heidelberg, August 2015. doi:10.1007/978-3-662-47989-6_3.

[Kho04] Subhash Khot. Hardness of approximating the shortest vector problem in lattices. In *45th FOCS*, pages 126–135. IEEE Computer Society Press, October 2004. doi:10.1109/FOCS.2004.31.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Berlin, Heidelberg, August 1999. doi:10.1007/3-540-48405-1_25.

[KLS18] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 552–586. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78372-7_18.

[KM22] David Knichel and Amir Moradi. Low-latency hardware private circuits. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1799–1812. ACM Press, November 2022. doi:10.1145/3548606.3559362.

[KNR+24] Andrew D. King, Alberto Nocera, Marek M. Rams, Jacek Dziarmaga, Roeland Wiersema, William Bernoudy, Jack Raymond, Nitin Kaushal, Niclas Heinsdorf, Richard Harris, Kelly Boothby, Fabio Altomare, Andrew J. Berkley, Martin Boschnak, Kevin Chern, Holly Christiani, Samantha Cibere, Jake Connor, Martin H. Dehn, Rahul Deshpande, Sara Ejtemaee, Pau Farré, Kelsey Hamer, Emile Hoskinson, Shuiyuan Huang, Mark W. Johnson, Samuel Kortas, Eric Ladizinsky, Tony Lai, Trevor Lanting, Ryan Li, Allison J. R. MacDonald, Gaelen Marsden, Catherine C. McGeoch, Reza Molavi, Richard Neufeld, Mana Norouzpour, Travis Oh, Joel Pasvolsky, Patrick Poitras, Gabriel Poulin-Lamarre, Thomas Prescott, Mauricio Reis, Chris Rich, Mohammad Samani, Benjamin Sheldan, Anatoly Smirnov, Edward Sterpka, Berta Trullas Clavera, Nicholas Tsai, Mark Volkmann, Alexander Whiticar, Jed D. Whittaker, Warren Wilkinson, Jason Yao, T. J. Yi, Anders W. Sandvik, Gonzalo Alvarez, Roger G. Melko, Juan Carrasquilla, Marcel Franz, and Mohammad H. Amin. Computational supremacy in quantum simulation. arXiv preprint, paper 2403.00910v1, 2024. URL: `https://arxiv.org/abs/2403.00910`, `arXiv:2403.00910`.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Berlin, Heidelberg, August 1996. `doi:10.1007/3-540-68697-5_9`.

[KPP20] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on Keccak. *IACR TCHES*, 2020(3):243–268, 2020. URL: `https://tches.iacr.org/index.php/TCHES/article/view/8590`, `doi:10.13154/tches.v2020.i3.243-268`.

[KPR+22] Matthias J. Kannwischer, Richard Petri, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4, November 2022. Commit 918f379. URL: `https://github.com/mupq/pqm4`.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - statistical independence and leakage verification. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 787–816. Springer, Cham, December 2020. `doi:10.1007/978-3-030-64837-4_26`.

[LB61] M. Lehman and N. Burla. Skip techniques for high-speed carry-propagation in binary arithmetic units. *IRE Transactions on Electronic Computers*, EC-10(4):691–698, 1961. `doi:10.1109/TEC.1961.5219274`.

[LDK+20] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-qua`

ntum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[LLJ+19] Xianhui Lu, Yamin Liu, Dingding Jia, Haiyang Xue, Jingnan He, Zhenfei Zhang, Zhe Liu, Hao Yang, Bao Li, and Kunpeng Wang. LAC. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-2-submissions.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Berlin, Heidelberg, May / June 2010. doi:10.1007/978-3-642-13190-5_1.

[LPS15] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A LLVM-based Python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2833157.2833162.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *DCC*, 75(3):565–599, 2015. doi:10.1007/s10623-014-9938-4.

[LS19] Vadim Lyubashevsky and Gregor Seiler. NTTRU: Truly fast NTRU using NTT. *IACR TCHES*, 2019(3):180–201, 2019. URL: https://tches.iacr.org/index.php/TCHES/article/view/8293, doi:10.13154/tches.v2019.i3.180-201.

[Lyu08] Vadim Lyubashevsky. Lattice-based identification schemes secure under active attacks. In Ronald Cramer, editor, *PKC 2008*, volume 4939 of *LNCS*, pages 162–179. Springer, Berlin, Heidelberg, March 2008. doi:10.1007/978-3-540-78440-1_10.

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Berlin, Heidelberg, December 2009. doi:10.1007/978-3-642-10366-7_35.

[Mac61] O. L. Macsorley. High-speed arithmetic in binary computers. *Proceedings of the IRE*, 49(1):67–91, 1961. doi:10.1109/JRPROC.1961.287779.

[Mac03] David J.C. MacKay. *Information theory, inference and learning algorithms*, chapter 26. Cambridge University Press, 2003.

[Mag21] Frédéric Magniez. Optimisation quantique : algorithme de Grover, estimateurs quantiques, chaînes de Markov quantiques, heuristiques quantiques.

Lecture on quantum algorithms, May 2021. College de France. URL: https://www.college-de-france.fr/site/frederic-magniez/course-2021-05-19-10h00.htm [cited 2021-07-01].

[MGTF19] Vincent Migliore, Benoît Gérard, Mehdi Tibouchi, and Pierre-Alain Fouque. Masking Dilithium - efficient implementation and side-channel evaluation. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19International Conference on Applied Cryptography and Network Security*, volume 11464 of *LNCS*, pages 344–362. Springer, Cham, June 2019. doi:10.1007/978-3-030-21568-2_17.

[Mic02] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In *43rd FOCS*, pages 356–365. IEEE Computer Society Press, November 2002. doi:10.1109/SFCS.2002.1181960.

[Mic18] Daniele Micciancio. On the hardness of learning with errors with binary secrets. *Theory of computing*, 14(13):1–17, 2018. doi:10.4086/toc.2018.v014a013.

[Min10] Hermann Minkowsli. *Geometrie der Zahlen*. Teubner, B. G., Leipzig und Berlin, 1910.

[MLA+22] Lars S Madsen, Fabian Laudenbach, Mohsen Falamarzi Askarani, Fabien Rortais, Trevor Vincent, et al. Quantum computational advantage with a programmable photonic processor. *Nature*, 606(7912):75–81, 2022. doi:10.1038/s41586-022-04725-x.

[MR04] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th FOCS*, pages 372–381. IEEE Computer Society Press, October 2004. doi:10.1109/FOCS.2004.72.

[MUTS22] Soundes Marzougui, Vincent Ulitzsch, Mehdi Tibouchi, and Jean-Pierre Seifert. Profiling side-channel attacks on Dilithium: A small bit-fiddling leak breaks it all. Cryptology ePrint Archive, Report 2022/106, 2022. URL: https://eprint.iacr.org/2022/106.

[MX23] Varun Maram and Keita Xagawa. Post-quantum anonymity of Kyber. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 3–35. Springer, Cham, May 2023. doi:10.1007/978-3-031-31368-4_1.

[NAB+17] Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions.

[NAB+19] Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-2-submissions`.

[NAB+20] Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions`.

[Nat16] National Institute of Standards and Technology. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. Call for proposals, December 2016. URL: `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf`.

[NDGJ21] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A side-channel attack on a masked IND-CCA secure Saber KEM implementation. *IACR TCHES*, 2021(4):676–707, 2021. URL: `https://tches.iacr.org/index.php/TCHES/article/view/9079`, `doi:10.46586/tches.v2021.i4.676-707`.

[NDJ21] Kalle Ngo, Elena Dubrova, and Thomas Johansson. Breaking masked and shuffled CCA secure saber KEM by power analysis. In *Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security*, ASHES '21, pages 51–61, New York, NY, USA, 2021. Association for Computing Machinery. `doi:10.1145/3474376.3487277`.

[Nil23] Alexander Nilsson. *Decryption Failure Attacks on Post-Quantum Cryptography*. Doctoral thesis, Department of electrical and information technology, Lund University, 2023.

