quantum reports

Tracked for Impact Factor

CITESCORE 3.3

Article

Practical Implementation of Unconditionally Secure File Transfer Application with QKD and OTP

Alin-Bogdan Popa, Bogdan-Calin Ciobanu and Pantelimon George Popescu

MDPI

https://doi.org/10.3390/quantum7010012

quantum reports

MDPI

Article

Practical Implementation of Unconditionally Secure File Transfer Application with QKD and OTP

Alin-Bogdan Popa, Bogdan-Calin Ciobanu and Pantelimon George Popescu *

Department of Computer Science and Engineering, Faculty of Automatic Control and Computers, National University of Science and Technology POLITEHNICA Bucharest, 060042 Bucharest, Romania; alin_bogdan.popa@upb.ro (A.-B.P.); bgdnkln@gmail.com (B.-C.C.)
* Correspondence: pgpopescu@upb.ro

Abstract: With the looming threat of quantum computers capable of breaking classical encryption and the uncertainty regarding the security of post-quantum encryption algorithms, some highly sensitive applications aim for the highest level of security in information transfer: unconditional security. In this work we present an architecture and a practical implementation of a user-friendly unconditionally secure file transfer client based on quantum key distribution and one time pad cipher. We test the implementation on the live QKD research infrastructure within POLITEHNICA Bucharest, thus proving the approach is feasible for real information transfer use-cases.

Keywords: quantum key distribution; unconditionally secure encryption; secure file transfer; one-time pad

1. Introduction

In the modern age of information, society relies heavily on various encryption algorithms (such as AES, RSA, ECDSA, etc.) to secure data in transit, especially in applications where security is critical, such as: governmental communication, defense secrets, patients' private medical data, and more. However, with the looming threat of quantum computers, classical encryption algorithms widely used at present (RSA, ED25519, etc.) are expected to become breakable by 2035 [1]. Yet many of the secrets transmitted have a life-time that is much longer than that; for example, government-issued information may be classified for 25–100 years, and patient data is expected to remain secret for the entire life of the patient. The "Harvest Now, Decrypt Later" strategy enables a malicious actor to intercept and gather currently undecryptable data until the decryption is possible (either via quantum computers powerful enough to run Shor's algorithm on meaningful input, but also perhaps via theoretical breakthroughs against classical algorithms). Thus, there is a strong incentive to replace encryption algorithms for sensitive data transmissions with a safer alternative as soon as possible.

While post-quantum cryptography is under development and awaits to be subjected to more comprehensive testing, quantum key distribution (QKD) [2] is the only known protocol for unconditionally secure (that is, it remains secure even against attackers with infinite computation power) key exchange [3]. The keys can then be used for unconditionally secure encryption and decryption between distant locations using an information-theoretically secure encryption algorithm such as One-Time Pad (OTP) [4].

QKD is perhaps the first quantum technology to achieve a fully mature state, with multiple commercial QKD hardware available and multiple small- and large-scale QKD networks already deployed. Vendors providing QKD devices include IDQuantique (the

check for updates

Received: 21 January 2025
Revised: 6 March 2025
Accepted: 12 March 2025
Published: 14 March 2025

Citation: Popa, A.-B.; Ciobanu, B.-C.; Popescu, P.G. Practical Implementation of Unconditionally Secure File Transfer Application with QKD and OTP. *Quantum Rep.* **2025**, *7*, 12. https://doi.org/10.3390/quantum7010012

first to provide commercially available QKD devices, in 2003), Toshiba, ThinkQuantum, LuxQuanta, QTI, and several others. The first QKD network to be deployed in practice was the DARPA Quantum Network [5], which operated between 2003–2007 and consisted of 10 nodes across Boston and Cambridge, MA (USA). The first European network was the SECOQC project [6], operating between 2004–2008 and consisting of 6 nodes in Vienna, Austria. QKD keys have been used intercontinentally for the first time as part of the QUESS space mission in China [7], wherein the Micius satellite was used to establish an intercontinental quantum-secure video call between Vienna, Austria and Beijing, China (over a ground distance of 7500 km). In the European Union, in 2019, the largest international QKD initiative to date started as the European Quantum Communication Infrastructure (EuroQCI) [8] declaration was signed. Subsequently joined by all 27 EU Member States, EuroQCI involves building and interconnecting national QKD networks in all EU Member States in order to secure varied national and international use-cases, such as the communication of EU agencies, national bodies, public administrations, medical services, data center activities, and more.

