Quantum1Net

A protocol for key generation and message encryption

Introduction

A critical step in the transmission of encoded messages is the exchange of information to allow decryption upon receiving the message. This ancillary information is often embodied in keys, which act as additional input to the encryption and decryption algorithms. Encryption techniques can be classified in two broad categories based on the nature of the keys: symmetric and asymmetric methods [1][2].

Symmetric algorithms are those for which the encryption and decryption keys are equal. These systems are inherently weak against eavesdroppers: Alice must send the key over a communication channel to Bob. If a third party, Charlie, was sniffing the network he could find the key and be able to decipher the message. Asymmetric encryption techniques solve this problem by utilizing algorithms which require different ancillary keys for the encryption and decryption processes. For example, the RSA protocol specifies that when a user (Alice) wants to send a message to another (Bob), she first requests his encryption key. The key is sent over the network (and hence is denoted as the public key) and used by Bob to generate cypher text. To recover the message, Alice uses the decryption (private) key, which is never made public. RSA, elliptic-curve cryptography and others rely on this. As the private key is never revealed, in principal the message is safe. However, it is possible (but extremely difficult) to calculate the private key from the public one in the case of many asymmetric methods.

Whether symmetric, asymmetric or more complicated set-ups are implemented, the transmission of any kind of key is an intrinsically weak component of any cryptographic system. In the case of asymmetric systems, these are designed so that obtaining the private key from the public key is an effectively impossible task with conventional methods, such as factoring large integers into primes. However, the future rise of quantum computers will bring the obsolescence of many asymmetric cyphers, as these devices are well suited for completing these tasks. This forces cryptology to find new mechanisms to obscure information. Several algorithms which are thought to be quantum-secure have been proposed. Another approach is to utilize quantum mechanics itself to protect key distribution, for example by means of the quantum no-cloning theorem. However, this would require the implementation of a fully-quantum network, which to this day is not yet achievable [3] [4].

In this paper we describe the implementation of a key generation and exchange protocol for use with Quantum1Net's quantum encryption key generator. The design avoids the explicit exchange of encryption keys over the network, which are instead generated locally on Alice and Bob's machines. The process we utilize to generate keys encryption keys locally is computationally simple, but requires large number of random numbers, where our QRNG plays an important role by providing continuous source of fundamentally random numbers.

Quantum1Net's QRNG and network

Quantum1Net's quantum random number generator (QRNG) based on a two-arm quantum optical device. The numbers a generated from pairs of detection coincidences between the arms. It uses a Type-I or Type-II non-linear crystal where fraction of laser photons undergoes a process known as spontaneous parametric down conversion (SPDC) as a source of polarization-entangled pairs of photons [5]. The time coincidence windows and detection probabilities at the detectors can be controlled by modifying the setup parameters of the different components of the QRNG. This allows for the implementation of additional layers of security, as the entropy (or statistical properties) of the random numbers can be tuned upon agreement of the parties which obtain random numbers from the device. For a more extensive description of this device, the reader is referred to [6] and [7]. For this paper, it is sufficient to state that the QRNG generates streams of truly random numbers, with no underlying mathematical algorithm or a predictable pattern.

A fully quantum mechanical network would allow for protection from eavesdroppers. This kind of network would use photons or other suitable quanta as a means of transporting information, which would be encoded in their states. An attempt to intercept the particle en-route would constitute a measurement, which automatically modifies the state, by means of the wave function collapse postulate [8]. In addition, the quantum no-cloning theorem makes it impossible to accurately recreate an unknown quantum state, as when this is observed the collapse destroys superposition. A quantum network would allow for the safe transmission of encryption key. Information exchanging parties would expect a given quantum state, and hence measurement statistics. The presence of an eavesdropper intercepting the message would ultimately modify this, which would allow for the communication to be aborted before crucial information is endangered.

However, at the current technological level, long-range quantum mechanical networks, necessary for global-scale quantum communications, are still very difficult to implement on practice. While numerous long-distance experiments using fiber-optical channels have been performed on distances over 100km, this is not sufficient to implement the worldwide quantum network [9]. Additionally, due to photon decay in a long fiber, the data transfer rates are very low [10]. Developing a quantum repeater that would be able to 'amplify' the signal is even more difficult to implement [11].