[NP04] Olaf Neiße and Jürgen Pulkus. Switching blindings with a view towards IDEA. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 230–239. Springer, Berlin, Heidelberg, August 2004. `doi:10.1007/978-3-540-28632-5_17`.

[NS01] Phong Q. Nguyen and Jacques Stern. The two faces of lattices in cryptology. In Joseph H. Silverman, editor, *Cryptography and Lattices – CaLC 2001*, pages 146–180, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. `doi:10.1007/3-540-44670-2_12`.

[NUN+23] Pascal Nasahl, Martin Unterguggenberger, Rishub Nagpal, Robert Schilling, David Schrammel, and Stefan Mangard. SCFI: State machine control-flow hardening against fault attacks. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2023. doi:10.23919/DATE56975.2023.10137038.

[Nyb94] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 55–64. Springer, Berlin, Heidelberg, May 1994. doi:10.1007/3-540-48285-7_6.

[OSPG18] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2-secure masked Ring-LWE implementations. *IACR TCHES*, 2018(1):142–174, 2018. URL: https://tches.iacr.org/index.php/TCHES/article/view/836, doi:10.13154/tches.v2018.i1.142-174.

[Oun24] Mike Ounsworth. Updates on pre-hash for FIPS 204 and 205. Public posts on NIST pqc-forum Google Group, May 2024. URL: https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/JKMh0D0pa30/m/vOSEtJ-lAAAJ.

[PAA+19] Thomas Pöppelmann, Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Peter Schwabe, Douglas Stebila, Martin R. Albrecht, Emmanuela Orsini, Valery Osheter, Kenneth G. Paterson, Guy Peer, and Nigel P. Smart. NewHope. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-2-submissions.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 333–342. ACM Press, May / June 2009. doi:10.1145/1536414.1536461.

[Per24] Ray A. Perlner. Official comment on FIPS 203 ipd: seed as decapsulation key. Public post on NIST pqc-forum Google Group, May 2024. URL: https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/5CT4NC_6zRI/m/KyFx0sapAgAJ.

[PK20] John Preskill and Alexei Kitaev. Lecture notes on quantum computation. California Institute of Technology, 1997-2020. URL: http://theory.caltech.edu/~preskill/ph219/index.html [cited 2021-04-26].

[Pla21] Thomas Plantard. Efficient word size modular arithmetic. *IEEE Transactions on Emerging Topics in Computing*, 9(3):1506–1518, 2021. doi:10.1109/TETC.2021.3073475.

[PM19] Alice Pellet-Mary. *On ideal lattices and the GGH13 multilinear map*. PhD thesis, Université de Lyon, Lyon, France, October 2019. URL: https://theses.hal.science/tel-02337930.

[POG15] Thomas Pöppelmann, Tobias Oder, and Tim Güneysu. High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 346–365. Springer, Cham, August 2015. doi:10.1007/978-3-319-22174-8_19.

[PP19a] Chris Peikert and Zachary Pepin. Algebraically structured LWE, revisited. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 1–23. Springer, Cham, December 2019. doi:10.1007/978-3-030-36030-6_1.

[PP19b] Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In Peter Schwabe and Nicolas Thériault, editors, *LATINCRYPT 2019*, volume 11774 of *LNCS*, pages 130–149. Springer, Cham, October 2019. doi:10.1007/978-3-030-30530-7_7.

[PP21] Peter Pessl and Lukas Prokop. Fault attacks on CCA-secure lattice KEMs. *IACR TCHES*, 2021(2):37–60, 2021. URL: https://tches.iacr.org/index.php/TCHES/article/view/8787, doi:10.46586/tches.v2021.i2.37-60.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 513–533. Springer, Cham, September 2017. doi:10.1007/978-3-319-66787-4_25.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, Berlin, Heidelberg, May 2013. doi:10.1007/978-3-642-38348-9_9.

[PZ03] John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *Quantum Information and Computation*, 3(4):317–344, July 2003. doi:10.5555/2011528.2011531.

[QCZ+21] Yue Qin, Chi Cheng, Xiaohan Zhang, Yanbin Pan, Lei Hu, and Jintai Ding. A systematic approach and analysis of key mismatch attacks on lattice-based NIST candidate KEMs. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 92–121. Springer, Cham, December 2021. doi:10.1007/978-3-030-92068-5_4.

[RAC+24] Ben W. Reichardt, David Aasen, Rui Chao, Alex Chernoguzov, Wim van Dam, John P. Gaebler, Dan Gresh, Dominic Lucchetti, Michael Mills, Steven A. Moses, Brian Neyenhuis, Adam Paetznick, Andres Paz, Peter E. Siegfried, Marcus P. da Silva, Krysta M. Svore, Zhenghan Wang, and Matt Zanner. Demonstration of quantum computation and error correction with a tesseract code. arXiv prepreint, paper 2409.04628, September 2024. URL: `https://arxiv.org/abs/2409.04628`.

[Rav23] Prasanna Ravi. New vulnerability announcement on variable time Kyber implementations. Public post on NIST pqc-forum Google Group, December 2023. URL: `https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/ldX0ThYJuBo/m/ovODsdY7AwAJ`.

[RBRC22] Prasanna Ravi, Shivam Bhasin, Sujoy Sinha Roy, and Anupam Chattopadhyay. On exploiting message leakage in (few) NIST PQC candidates for practical message recovery attacks. *IEEE Transactions on Information Forensics and Security*, 17:684–699, 2022. `doi:10.1109/TIFS.2021.3139268`.

[RCDB23] Prasanna Ravi, Anupam Chattopadhyay, Jan Pieter D'Anvers, and Anubhab Baksi. Side-channel and fault-injection attacks over lattice-based post-quantum schemes (Kyber, Dilithium): Survey and new results. *ACM Transactions on Embedded Computing Systems*, 23(2), mar 2023. `doi:10.1145/3603170`.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. `doi:10.1145/1060590.1060603`.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6), sep 2009. `doi:10.1145/1568318.1568324`.

[RPBC20] Prasanna Ravi, Romain Poussier, Shivam Bhasin, and Anupam Chattopadhyay. On configurable SCA countermeasures against single trace attacks for the NTT. In Lejla Batina, Stjepan Picek, and Mainack Mondal, editors, *Security, Privacy, and Applied Cryptography Engineering*, pages 123–146, Cham, 2020. Springer International Publishing. `doi:10.1007/978-3-030-66626-2_7`.

[RPD09] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-order masking and shuffling for software implementations of block ciphers. In Christophe Clavier and Kris Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 171–188. Springer, Berlin, Heidelberg, September 2009. `doi:10.1007/978-3-642-04138-9_13`.

[RRB+19] Prasanna Ravi, Debapriya Basu Roy, Shivam Bhasin, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Number "not used" once - practical fault attack on pqm4 implementations of NIST candidates. In Ilia Polian and Marc Stöttinger, editors, *COSADE 2019*, volume 11421 of *LNCS*, pages 232–250. Springer, Cham, April 2019. doi:10.1007/978-3-030-16350-1_13.

[RRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR TCHES*, 2020(3):307–335, 2020. URL: https://tches.iacr.org/index.php/TCHES/article/view/8592, doi:10.13154/tches.v2020.i3.307-335.

[RRD+23] Gokulnath Rajendran, Prasanna Ravi, Jan-Pieter D'Anvers, Shivam Bhasin, and Anupam Chattopadhyay. Pushing the limits of generic side-channel attacks on LWE-based KEMs - parallel PC oracle attacks on Kyber KEM and beyond. *IACR TCHES*, 2023(2):418–446, 2023. doi:10.46586/tches.v2023.i2.418-446.

[RRVV15] Oscar Reparaz, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. A masked ring-LWE implementation. In Tim Güneysu and Helena Handschuh, editors, *CHES 2015*, volume 9293 of *LNCS*, pages 683–702. Springer, Berlin, Heidelberg, September 2015. doi:10.1007/978-3-662-48324-4_34.

[RY10] Thomas Ristenpart and Scott Yilek. When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography. In *NDSS 2010*. The Internet Society, February / March 2010.