Although interaction with QKD devices is thoroughly understood in the community (with widely-supported standards such as ETSI-014 [9] already in place), and protocols for secure file transfer have been developed [10–12], there is still a lack of production-ready client software that can enable non-technical users (diplomats, members of the public administration, doctors, etc.) to leverage the security of a QKD network within their usual communication. Our contribution in this paper is to lay the architecture for such a system, as well as provide a practical implementation of an unconditionally-secure file transfer client that has been tested over a real QKD link.

2. Prerequisites

The BB84 [2] QKD protocol works as follows. Alice generates two random series of bits, a_i and b_i, which encode the polarization of a photon. For example: if both bits are 0, then the photon is polarized horizontally (at an angle of 0 deg); if $b_i = 1$ and $a_i = 0$, then the photon is polarized diagonally at an angle of 45 deg. If $b_i = 0$ and $a_i = 1$, then the photon is polarized vertically (at an angle of 90 deg). If both bits are 1, then the photon is polarized diagonally at an angle of 135 deg. Alice generates photons sequentially following the series of bits (a quantum random number generator can be used for the bits in order to ensure true randomness), and sends each photon to Bob. For each photon, Bob generates a bit b'_i which encodes the basis in which he will measure the photon: if $b'_i = 0$, then he will measure in rectilinear basis (obtaining either the horizontal or the vertical polarization), otherwise he will measure in diagonal basis (obtaining either the diagonal at 45 deg polarization or the diagonal at 135 deg polarization). If Bob's b'_i matches Alice's b_i, then Bob's measurement a'_i will exactly match Alice's a_i; otherwise, Bob's measurement will be random. After measuring all photons, both Alice and Bob publicly disclose the bases used (b_i and b'_i); then, they keep the bits a_i and a'_i where the bases matched ($b_i = b'_i$). Following this, they perform an additional error correction step to guarantee they both have the exact sequence of bits, and an additional privacy amplification step [13] to guarantee that no eavesdropper holds any partial information about the key. The final series of bits after error correction and privacy amplification forms the key which can then be used to encrypt further communication. The security of the protocol lies in the fact that any measurement performed by an eavesdropper to one of the photons necessarily collapses its polarization to one of the eigenstates of the selected measurement basis, thus introducing errors that Alice and Bob will be able to detect during the error correction process; secondly, copying the state of an unkown photon and measuring it after the public basis disclosure is not possible by virtue of the no-cloning theorem. Other

QKD protocols may use different physical primitives (for example: E91 [14] relies on entanglement and violating the Clauser–Horne–Shimony–Holt (CHSH) inequality), but the security implications are the same.

We assume two users, Alice and Bob, are physically present at different end points of a QKD network, such that secure keys may be generated between them, with an ID mechanism which allows them to uniquely identify keys. In practice, QKD devices typically provide a key interface API following the ETSI-014 standard [9]. In this standard, each node in the QKD network provides a Key Management Entity (KME) which exposes the REST endpoint **Get key**, which allows a Secure Application Entity (SAE) within the node to get a number of secure key material, together with each key's unique ID, with a different SAE running on a different node. The requesting user would then send the key IDs to the other SAE via any classical communication method (intercepting these IDs does not compromise the security of the protocol). A different KME endpoint **Get key with key IDs**, when called on the other node with the key IDs as received by the other user, provides the same keys that were generated in the first step. The two users now share the secret keys and can use them further for encryption; at the same time, the keys are discarded and rendered inaccessible from the QKD devices. The caller's SSL certificate data is used in order to uniquely identify the calling SAE whenever one of the endpoints is called. The keys themselves have a set size (which, depending on the QKD device used, may be configurable), and are base64-encoded. The full data flow between KMEs and SAEs is displayed in Figure 1. It should be noted here that the endpoints are only accessible from the physical (presumably secure) premises of the QKD node; requesting the keys from a remote location would involve the keys being encrypted with a separate protocol for them to be transferred from the QKD node to the actual location of the user, thus rendering the entire key distribution only as secure as this transfer protocol.

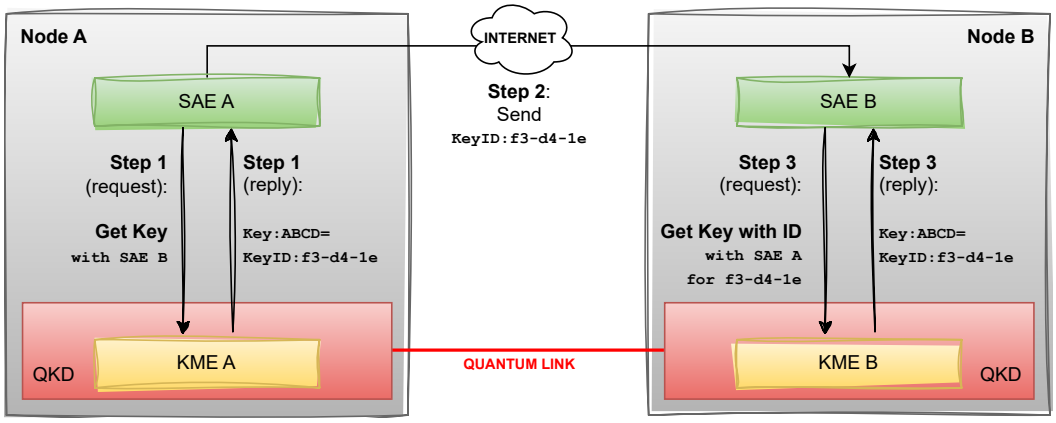

Figure 1. Schematic of request and reply flow between KMEs and SAEs as defined in the ETSI-014 standard.

To ensure the perfect secrecy property of OTP encryption, the following conditions must be met:

1. The key length must be at least equal to the length of the plaintext.
2. The key must be truly random.
3. No part of the key may be reused.
4. The key must be kept completely secret and must not be stored after it has been used.

Condition 1 implies that the number of key bits must exactly match (or be greater than) the number of bits in the plaintext message to be sent. Condition 2 is guaranteed by virtue of using QKD-generated keys. To ensure conditions 3 and 4, the key must be

generated on the spot for each piece of the file to be sent, must be used only once, and then must be deleted from the memory of the device.

Additionally, to ensure the authenticity and integrity of the message, a message authentication code (MAC) should be appended as part of the message transmission. One may think a potential attacker does not have physical access to the QKD node, and hence cannot generate valid keys that would produce a valid message; this is correct. Moreover, since keys are discarded from the QKD device after the request, replay attacks are impossible because sending the same key IDs that have been sent before would result in an error for the user requesting them via the **Get key with key IDs** endpoint. Thus, not including a MAC does not compromise the perfect secrecy of the file transmission (that is, the ciphertext conveys no information about the plaintext); however, it does prevent a denial of service attack wherein the attacker submits randomly crafted messages trying to impersonate a counterparty (missing message authenticity), as well as message tampering which would result in the file getting transmitted incorrectly (missing message integrity). To achieve both integrity and authenticity, we must add an information-theoretic secure MAC, such as Wegman-Carter MAC [15]. It computes a universal hash function of the message which it then encrypts with a OTP key, using an additional QKD key per message.

A discussion is perhaps needed on the choice of OTP for encryption, as opposed to other symmetric encryption algorithms such as AES. It can be noted that as long as the key is generated with a trusted QKD system, the resulting system is quantum resistant. However, the goal is to achieve information theoretic security (unconditional security) for the file transfer—that is, security which does not rely on assumptions of limited computation power of the adversary. While this may seem to be an overkill for day to day communication, there are plenty of historical examples where OTP was used in diplomatic or military communication [16]. One obvious disadvantage of OTP is the fact it requires keys as long as the message to be transmitted; thus, the rate of transmission is necessarily, at best, as high as the rate of secret key generation. With today's commercial QKD devices providing key rates of 1–2 kbps on average, this is severely limiting. However, software algorithms have been proposed to alleviate the issue by optimal key forwarding and redistribution [17].