[Saa18] Markku-Juhani O. Saarinen. Arithmetic coding and blinding countermeasures for lattice signatures - engineering a side-channel resistant post-quantum signature scheme with compact signatures. *Journal of Cryptographic Engineering*, 8(1):71–84, April 2018. doi:10.1007/s13389-017-0149-6.

[SAB+20] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[SE94] Claus Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66(1):181–199, August 1994. doi:10.1007/BF01581144.

[SF07] Dan Shumow and Niels Ferguson. On the possibility of a back door in the NIST SP800-90 Dual Ec PRNG. CRYPTO 2007 Rump Session, August 2007. URL: `http://rump2007.cr.yp.to/15-shumow.pdf`.

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979. `doi:10.1145/359168.359176`.

[Sho94] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. `doi:10.1109/SFCS.1994.365700`.

[Sho95] Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Physical Review A*, 52:R2493–R2496, Oct 1995. `doi:10.1103/PhysRevA.52.R2493`.

[SKB22] Hauke Malte Steffen, Lucie Johanna Kogelheide, and Timo Bartkewitz. In-depth analysis of side-channel countermeasures for CRYSTALS-Kyber message encoding on ARM Cortex-M4. In Vincent Grosso and Thomas Pöppelmann, editors, *Smart Card Research and Advanced Applications*, pages 169–188, Cham, 2022. Springer International Publishing. `doi:10.1007/978-3-030-97348-3_10`.

[Skl60] Jack Sklansky. Conditional-sum addition logic. *IRE Transactions on Electronic Computers*, EC-9(2):226–231, 1960. `doi:10.1109/TEC.1960.5219822`.

[SLKG23] Hauke Malte Steffen, Georg Land, Lucie Johanna Kogelheide, and Tim Güneysu. Breaking and protecting the crystal: Side-channel analysis of dilithium in hardware. In Thomas Johansson and Daniel Smith-Tone, editors, *Post-Quantum Cryptography - 14th International Workshop, PQCrypto 2023*, pages 688–711. Springer, Cham, August 2023. `doi:10.1007/978-3-031-40003-2_25`.

[SMG15] Tobias Schneider, Amir Moradi, and Tim Güneysu. Arithmetic addition over Boolean masking - towards first- and second-order resistance in hardware. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *ACNS 15International Conference on Applied Cryptography and Network Security*, volume 9092 of *LNCS*, pages 559–578. Springer, Cham, June 2015. `doi:10.1007/978-3-319-28166-7_27`.

[SMRM19] Rajat Sadhukhan, Paulson Mathew, Debapriya Basu Roy, and Debdeep Mukhopadhyay. Count your toggles: a new leakage model for pre-silicon power analysis of crypto designs. *Journal of Electronic Testing*, 35(5):605–619, 2019. `doi:10.1007/s10836-019-05826-8`.

[SPH22] Bo-Yeon Sim, Aesun Park, and Dong-Guk Han. Chosen-ciphertext clustering attack on CRYSTALS-KYBER using the side-channel leakage of Barrett reduction. *IEEE Internet of Things Journal*, 9(21):21382–21397, 2022. doi:10.1109/JIOT.2022.3179683.

[SPOG19] Tobias Schneider, Clara Paglialonga, Tobias Oder, and Tim Güneysu. Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 534–564. Springer, Cham, April 2019. doi:10.1007/978-3-030-17259-6_18.

[SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 617–635. Springer, Berlin, Heidelberg, December 2009. doi:10.1007/978-3-642-10366-7_36.

[TH08] Stefan Tillich and Christoph Herbst. Attacking state-of-the-art software countermeasures-a case study for AES. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES 2008*, volume 5154 of *LNCS*, pages 228–243. Springer, Berlin, Heidelberg, August 2008. doi:10.1007/978-3-540-85053-3_15.

[TPD21] Chengdong Tao, Albrecht Petzoldt, and Jintai Ding. Efficient key recovery for all HFE signature variants. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 70–93, Virtual Event, August 2021. Springer, Cham. doi:10.1007/978-3-030-84242-0_4.

[TUX+23] Yutaro Tanaka, Rei Ueno, Keita Xagawa, Akira Ito, Junko Takahashi, and Naofumi Homma. Multiple-valued plaintext-checking side-channel attacks on post-quantum KEMs. *IACR TCHES*, 2023(3):473–503, 2023. doi:10.46586/tches.v2023.i3.473-503.

[UXT+22] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of re-encryption: A generic power/EM analysis on post-quantum KEMs. *IACR TCHES*, 2022(1):296–322, 2022. doi:10.46586/tches.v2022.i1.296-322.

[VBDK+21] Michiel Van Beirendonck, Jan-Pieter D'Anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A side-channel-resistant implementation of SABER. *ACM Journal on Emerging Technologies in Computing Systems*, 17(2), April 2021. doi:10.1145/3429983.

[VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 282–296. Springer, Berlin, Heidelberg, December 2014. doi:10.1007/978-3-662-45611-8_15.

[VMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 740–757. Springer, Berlin, Heidelberg, December 2012. doi:10.1007/978-3-642-34961-4_44.

[WT90] B.W.Y. Wei and C.D. Thompson. Area-time optimal adder design. *IEEE Transactions on Computers*, 39(5):666–675, 1990. doi:10.1109/12.53579.

[WZJ20] Ke Wang, Zhenfeng Zhang, and Haodong Jiang. Key recovery under plaintext checking attack on LAC. In Khoa Nguyen, Wenling Wu, Kwok Yan Lam, and Huaxiong Wang, editors, *Provable and Practical Security*, ProvSec 2020, pages 381–401, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-62576-4_19.

[XIU+21] Keita Xagawa, Akira Ito, Rei Ueno, Junko Takahashi, and Naofumi Homma. Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 33–61. Springer, Cham, December 2021. doi:10.1007/978-3-030-92075-3_2.

[XL21] Yufei Xing and Shuguo Li. A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA. *IACR TCHES*, 2021(2):328–356, 2021. URL: https://tches.iacr.org/index.php/TCHES/article/view/8797, doi:10.46586/tches.v2021.i2.328-356.

[Yil10] Scott Yilek. Resettable public-key encryption: How to encrypt on a virtual machine. In Josef Pieprzyk, editor, *CT-RSA 2010*, volume 5985 of *LNCS*, pages 41–56. Springer, Berlin, Heidelberg, March 2010. doi:10.1007/978-3-642-11925-5_4.

[YJ00] Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, 2000. doi:10.1109/12.869328.

[ZBT19] Timo Zijlstra, Karim Bigou, and Arnaud Tisserand. FPGA implementation and comparison of protections against SCAs for RLWE. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *INDOCRYPT 2019*, volume 11898 of *LNCS*, pages 535–555. Springer, Cham, December 2019. doi:10.1007/978-3-030-35423-7_27.

List of Figures

List of Tables

List of Algorithms

A Quantum computing

A.1 Introductory considerations

The laws of physics as we have known them until the beginning of the twentieth century are known as *classical*, and they are very accurate at a macroscopic scale, down to the molecular level, as embodied in Newton's laws of mechanics and Maxwell's laws of electromagnetics. However, at and below the atomic scale, they become entirely incorrect, which led to the development of so-called quantum mechanics to explain the behavior observed at such scales.

A.1.1 Information is physical

As presented in [Joz19], information is generally represented as one or more Boolean variables[1], whose combination of values can be considered as a label selecting one of several possible answers to a question. To keep things simple, if the question expects integral answers, then the Boolean variables hold a bit string that corresponds to the base-two representation of an integer. Performing computations consists in transforming this information by passing it through a sequence of logic gates, that is, local transformations of the bit string. To make these definitions useful, they have to be implemented, so that such objects can be used in practice. We thus choose to associate Boolean values with two states of a system that can be reliably distinguished through measurement. Then, computations can be performed through physical transformations of the system.

Another reason is given by Preskill and Kitaev [PK20, Chapter 1] for the importance of physical considerations in the handling of information. This reason lies in thermodynamics, that show[2] that the handling of information (however it is represented) must involve entropy, and in particular, the erasure or overwriting of information results in an decrease of entropy: it is a dissipative process.