3. Architecture

The architecture for the unconditionally secure file transfer system, shown in Figure 2, is composed of three separate zones: Alice's secure location (in purple), Bob's secure location (in blue), and the central public broker (in green).

Client Alice wishes to securely send a file to client Bob, with whom she is connected via a quantum link within a QKD network. To this end, Alice requests $N + 1$ keys of size S from her QKD device using the **Get Key** endpoint, where N is the number of blocks sent per message. One key (displayed in orange) will be reserved for the MAC algorithm, while the rest of the keys are concatenated into a message key of size NS. She splits the file into blocks of size S each (padding the last block with zeros, if necessary) and generates the current message plaintext by concatenating the next N blocks of the file. To produce the ciphertext, she bitwise XORs the message plaintext with the concatenated key. The plaintext together with the MAC key are passed into the MAC algorithm, which produces the message MAC. The final message consists of the ciphertext, the message MAC, the ID of the MAC key, and the list of IDs of the keys used to build the message key (preserving their order).

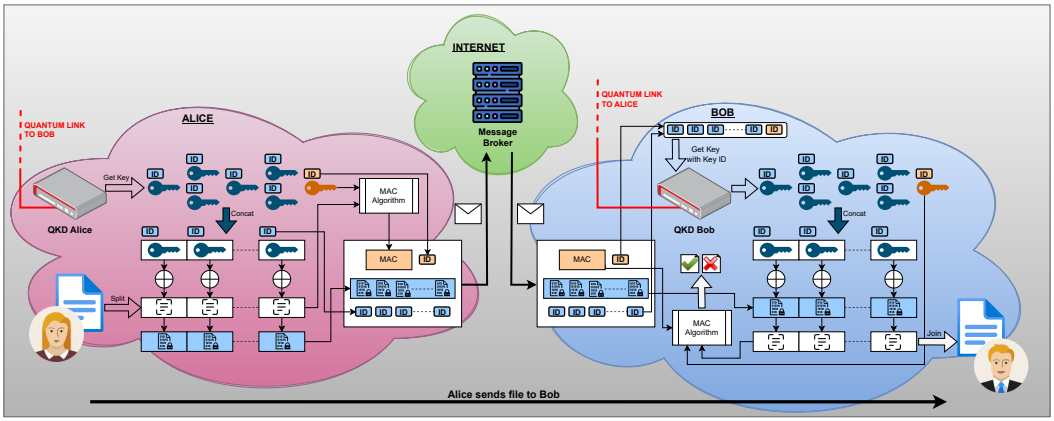

Figure 2. The system architecture for unconditionally secure file transfer.

Although the message could be sent directly to Bob, we decided to use a centralized message broker in order to prevent network address translation (NAT) issues. The role of the broker is to maintain an open connection with Alice and Bob and relay any message from one to another without any check on the authenticity of the messages. With this approach, even if neither Alice nor Bob have a public IP address and they can't connect to each other directly, they will still be able to send and receive files from each other. It should be noted here that since the ciphertext is encrypted with perfect secrecy, the messages can be treated as public with no impact on the security of the system; thus, introducing a broker does not compromise the security of the protocol.

In order to integrate the broker, an additional field is required in the message to specify to the broker that the message is to be forwarded to Bob.

On the receiving end, Bob builds an array of all key IDs (the message key IDs and the MAC key ID), and triggers a call to the **Get key with key IDs** endpoint of his QKD devices with the array of IDs as parameter. If the IDs are correct, Bob will receive a list of keys from the QKD device that exactly match the keys used by Alice. Bob isolates the MAC key for later usage; the rest of the keys are concatenated in the same order provided by Alice to build the message key. Bob then bitwise XORs the message key with the ciphertext received from Alice to obtain a plaintext candidate. By applying the MAC algorithm using the computed plaintext and the MAC key, Bob is able to compare the result with the MAC received from Alice; if the two MACs match, the message is accepted, otherwise it is rejected. If accepted, the plaintext blocks are appended to the received file.

Additional file metadata may be needed to be transferred between Alice and Bob. For example, Alice may want to include the name or type of the file. Additionally, for Bob to know when the communication has ended and the file has been successfully saved, Alice should transmit to Bob either the total size of the file (if known in advance), or a special control string to mark the end of the transmission. If any such metadata is required, we consider it part of the plaintext that Alice generates, and thus it does not require additional fields or headers in the message.

If Bob's connection is not point-to-point (i.e., there are potential senders other than Alice), then, in order for Bob to know with whom he must call the **Get key with key IDs** endpoints, some identification element of Alice should be appended to the message. This could be a unique ID or name for Alice's node, or perhaps a classical authentication certificate.

4. Practical Implementation

For the practical implementation, we have built a user-friendly GUI client following the architecture outlined above. The implementation is written in Python 3.10 with PyQT5 for the graphical user interface, and is publicly available on Github [18,19].

The message broker is a CLI server which should be run on a publicly accessible, static IP machine. The broker supports incoming connections from clients using ZeroMQ sockets bound to a listening port. We have decided for using ZeroMQ due to its high performance, support for the multi-part request-reply messaging pattern, and its ability to handle asynchronous messaging and robust socket abstractions. Each message received by the broker is expected to be a multipart message consisting of the client's ID and a command, with optional data following. The commands supported by the broker are the following:

- Register—allows a new client to record its unique client ID (which must be different from all other registered clients).
- List Clients—allows a client to query the list of registered clients and see the client IDs of each.
- Relay—instructs the broker that the message is to be relayed to another client, specified by their client ID as part of the command.

The client GUI starts with a connection screen (displayed in Figure 3), prompting the user to fill in the broker's IP address and port, as well as the client's unique name. Upon connecting, the client makes a Register call to the broker, and the GUI switches to the main interface (displayed in Figure 4), which resembles the interface of a FTP client. A sidebar on the left allows the user to browse the system files and see their name, size, and file type. In the main section, a Refresh Client List button triggers a call to the broker with the List Clients command, and refreshes the list of clients in the dropdown below. With a file selected in the file explorer sidebar and a client selected in the dropdown list, the user can click the Send button to initiate the file transfer. The progress towards sending or receiving a file is displayed with a green progress bar. A status bar at the bottom of the main interface displays the current action (sending or receiving), the counterparty, and the time elapsed since the start of the action. If no action is currently underway, the message "Status: Ready" is displayed instead.

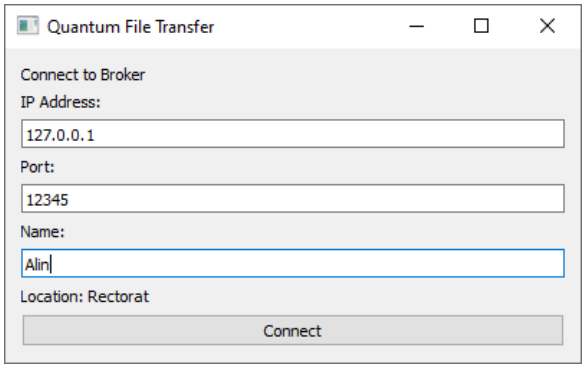

Figure 3. Connection screen of the file transfer client.

In order to ensure the GUI is not blocked while a send or receive is in progress, two separate threads (implemented using QThread from PyQT5) handle file sending and receiving.

The sending worker thread keeps a list of files currently being sent, with each file's path, QKD nodes location, counterparty client name, and file sending progress. While there are files remaining in the list, the worker requests the keys from the QKD device using the **Get key** endpoint, performs the XOR operation, computes the MAC, sends the message to the broker, and locks sending for this file until an ACK message is received from the receiving counterparty (via the receiving worker thread).

The receiving worker thread listens for messages from the broker and processes each message as it arrives. If the message is an ACK for a file currently being sent, it notifies the sending worker accordingly. If the message is a new part of a file, then it requests the respective keys from the QKD using the **Get key with key IDs** endpoint, performs the XOR operation to generate the plaintext, computes and validates the MAC, stores the result, and sends an ACK to the sending counterparty.