[1] Using binary arithmetic as the base of computations is convenient in digital processing, but strictly mandatory. In fact, existing digital quantum computers use Boolean logic, but using more values per variable would be possible [Gra21, Part 1].

[2] Maxwell's-demon thought experiment presented a setting in which the thermodynamic entropy of an isolated system could apparently be decreased, by separating a homogeneous gas into two containers: one containing the hottest molecules, and the other containing the coldest. This setting was thought to violate the second law of thermodynamics — that the entropy of an isolated system cannot decrease. However, Leó Szilárd and Léon Brillouin later solved the paradox by pointing out that such process involves the need for storing some information about the molecules, which has an associated entropy. Consequently, the system comprised of the gas and the acquired information does obey the second law of thermodynamics.

A.1.2 Quantum supremacy

Considering this physical reality of information, a straightforward consequence is that processing, i.e. the transformation of information, must follow the laws obeyed by the physical representation of information. Thus, computations using classical information (i.e. represented by classical properties of a physical system) can be expected to be essentially different from computations on quantum information (represented in quantum properties of a physical system).

This difference is not in the notion of *computability*, because anything that can be computed by a quantum system can also, in theory, be computed to an arbitrary precision by a classical system, if only by simulating classically the quantum computer. However, what does differ is the practicality and scalability of such computations, that is, their complexity. It can be seen in [Joz19] that there exist some quantum algorithms that solve in polynomial time some problems for which no classical polynomial-time solver is known. Quantum supremacy — a term coined by John Preskill to designate the ability of a quantum computer to complete a task that no classical computer can complete in reasonable time — has been claimed to be attained in 2019 on Google's *Sycamore* computer. Arute et al. reported in [AAB+19c] that they had performed in 200 seconds the sampling of a quantum pseudorandom number generator, and that a classical supercomputer would have needed 10 000 years to do the same. Since then, the claim has been repeated several times in progressively more natural experiments (*e.g.* [MLA+22, KNR+24]).

A.2 Notations used in this chapter

This section and the following are mainly based on [Joz19], but the notations in use are standard in quantum physics.

A.2.1 Hilbert spaces and the Dirac notation

The state of a quantum system can be seen as a vector in an m-dimensional[3] complex vector space V endowed with an inner product $\langle \cdot \,|\, \cdot \rangle$ — a Hilbert space. Every vector of V will be denoted as a *ket* $|x\rangle$, that can be represented as a column vector in an orthonormal basis of V.

We can furthermore consider, for any ket $|x\rangle$, its dual $\langle x| = |x\rangle^\dagger$, where notation $\langle \cdot|$ is known as a *bra*. In an orthonormal basis, the dual of a vector, denoted by the dagger †, is represented as the transposition of its coordinate-wise complex conjugate. This notation allows us to write the inner product of any two vectors $(|x\rangle, |y\rangle) \in V^2$ as $|x\rangle^\dagger |y\rangle = \langle x| \, |y\rangle = \langle x \,|\, y\rangle$.

[3] In the most general setting, quantum systems have an infinite-dimensional state space, and states are represented by *wave functions* instead of vectors. However, in the formalism of (digital) quantum computing, finite-dimensional spaces are considered.

A.2.2 Tensor product of two Hilbert spaces

Given two spaces V and W, their tensor product $V \otimes W$ is a Hilbert space as well. The dimension of the product is the product of the dimensions of V and W. If two kets $|x\rangle \in V$ and $|y\rangle \in W$ are expressed as $|x\rangle = x_1 |v_1\rangle + \cdots + x_n |v_n\rangle$ and $|y\rangle = y_1 |w_1\rangle + \cdots + y_m |w_m\rangle$ in orthonormal bases of V and W, then their tensor product $|x\rangle \otimes |y\rangle \in V \otimes W$, when considered on the orthonormal basis $(|v_i\rangle \otimes |w_j\rangle)_{i,j}$, has coordinates

$$|x\rangle \otimes |y\rangle = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \otimes \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} x_1 \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \\ \vdots \\ x_n \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \end{pmatrix}.$$

Note that the tensor product is not commutative, and that its symbol $\otimes$ is often omitted, simply expressing the above as $|x\rangle\,|y\rangle$. The tensor product in the dual space will be noted in the same order: by writing as a^* the complex conjugate of a,

$$(|x\rangle\,|y\rangle)^\dagger = \langle x|\,\langle y| = \begin{pmatrix} x_1^* \begin{pmatrix} y_1^* & \cdots & y_m^* \end{pmatrix} & \cdots & x_n^* \begin{pmatrix} y_1^* & \cdots & y_m^* \end{pmatrix} \end{pmatrix}.$$

A very important property of $V \otimes W$ is that only a small fraction of its kets can be expressed as tensor products, i.e. $|z\rangle \in V \otimes W$ cannot generally be expressed as $|x\rangle\,|y\rangle$ with $|x\rangle \in V$ and $|y\rangle \in W$. The most general expression for $|z\rangle$ is a linear combination of the nm tensor products of the basis vectors of V and W: $|z\rangle = \sum_{i,j} z_{i,j} |v_i\rangle\,|w_j\rangle$. Vectors of $V \otimes W$ that are not product vectors will be called *entangled vectors*.

A.2.3 Operators

Given a ket $|x\rangle \in V$ and a bra $\langle y| \in V^*$, we define the *ket-bra*

$$|x\rangle\,\langle y| = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \begin{pmatrix} y_1 & \cdots & y_n \end{pmatrix} = \begin{pmatrix} x_1 y_1 & \cdots & x_1 y_n \\ \vdots & \ddots & \vdots \\ x_n y_1 & \cdots & x_n y_n \end{pmatrix}$$

such that for any ket $|z\rangle \in V$, $(|x\rangle\,\langle y|)\,|z\rangle = |x\rangle\,\langle y|\,|z\rangle = \langle y\,|\,z\rangle\,|x\rangle$. We call $|x\rangle\,\langle y|$ a rank-one operator. More generally, any linear map on V can be expressed as a linear combination of the $|v_i\rangle\,\langle v_j|$, where the $|v_i\rangle$ form an orthonormal basis of V. We thus express any operator as an $n \times n$ matrix in this basis.

Two special cases are of importance. A *unitary operator* is a U such that $U^\dagger = U^{-1}$, that is, an operator whose matrix (in an orthonormal basis) is orthogonal. A *projector* is an operator P such that $P^2 = P$. In particular, any rank-one projector is of the form $P = |x\rangle\,\langle x|$, called the projector onto $|x\rangle$.

Given operators A and B on Hilbert spaces V and W respectively, one can define the tensor product $A \otimes B$ by its action on the product basis of $V \otimes W$: if $(|v_i\rangle)$ and $(|w_j\rangle)$ are orthonormal bases of V and W respectively, we define $(A \otimes B)\,|v_i\rangle\,|w_j\rangle = (A\,|v_i\rangle) \otimes (B\,|w_j\rangle)$ for all i, j, and extend linearly.

A.2.4 Partial inner products

We have seen that any ket $|x\rangle \in V$ can be used to compute inner products in V through its associated bra: $\langle x|\,|v\rangle = \langle x\,|\,v\rangle$. We furthermore use it to define a *partial inner product* on $V \otimes W$: for any $|x\rangle, |v\rangle \in V$ and $|w\rangle \in W$, $\langle x|\,(|v\rangle \otimes |w\rangle) = \langle x\,|\,v\rangle\,|w\rangle \in W$ and we extend linearly to all vectors of $V \otimes W$. The definition is alike when $|x\rangle \in W$, giving a result in V.

In case it is not clear from context, subscripts can be added to the kets to indicate which part of the tensor product the partial inner product applies to. This is particularly important when taking the inner product of a space with itself ($V = W$), e.g. ${}_1\langle x|\,|v\rangle_1\,|w\rangle_2 = \langle x\,|\,v\rangle\,|w\rangle$, but ${}_2\langle x|\,|v\rangle_1\,|w\rangle_2 = \langle x\,|\,w\rangle\,|v\rangle$.