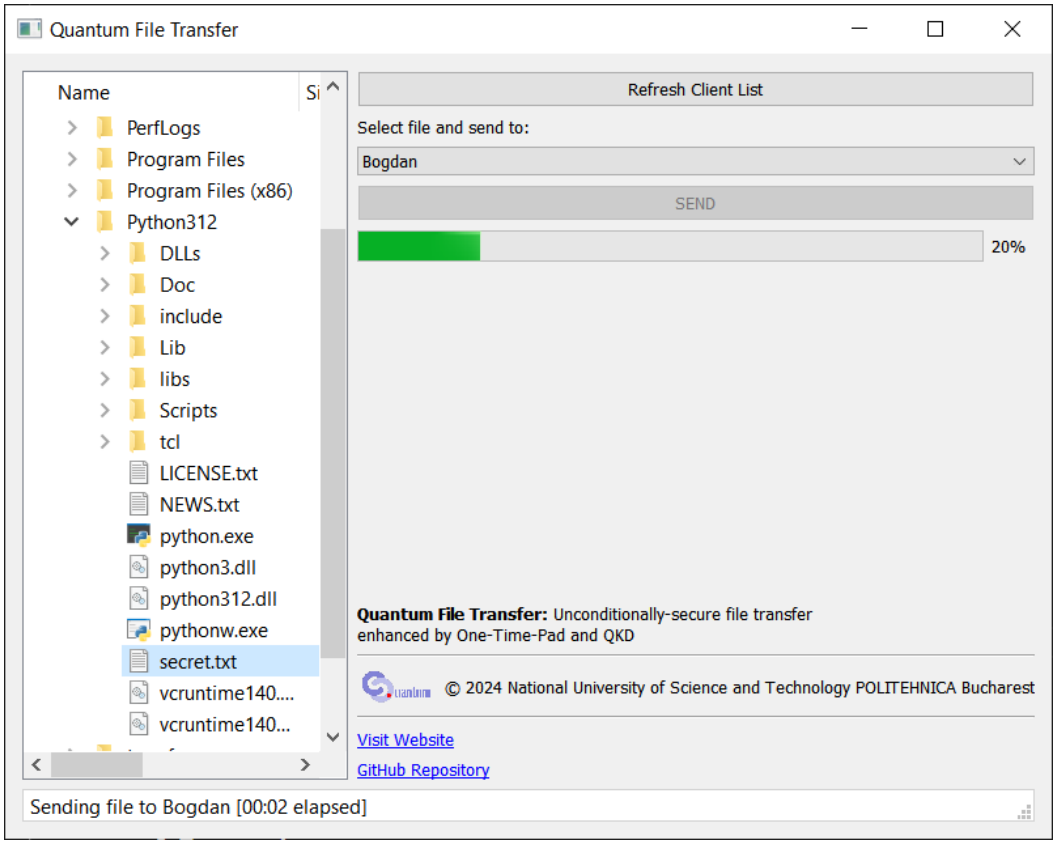

Figure 4. Main interface of the file transfer client.

When needed, the worker threads notify the main thread via PyQT signals (for example, to notify the sender thread that an ACK has been received, or to update the file sending or receiving progress). To track progress and to have the receiver know when the communication has ended, whenever a file transfer is initiated, the first part of the plaintext consists of the metadata of the file which includes the file name and size. This approach allows, both on the client and the broker side, for multiple concurrent transfers, even between the same clients.

To separate the unconditionally secure file transfer logic from the interface provided by the specific QKD devices used (in our case, ETSI-014), the calls to the QKD endpoints have been implemented in a separate module QKDGKT (QKD Get Key Tool). The QKDGKT module has a JSON configuration in which the user specifies the KME hostnames and the SAE names for each accessible node in the QKD network, the location of the current user, as well as the required authentication certificates for interacting with the QKD device. Following the ETSI-014 standard, calls for **Get key** are performed as GET requests to https://[KME_hostname]/api/v1/keys/[SAE_ID]/enc_keys, which returns a key container message that consists of an array of base64-encoded keys and their IDs. Calls for **Get key with key IDs** are performed as POST requests to https://[KME_hostname]/api/v1/keys/[SAE_ID]/dec_keys, with a JSON body consisting of an array of key IDs to be requested. If the key IDs are correct, the response is identical with the response obtained for the call to

Get key. Both requests include a SSL certificate of the requesting user, which is configured in the QKD device as part of its consumer path.

The file transfer architecture and client are designed to work with any QKD infrastructure that supports key requests via the ETSI GS QKD 014 interface, which is implemented by all major QKD vendors. The file transfer client was tested in practice on the QKD infrastructure of the National University of Science and Technology POLITEHNICA București, achieving the first Romanian unconditionally secure file transfer using QKD and OTP in July 2023. The broker was deployed on a publicly accessible virtual machine within the university's computing cluster, while the two file transfer clients were opened between two buildings ("Precis" and "Rectorat") on the university campus. These locations are connected via an ID Quantique Cerberis XGR QKD link, as illustrated in Figure 5.

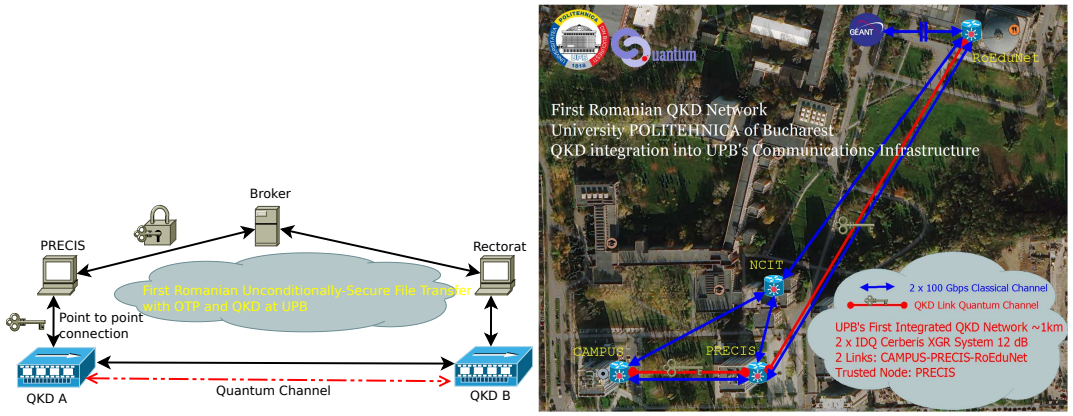

Figure 5. Architecture for the first Romanian unconditionally-secure file transfer with OTP and QKD (**left**) performed over the QKD infrastructure at POLITEHNICA Bucharest (**right**).

The IDQ Cerberis XGR system operates using the Coherent One-Way (COW) QKD protocol, with a dedicated fiber for the quantum channel and a separate line wherein three signals (service, management, and KMS channels) are multiplexed over a Dense Wavelength Division Multiplexing (DWDM) fiber in the C-band. The vendor datasheet specifies a key generation rate of up to 2 kbps at a security parameter of 4×10^{-9}, with a maximum fiber distance of 60 km at 12 dB loss. In our experimental setup, we successfully transferred a file between the two locations using OTP encryption with QKD-derived keys. The observed file transfer rate was approximately 0.5 kbps, which we attribute to factors such as QKD key buffer depletion, network protocol overhead, and message synchronization delays.

5. Conclusions

In this paper we have provided the architecture for an unconditionally-secure file transfer application, and we have explained the approach we have taken to implement it in practice when conducting the first unconditionally-secure file transfer with QKD and OTP in Romania. We conclude that leveraging QKD-generated keys for secure file transfer is feasible practically, even for non-technical users.

Author Contributions: Conceptualization, methodology, validation, writing—original draft preparation, writing—review and editing: A.-B.P., B.-C.C. and P.G.P.; supervision: P.G.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The source code is publicly available on Github [18,19].