A.3 Principles of quantum mechanics

In the setting introduced above, quantum mechanics respect four main principles.

Principle A.1 (State of an isolated quantum system)**.** *The state $|\psi\rangle$ of an isolated quantum system is represented by a unit vector (i.e. of norm 1) in a Hilbert space V.*

Space V is called the state space of the system, although only its unit vectors are valid states.

If we consider a bidimensional state space, we can choose an orthonormal basis thereof: let us note it $(|0\rangle, |1\rangle)$. Then, the states of the system are exactly of the form $\alpha\,|0\rangle + \beta\,|1\rangle$, where $(\alpha, \beta) \in \mathbb{C}^2$ with $|\alpha|^2 + |\beta|^2 = 1$.

Principle A.2 (State of a complex quantum system)**.** *Given two quantum systems S_1 and S_2 with state spaces V and W respectively, the composite system in which S_1 and S_2 are joined has state space $V \otimes W$.*

That second principle is extremely interesting, because it means that two quantum systems allowed to interact with one another are much more complex than the two in isolation: the two systems are entangled, i.e. they cannot be described independently of each other. In fact, increasing linearly the size of a quantum system makes its state space increase exponentially.

Principle A.3 (Evolution of an isolated quantum system)**.** *Any finite-time evolution of an isolated quantum system can be described as a unitary operator acting on its state space.*

It is very important to note that this principle only holds if the quantum system is isolated from the environment. In fact, any external perturbation from the environment

can alter the state of the system, and in a non-unitary way. This is the case in particular for measurements, as stated below.

For the last principle, we consider a quantum system S with state space V of dimension n, an orthonormal basis $\mathcal{B} = (|1\rangle, \ldots, |n\rangle)$ of V, and a set of d mutually orthogonal subspaces E_i of V, such that $V = E_1 \oplus \cdots \oplus E_d$. We assume that the state of the system is $|\psi\rangle = a_1 |1\rangle + \cdots + a_n |n\rangle$, where $a_i \in \mathbb{C}$ and $\sum_i |a_i|^2 = 1$. We denote by Π_i the projection operator onto subspace E_i.

Principle A.4 (Measurement of a quantum system)**.** *Measuring the state of a quantum system is a probabilistic process that gives only limited information and alters the state of the system:*

- *A* complete projective measurement *or* von Neumann measurement *of S over the basis $\mathcal{B}$ gives result $r \in [\![1, n]\!]$ with probability $|a_r|^2$, and forces S into state $|r\rangle$.*

- *An* incomplete projective measurement *of S relative to the orthogonal decomposition $(E_1, \ldots, E_d)$ gives result $r \in [\![1, d]\!]$ with probability $|\Pi_r |\psi\rangle|^2 = \langle\psi| \Pi_r |\psi\rangle$, and forces S into state $\frac{|p\rangle}{||p\rangle|}$ where $|p\rangle = \Pi_r |\psi\rangle$.*

Informally, the above means that the more we learn of the state of S, the more we alter it — but the post-measurement state of the system conforms to the outcome of the measurement, up to the degrees of freedom preserved by an incomplete measurement.

The measurement basis needs not be fixed, any orthonormal basis may be used. In fact, it is even possible to measure a system relative to an arbitrary basis $\mathcal{B}'$, while actually performing the measurement in a different basis $\mathcal{B}$: to do so, it suffices to apply to S, before the measurement, the operator that changes basis $\mathcal{B}'$ into basis $\mathcal{B}$.

A different formalism for measurement is used in [PK20, chapter 2], namely that of *observables* — self-adjoint operators that project the measured system onto one of their eigenspaces, and give as measurement outcome the corresponding eigenvalue. This equivalent formalism is not developed here.

Note that the *global phase* of a state is irrelevant (not measurable and of no influence under unitary evolution), i.e. states $|\psi\rangle$ and $e^{i\theta}|\psi\rangle$ are exactly equivalent. However, *relative phases* in a linear combination are very significant, e.g. $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ is not the same state as $\frac{1}{\sqrt{2}}(|0\rangle + e^{i\theta}|1\rangle)$ for $\theta \neq 0$.

No-cloning theorem An implicit consequence of the other rules of quantum information is the *no-cloning theorem* [Joz19, §3.2]. This theorem states that, given a system $\mathcal{S}$ in an unknown state $|\psi\rangle$, and a second system $\mathcal{T}$ in a known state independent from the state of $\mathcal{S}$, it is not possible to bring $\mathcal{T}$ into state $|\psi\rangle$ without altering the state of $\mathcal{S}$. In other words, it is not possible to make independent copies of an unknown quantum state. This property of quantum information is in total contrast to classical information, where information cloning can be and is routinely done.

A.4 Storage and processing of quantum information

A.4.1 Preparation and use of qubits

In classical digital computing, the atom of information is the bit, that is, a digit taking values 0 or 1. Similarly, we base quantum information on the *qubit* (quantum bit), a vector in a two-dimensional Hilbert space, and we label $|0\rangle$ and $|1\rangle$ an orthonormal basis of that space [PK20]. The analogy becomes clear when we consider measuring a qubit relative to that basis: the outcome is a bit, with $|0\rangle$ measured to 0 and $|1\rangle$ measured to 1. More generally, $a\,|0\rangle + b\,|1\rangle$ is measured to 0 with probability $|a|^2$, and to 1 with probability $|b|^2$.

That construction becomes interesting when using strings of qubits, because an n-qubit string lives in a 2^n-dimensional Hilbert space. We denote a basis of this space by $|0\ldots00\rangle$, $|0\ldots01\rangle$, $|0\ldots10\rangle$, up to $|1\ldots1\rangle$, corresponding to each qubit (in order) independently taking value $|0\rangle$ or $|1\rangle$. We can also denote these basis vectors by the integer corresponding to each bitstring: $|\overline{0}\rangle$, $|\overline{1}\rangle$, up to $|\overline{2^n-1}\rangle$. Then, any state in this space can be written as

$$|\psi\rangle = \sum_{i=0}^{2^n-1} a_i\,|\overline{i}\rangle\ ,$$

where the a_i are complex numbers such that $\sum |a_i|^2 = 1$. Measuring $|\psi\rangle$ in the canonical basis gives result i with probability $|a_i|^2$.

A quantum computation is performed as follows: first, the qubits are prepared in a known initial state, for instance: $|00\ldots0\rangle$. Then, the computation is performed through a sequence of unitary operations, either fixed or (more interestingly) programmable through classical stimuli. Finally, the result is read through a measurement relative to the appropriate basis.

A.4.2 Quantum parallelism

The very motivation for quantum computing is known as *quantum parallelism* [PK20]. Due to the possibility for qubits to be in a superposition of states, that is, in a linear combination of several basis vectors, we can perform a computation on multiple inputs at a time. In fact, with n qubits, we can compute any unitary transformation X of the qubits *on all 2^n possible input values at a time*, as

$$X\left(a_0\,|0\rangle + \cdots + a_{2^n-1}\,|2^n-1\rangle\right) = a_0 X\,|0\rangle + \cdots + a_{2^n-1} X\,|2^n-1\rangle\ .$$

The only restriction to this, is that we may only read one result, given that a measurement can only return a single bit of information per qubit. The crux of quantum computing lies in being able to efficiently compute a *global* characteristic of a given function, by computing it simultaneously for all possible inputs, and aggregating all outputs into a single quantity before measurement. We should also remember that by the nature of measurement, quantum algorithms are intrinsically probabilistic. If a given computation has several possible outcomes, running this computation several

times in the exact same setting will give different results. However, it is not necessarily an issue, for instance when the result can be verified (but not found) efficiently by a classical computer.

A.4.3 Need for reversible computations

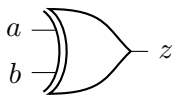
In classical computing, most logic gates are irreversible, which means that the value of their inputs cannot be uniquely determined from their outputs. This is true, in particular, for logic gates that have fewer outputs than inputs. Consider, for instance, the two-input XOR (exclusive-or) gate: a logic-low output could either mean that both inputs were logic-low, or that both were logic-high: the gate is not reversible. If, additionally to the exclusive-or, one of the input bits is passed unchanged to the output, the resulting CNOT (controlled-not) gate becomes reversible. Both gates are shown in fig. A.1.

We have seen that it is important that a quantum system be as isolated as possible from its environment, otherwise it can be projected into a classical state, and all the benefits of quantum computation are lost. Given that irreversible gates dissipate some energy (they destroy some information, and we have seen in appendix A.1.1 that the erasure of information necessarily involves to expend some energy), this energy must go somewhere, namely: in the environment. Consequently, only reversible computations can fully preserve the quantum properties of a system.

However, as stated by Childs [Chi21], this restriction to reversible operations is not a real limitation, because any logic operation can be made reversible. Consider, for instance, a black-box function $g\colon \{0,1\}^n \to \{0,1\}^m$. Then, a reversible equivalent of this function (modulo the extension of input and output lengths) is

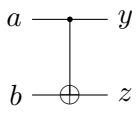
$$\begin{aligned} \tilde{g}\colon \{0,1\}^{n+m} &\to \{0,1\}^{n+m} \\ (x,y) &\mapsto (x, g(x) \oplus y)\,, \end{aligned}$$

which can be used in place of g, by supplying 0^m as its second argument.



a	b	z
0	0	0
0	1	1
1	1	0
1	0	1

(a) XOR gate



a	b	y	z
0	0	0	0
0	1	0	1
1	1	1	0
1	0	1	1

(b) CNOT gate

Figure A.1: Symbol and truth table of the XOR and CNOT gates

A.4.4 Uncomputing temporary results

Any step of a quantum computation may generate some garbage, that is, some outputs that are useless for the rest of the computation. If the storage space taken up by this

garbage is needed in later steps, it is not acceptable to simply reset the corresponding qubits to zero, because this operation would not be reversible, thus having an unwanted influence on the whole quantum state (including the qubits that hold valuable information). Gray [Gra21] shows that it is possible to reset these temporary qubits to their initial state of 0, by *uncomputing* the function that generated this garbage, after having stored the useful result elsewhere. An example of this is shown in fig. A.2.

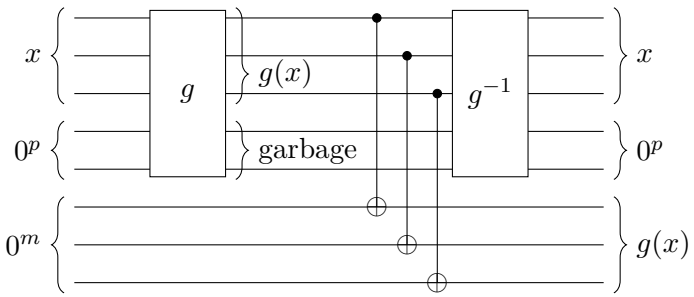


Figure A.2: Clean computation of $g(x)$ by uncomputing temporary results

A.5 Error correction

A drawback of current quantum computers is that they quickly accumulate errors, as a direct consequence of operators being reversible [PK20]. In fact, classical digital computers achieve robustness to perturbations through their dissipative nature. Every logic gate corrects for any small perturbation at its input, by dissipating this error as heat. We can see in fig. A.3 that the nonlinearity in a classical NOT gate (implemented, for instance, using a CMOS circuit) attenuates any error present at its input. We do not have this possibility with quantum computers, because all transformations must be unitary.

In classical computing, it is nonetheless possible for errors to occur in memory cells, in the form of bit flips (a bit changing from 0 to 1 or vice-versa). To minimize the impact of

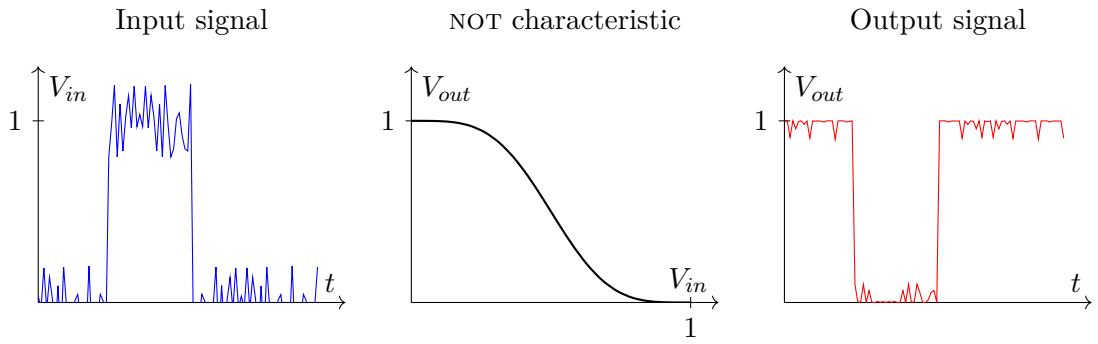


Figure A.3: Noise attenuation in a classical NOT gate

such errors, it is possible to store redundant information, so that the errors can be detected and corrected. In fact, it is possible to use similar techniques in quantum computing. We will only present here the main principles backing quantum error correction, we refer the reader to [PK20] for details.

The operating principle of quantum error correction is to spread the information contained in a *logical* qubit over several *physical* qubits, much like classical error-correction codes, but with the difference that in the quantum case, error correction takes up several times more room that raw qubits: for example, in the encoding proposed by Shor [Sho95], nine physical qubits are needed to encode a single logical qubit.

The most straightforward classical error-correction scheme is to triplicate every bit of information, and to perform a majority voting on the three bits to determine which value they encode. If one bit does not agree with the two others, it is flipped to correct the error. This scheme cannot work for quantum information, for two main reasons: first, due to the no-cloning theorem (see appendix A.3), it is not possible to create these three copies in the first place; secondly, measuring the three qubits to check for errors would irreparably damage the information they hold. Instead, we encode information into correlations between logical qubits — so that a single-qubit error does not change the information that is represented – and determine whether errors have occurred, not through *measurements*, but through *comparisons*: partial measurements over several qubits at once. Performing comparisons allows to detect errors without damaging valuable information, because the actual value represented by the qubit is not measured: only the mismatch between qubits is. With this information, the erroneous qubit can be corrected — still without measuring its value.

Classical digital information is discrete (usually Boolean), so only large errors can occur, in the shape of bit-flips. The quantum case is more complicated, because quantum information is continuous, so small errors can occur: for instance, the state of a qubit can change from $|0\rangle$ to $|0\rangle + \varepsilon\,|1\rangle$, for ε a small complex number[4]. Quantum error correction corrects both small and large errors. Large errors are detected and corrected through the application of a unitary operation on the erroneous qubit. For small errors, two cases can occur: either the partial measurement of the error projects the erroneous qubit back to its correct state, silently correcting the error; or the measurement projects the small error into a large one and detects it, allowing it to be corrected with a unitary operator.

A.6 Quantum algorithms

Quantum computation can be used to solve some problems more efficiently (in terms of asymptotic complexity) than purely classical algorithms. In general, it is not interesting to use quantum computation exclusively to solve such problems, because some steps of the solution can be performed efficiently using a classical computer. Thus, only the steps that cannot be done efficiently in the classical setting are performed quantumly. A comprehensive list of all currently known quantum algorithms, together with their

[4]For simplicity, we will not always normalize state vectors, and may write $|0\rangle + \varepsilon\,|1\rangle$ instead of $(1 + |\varepsilon|^2)^{-1/2}(|0\rangle + \varepsilon\,|1\rangle)$.

description and their speedup over the equivalent classical algorithms, is maintained in the *Quantum Algorithm Zoo* [Jc24].

We first present Simon's algorithm to give a simple example of how quantum computing can be much more powerful than classical computing, then discuss two well-known quantum algorithms with applications in cryptanalysis — Shor's algorithm being the very trigger that led to the development of post-quantum cryptography. All three algorithms are described in detail by Preskill and Kitaev [PK20, chapter 6].