Acknowledgments: This work has been partially supported by RoNaQCI, part of EuroQCI, DIGITAL-2021-QCI-01-DEPLOY-NATIONAL, 101091562.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. McKinsey & Company. Steady Progress in Approaching the Quantum Advantage. Available online: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/steady-progress-in-approaching-the-quantum-advantage (accessed on 10 June 2024).
2. Bennett, C.H.; Brassard, G. Quantum cryptography: Public key distribution and coin tossing. *Theor. Comput. Sci.* **2014**, *560*, 7–11. [CrossRef]
3. Scarani, V.; Bechmann-Pasquinucci, H.; Cerf, N.J.; Dušek, M.; Lütkenhaus, N.; Peev, M. The security of practical quantum key distribution. *Rev. Mod. Phys.* **2009**, *81*, 1301–1350. [CrossRef]
4. Shannon, C.E. Communication theory of secrecy systems. *Bell Syst. Tech. J.* **1949**, *28*, 656–715. [CrossRef]
5. Elliott, C.; Colvin, A.; Pearson, D.; Pikalo, O.; Schlafer, J.; Yeh, H. Current status of the DARPA quantum network. In Proceedings of the Quantum Information and Computation III, Orlando, FL, USA, 28 March–1 April 2005; SPIE: Bellingham, WA, USA, 2005; Volume 5815, pp. 138–149.
6. Peev, M.; Pacher, C.; Alléaume, R.; Barreiro, C.; Bouda, J.; Boxleitner, W.; Debuisschert, T.; Diamanti, E.; Dianati, M.; Dynes, J.; et al. The SECOQC quantum key distribution network in Vienna. *New J. Phys.* **2009**, *11*, 075001. [CrossRef]
7. Liao, S.K.; Cai, W.Q.; Liu, W.Y.; Zhang, L.; Li, Y.; Ren, J.G.; Yin, J.; Shen, Q.; Cao, Y.; Li, Z.P.; et al. Satellite-to-ground quantum key distribution. *Nature* **2017**, *549*, 43–47. [CrossRef] [PubMed]
8. European Commission. The European Quantum Communication Infrastructure (EuroQCI) Initiative. Available online: https://digital-strategy.ec.europa.eu/en/policies/european-quantum-communication-infrastructure-euroqci (accessed on 10 June 2024).
9. ETSI. ETSI GS QKD 014: Quantum Key Distribution (QKD); Protocol and Data Format of REST-Based Key Delivery API. Available online: https://www.etsi.org/deliver/etsi_gs/QKD/001_099/014/01.01.01_60/gs_qkd014v010101p.pdf (accessed on 22 December 2024).
10. Armanuzzaman, M.; Alam, K.M.R.; Hassan, M.M.; Morimoto, Y. A secure and efficient data transmission technique using quantum key distribution. In Proceedings of the 2017 4th International Conference on Networking, Systems and Security (NSysS), Dhaka, Bangladesh, 18–20 December 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–5.
11. Fujiwara, M.; Kato, G.; Sasaki, M. Information theoretically secure data relay using QKD network. *IEEE Access* **2024**, *12*, 141167–141178. [CrossRef]
12. Yalla, S.P.; Uriti, A.; Sethy, A.; Sathishkumar, V. Secure method of communication using Quantum Key Distribution. *Appl. Comput. Eng.* **2024**, *30*, 32–37. [CrossRef]
13. Bennett, C.H.; Brassard, G.; Robert, J.M. Privacy amplification by public discussion. *SIAM J. Comput.* **1988**, *17*, 210–229. [CrossRef]
14. Ekert, A.K. Quantum cryptography based on Bell's theorem. *Phys. Rev. Lett.* **1991**, *67*, 661. [CrossRef] [PubMed]
15. Carter, J.; Wegman, M. Universal classes of hash functions (extended abstract). In Proceedings of the STOC'77: Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, Boulder, CO, USA, 4 May 1977.
16. Kahn, D. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*; Simon and Schuster: New York, NY, USA, 1996.
17. Popa, A.B.; Popescu, P.G. Optimal key forwarding strategy in QKD behaviours. *Sci. Rep.* **2024**, *14*, 13977. [CrossRef] [PubMed]
18. Popa, A.B.; Ciobanu, B.C.; Popescu, P.G. QKD Get Key Tool. Available online: https://github.com/QuantumUPB/QKD-Infra-GetKey (accessed on 10 July 2024).
19. Popa, A.B.; Ciobanu, B.C.; Popescu, P.G. Quantum File Transfer. Available online: https://github.com/QuantumUPB/QKD-App-FileTransfer (accessed on 10 July 2024).