A.6.1 Solving the Deutsch-Josza problem

Suppose we are given an unknown function $f\colon \{0,1\}^n \to \{0,1\}$, with the guarantee that is is either constant, or balanced, which means it maps exactly half of its inputs to 0 and the other half to 1. The goal is to determine whether f is constant or balanced. To solve this problem in the classical setting, the only solution is to apply f to all possible input strings, until f returns an output different from the previous ones (indicating it is balanced). If f is found to be constant on $2^{n-1}+1$ inputs, then it is constant. Thus, in the worst case, we need $2^{n-1}+1$ queries to f to determine with certainty whether it is constant or balanced.

There exists a quantum algorithm solving this problem with exponential speedup. Consider a black-box unitary operator U_f implementing f: for all $x \in [\![0, 2^n-1]\!]$, and $y \in \{0,1\}$, $U_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle$. Applying U_f to $|x\rangle |-\rangle$, where $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, gives

$$U_f |x\rangle |-\rangle = |x\rangle \frac{1}{\sqrt{2}}\Big(|f(x)\rangle - |1 \oplus f(x)\rangle \Big) = |x\rangle (-1)^{f(x)} |-\rangle .$$

Now, we prepare the $(n+1)$-qubit state that is the equally weighted superposition of all possible n-bit strings, followed by $|-\rangle$,

$$|\psi\rangle = \mathrm{H}^{\otimes(n+1)} |0\rangle^{\otimes n} |1\rangle = \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle |-\rangle ,$$

and apply U_f, then $\mathrm{H}^{\otimes n} \otimes \mathrm{I}$ on it, to obtain the following:

$$\begin{aligned} (\mathrm{H}^{\otimes n} \otimes \mathrm{I}) U_f |\psi\rangle |-\rangle &= (\mathrm{H}^{\otimes n} \otimes I) \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle (-1)^{f(x)} |-\rangle , \\ &= \underbrace{\frac{1}{2^{n/2}} \left(\mathrm{H}^{\otimes n} \sum_{x=0}^{2^n-1} (-1)^{f(x)} |x\rangle \right)}_{|f\rangle} |-\rangle . \end{aligned} \tag{A.1}$$

If f is constant such that $f(x) = c$ for all x, then the first n qubits in eq. (A.1) are given (omitting the normalization factor and the $(-1)^n$ global phase) by $|f\rangle = \mathrm{H}^{\otimes n} \sum_x |x\rangle =$

$|0\dots0\rangle$ because H is self-inverse. If, instead, f is balanced, then:

$$\begin{aligned}
|f\rangle &= \sum_{x=0}^{2^n-1} \frac{(-1)^{f(x)}}{2^{n/2}} \mathrm{H}^{\otimes n} |x\rangle \ , \\
&= \sum_{x=0}^{2^n-1} \frac{(-1)^{f(x)}}{2^{n/2}} \bigotimes_{i=0}^{n-1} \mathrm{H} |x_i\rangle \ , \qquad \textit{where } x = \textstyle\sum_{i=0}^{n-1} 2^i x_i \ , \\
&= \sum_{x=0}^{2^n-1} \frac{(-1)^{f(x)}}{2^{n}} \bigotimes_{i=0}^{n-1} \left(|0\rangle + (-1)^{x_i} |1\rangle \right) , \\
&= \sum_{x=0}^{2^n-1} \frac{(-1)^{f(x)}}{2^{n}} \sum_{y=0}^{2^n-1} \bigotimes_{i=0}^{n-1} (-1)^{x_i y_i} |y_i\rangle \ , \\
&= \sum_{x=0}^{2^n-1} \frac{(-1)^{f(x)}}{2^{n}} \sum_{y=0}^{2^n-1} (-1)^{\oplus(x \circ y)} |y\rangle \ , \\
&= \frac{1}{2^n} \sum_{y=0}^{2^n-1} \underbrace{\sum_{x=0}^{2^n-1} (-1)^{f(x)} (-1)^{\oplus(x \circ y)}}_{a_y} |y\rangle \ ,
\end{aligned}$$

where $\oplus(x \circ y)$ is the XOR reduction of the bitwise AND of x and y.

If f is constant, then $a_y = (-1)^{f(x)} \sum_x (-1)^{\oplus(x \circ y)}$, so $a_y \neq 0$ if and only if $y = 0$: otherwise, the sum has an equal number of positive and negative terms. If, instead, f is balanced, then $a_0 = \sum_x (-1)^{f(x)} = 0$ (equal number of positive and negative terms). So, measuring $|f\rangle$ relative to the canonical basis gives as outcome:

- an all-zeroes bitstring with certainty when f is constant,
- a nonzero bitstring (i.e. at least one nonzero bit) with certainty when f is balanced.

This result was obtained by evaluating black-box operator U_f on a single input, which is an exponential speedup over the $2^{n-1} + 1$ evaluations that were needed classically.

The exponential speedup brought by the quantum solution to the previous problem has to be qualified, because in the classical case, it is possible to solve the Deutsch-Josza problem in polynomial time up to an exponentially small probability of error, so the exponential speedup is only true if the problem has to be solved with absolute certainty. However, we do know some problems for which quantum computing brings an exponential speedup with respect to the fastest classical algorithm, even when approximate solutions are enough. An example of such is Simon's algorithm, that determines the period of an unknown function, implemented as a quantum oracle [PK20, chapter 6].

A.6.2 Shor's algorithm

A practical application of quantum computing — provided a large-enough quantum computer is eventually built — is Shor's algorithm, that is able to factor an n-bit quasi-prime number N in polynomial time in n, while the best classical algorithm to date, the

general number field sieve, runs in subexponential time

$$L_N\Big(1/3, (64/9)^{1/3}\Big) \approx \exp\Big(1.34 n^{1/3} (\ln n)^{2/3}\Big) \ .$$

Quantum Fourier transform Shor's algorithm is based on the Quantum Fourier Transform (QFT), which is the quantum equivalent to the Fast Fourier Transform (FFT). It is a unitary transformation that, applied to n-qubit states, transforms the computational basis as such:

$$\mathrm{FFT}\,|x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2i\pi xy/N}\,|y\rangle \ ,$$

where $N = 2^n$. This transform is efficient, in that it can be implemented using a number of elementary quantum gates that is proportional to n. Applying the QFT to a superposition $|\psi\rangle = \sum_x a_x\,|x\rangle$ computes a state that, when measured, gives outcome $y \in [\![0, 2^n - 1]\!]$ with a probability proportional to the prevalence of frequency $y/2^n$ in (a_x) considered as a time series indexed by x.

Period finding The most straightforward way to determine the period of a black-box periodic function with inputs in $\{0,1\}^n$ is to check its output successively for all inputs until it repeats, for which $2^{n-1} + 1$ steps are needed in the worst case. Alternatively, it can also be determined by searching through its frequency spectrum, typically obtained with an FFT, to find the frequency with the largest amplitude. Yet, classically, this method is at least as slow as the previous one, because it results in a linear search among 2^n values. However, if one computes the QFT of the function and measures it, then the outcome of the measurement corresponds with high probability to the dominant frequencies of the function. Thus, by using the QFT, we can determine the period of the function with high probability in few quantum queries to the function.

Factoring of quasi-prime integers Shor and Long's algorithm applies this property of the QFT to the efficient factoring of semi-prime integers, that is, integers that are the product of two large primes. The algorithm is not restricted to these, but it is most interesting for these given that they are hard to factor classically.

Let n be the number to be factored. We can select at random $a \in [\![2, n-1]\!]$, such that $a \wedge n = 1$ where $\wedge$ represents the greatest common divisor. Note that if $a \wedge n \neq 1$, then we have found a nontrivial factor of n and we are done. Given that a and n are coprime, there exists a smallest r, called the order of a, such that $a^r \equiv 1 \pmod{n}$. Suppose that r is even (otherwise, start over until it is the case): then

$$n \mid (a^{r/2} - 1)(a^{r/2} + 1)\,,$$

and $n \nmid (a^{r/2} - 1)$, otherwise the order of a would be $\frac{r}{2}$ at most. We can also assume that $n \nmid (a^{r/2} + 1)$ (again, starting over if it is not the case). Thus $n \wedge (a^{r/2} - 1) \notin \{1, n\}$ and $n \wedge (a^{r/2} + 1) \notin \{1, n\}$ are nontrivial factors of n, that can be computed efficiently on a classical computer thanks to the Euclidean algorithm.

The only step in this algorithm that is hard classically is to find the order r of a. It appears that this step can be performed efficiently with a quantum computer, because r is the period of $f_a : x \mapsto a^x \pmod{n}$, that can be found using the QFT. Note that a quantum oracle for f_a is not difficult to implement, because we can compute the a^{2^i} classically, and the oracle only has to multiply them depending on the bits of $|x\rangle$. Finally, by running a polynomial-time classical-quantum algorithm, we are able to factor n with good probability — for details on the probability of success for a single iteration, we refer the reader to [PK20, chapter 6].

Consequences on cryptography The hardness of factoring semiprimes is a fundamental assumption for the security of the RSA public-key cryptography algorithm. Thus, the security of RSA breaks down entirely in the presence of a quantum computer sufficiently large to execute Shor's algorithm. Efficient quantum algorithms also exist for solving the discrete logarithm, which causes a similar breakdown in the security of elliptic-curve cryptography, the other public-key cryptography scheme currently in use. These algorithms must thus be abandoned if there is a threat of a large-enough quantum computer being built soon.

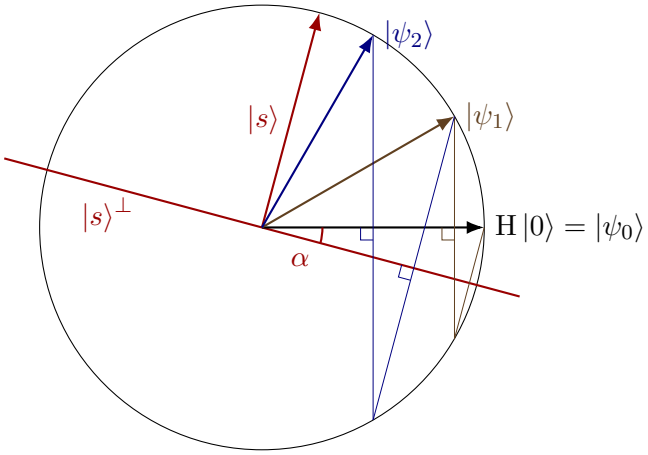
However, this threat is probably not immediate, because current quantum computers are very far from being large enough, considering the overhead of quantum error correction to bring the error rate of their computation down to acceptable levels.

A.6.3 Grover's algorithm

Another quantum algorithm of major importance, Grover's algorithm, allows a quantum computer to search within an unsorted database with a quadratic speedup with respect to the best corresponding classical algorithm: exhaustive search. Given an efficiently computable function f that has a single zero among $N = 2^n$ input values, Grover's algorithm allows to find this zero in $\frac{\pi}{4}\sqrt{N}$ quantum queries to f, while about $N/2$ queries would be needed classically [PK20].

Grover's algorithm can be explained geometrically (fig. A.4). Its principle is to first generate a uniform superposition of all possible inputs to f, then rotate that superposition towards a solution through a succession of symmetries determined by the application of f. This rotation, the Grover iteration, is performed with two reflections about hyperplanes: first, a reflection about the orthogonal to the solution is computed using f, then a reflection about the uniform superposition is performed. One Grover iteration is able to rotate the state vector away from the hyperplane orthogonal to the solution by an angle 2α, where $\alpha \approx 1/\sqrt{N}$ is the angle between that hyperplane and the uniform superposition. The solution can thus be attained after a global rotation of $\pi/4$, that is, after $\frac{\pi}{4}/\alpha \approx \frac{\pi}{4}\sqrt{N}$ Grover iterations [PK20].

The algorithm can handle functions having several zeros as well: if f has exactly k zeros (k being known), then we can find one of them in $\frac{\pi}{4}\sqrt{N/k}$ queries [PK20], and even if the number of zeros is entirely unknown, about $100\sqrt{N}$ queries suffice to find one by guessing the approximate number of solutions until the algorithm succeeds [Mag21].



Two iterations of Grover's algorithm searching for $|s\rangle$. Each iteration is composed of two orthogonal symmetries. $|\psi_i\rangle$ represents the state vector at the end of the i-th iteration, $|\psi_0\rangle$ being the initial state. Each iteration increases the angle between the state and $|s\rangle^\perp$ by 2α, where α is the initial angle.

Figure A.4: Geometrical interpretation of Grover's algorithm

Grover's algorithm, together with its generalization presented below, is extremely generic, and can be used to speed up almost any search or optimization algorithm, including minimum search, gradient descent, or Monte Carlo estimation methods [Mag21].

Generalizations — quantum amplification In some cases, f may exhibit some known structure, that can be used to further speed up quantum search. The basic algorithm rotates the state vector away from the orthogonal to the solution by the initial angle between the two, which is $\alpha/\sqrt{N}$. This situation is the worst case that can happen, when f is entirely unknown *a priori*. Now, if we know enough of f to build a circuit performing the reflection about a hyperplane having angle $\frac{\pi}{2} - \theta$ to the solution, with $\theta > \alpha$, we can use this reflection in place of the reflection about the uniform superposition in the Grover iteration. In this way, only $\frac{\pi}{4\theta} < \frac{\pi}{4}\sqrt{N}$ iterations are needed to complete the algorithm [PK20].

For some search problems, there exist heuristics that are able to speed up classical exhaustive search, by generating at random some inputs to f that are more likely to be solutions. Such a heuristic can also be used with Grover's algorithm, by integrating it into f, and searching not among the inputs to f, but among the seeds to the heuristic. The resulting speedup is still quadratic [PK20].

We can see this situation in another way: having a probabilistic or quantum algorithm $\mathcal{A}$ that succeeds with small probability ε, Grover's algorithm is able to boost this probability to (for instance) $\frac{2}{3}$ in about $1/\sqrt{\varepsilon}$ quantum executions of $\mathcal{A}$ and quantum verifications of the solution [Mag21]. In the classical case, about $1/\varepsilon$ steps would have been needed. This application, known as *quantum amplification* [Mag21], is obtained by using $\mathcal{A}$ as the heuristic and taking f to be the solution verifier.

Optimality It turns out that provided f is a black-box, Grover's algorithm is optimal: no quantum circuit can perform exhaustive search with high success probability in fewer

queries, and no quantum algorithm can obtain better success probability than Grover's when restricted to any fixed number of queries below $\frac{\pi}{4}\sqrt{n}$ [PK20].

Consequences on cryptography Given that it only brings quadratic speedup over classical computing, Grover's algorithm does not have as much impact on cryptography as Shor's algorithm, in that it cannot make a hard problem easy. It is, however, much more general than Shor's, and does affect the degree of intractability of hard problems. For instance, the only known method to break symmetric ciphers — barring attacks that make use of a flaw in the design or implementation of the cipher — is exhaustive key search, whose runtime is exponential in the length of the secret key. Due to the quadratic speedup, Grover's algorithm is able to effectively halve the security of the cipher, being able to break an n-bit key in as many iterations as breaking an $n/2$-bit key through classical means.

It can, likewise, lower the security of cryptographic hash functions. Finding a preimage or second preimage of an n-bit hash using Grover's algorithm only requires $2^{n/2}$ trials, in contrast to 2^n classically. For collision search, instead, it performs at most as well as classical search through the birthday method ($2^{n/2}$ operations), unless a quantum RAM[5] is built, in which case the runtime can be improved to $2^{n/3}$ operations [Mag21].

In either case, the impact of quantum computing can be compensated for by using symmetric ciphers with keys of twice the original length, without having to change the core algorithm. Hash functions, being generally chosen based on their collision resistance, are less affected by Grover's agorithm.

[5] A quantum RAM is a random-access memory storing classical information, that can be read quantumly at a superposition of addresses.