Quantum1Net's first proposal consists of an overlay-network for currently available infrastructures, such as the Internet. In a nutshell, the most basic setup consists of a three-way connection between two communicating clients A, B and a Q1N node server R. Note that, for example, a bank can be one of the clients and co-located with the Q1N node, or A, B and R can be at 3 different locations. Keys are never explicitly exchanged between the clients. Instead they are generated locally at A and B by means of random timestamps obtained from the QRNG. A node can optically include its own QRNG or external QRNG located on the Q1N network can be used. The key

generation depends on a series of rules and agreements between A and B. Intercepting a subset of timestamps between R and A or B would in principle be meaningless for an eavesdropper. However, in case all network traffic between A, B, and N is intercepted, additional software is used to hide actual key.

Key generation and synchronization procedure

We now describe the generalized protocol of key generation and synchronization for use on the Quantum1Net network. It is important to note that while the key generation method described below requires very low computational overhead the time needed to generate the cipher text primarily depends on the speed of the peer-to-peer connection between A and B.

We assume that two users, which are denoted as Alice (A) and Bob (B), as is traditional, are to communicate. Client A wishes to send a secure message to B, and both have access to the Quantum1Net QRNG and one or more Node Servers (denoted here Q and R respectively). The proposed protocol is as follows:

1. Three-way connections are established between A, B, a QRNG and a Q1N node R with the following topology (see figure 1): A and B are connected to each other via a peer-to-peer connection (we use a WebRTC data channel) and both clients are also connected to QRNG. Node R is initially used to establish the direct connection between A and B. If, in some cases, a direct connection cannot be established due to local network topologies at A and B's locations, the R node may also act as a TURN (relayer) server to forward data between A and B.
2. QRNG continuously generates four independent streams of timestamps, A1, A2, B1 and B2, corresponding to the four detectors, which A and B can request from R after a peer-to-peer connection is established.
3. A (B) receives timestamps from A1 (B1) and A2 (B2). While A1, A2 timestamps are correlated with B1, B2 timestamps, A1/A2 (or B1/B2) pairs are not.
4. A chooses the stream A1 or A2 and forwards it via a direct network connection to B.
5. B compares the received timestamps with B1 and B2, within a specified coincidence window, and sends B1 or B2 to A.
6. Similarly, A compares the timestamps received from B with A1 and A2. For each A1, B1 or A2, B2 coincidence a 1 is assigned, and 0 otherwise (or vice-versa, as long as long as the two parties agree on the choice).
7. This way, A and B obtain the same set of random bits (a vector). This vector is used to train a three-parity machine – a special type of multi-layer feed forward neural network (NN). Our simulated tests have shown that the time needed for training process ultimately depends on the rate at which random numbers are generated from timestamps and the speed of the network connection between A and B. In comparison, the computational time is negligible. To reduce the number of network requests we utilized multiple

relatively small neural networks which are trained and compared in parallel instead of using a single large neural net that may require too many training steps and take a significantly longer amount of time on low bandwidth networks.

8. Each neural network is initialized with a set of random weights, generated locally with a PRNG or by randomly selecting a subset of random numbers generated in previous steps. The neural networks must be initialized using a random set only known locally.
9. The following training steps are repeated until all A and B weights for each NN are synchronized (these steps are performed for each neural network in parallel):
    (a) Generate a random input vector **X**, as described in 3-7. A and B must use the same random vector.
    (b) Compute the values of the hidden neurons and the output neuron (Oa for A, Ob for B). Oa, Ob = {-1,1}.
    (c) A and B exchange Oa and Ob values. One of the two compares the results and informs the other party of the result.
    (d) If Oa and Ob do not match, return to a).
    (e) If they do match, a suitable learning rule is applied for the weights. A quick weight check sum (e.g crc23) is computed and compared between A and B.
    (f) If quick checksums match, we suspect that A and B are synchronized. Because quick checksum might not be unique, a more complex checksum (e.g sha-256) is computed and exchanged to verify that weights match.
    (g) If both checksums match, the neural networks at A and have the same weights, and the training is complete.

10. The weights are combined to form a random bit vector, which is the encryption key. This is used in conjunction with a chosen, non-periodic, chaotic cellular automata rule (e.g rule 30, in practice we researched more complex 2 or higher-dimensional rules that similarly display chaotic behavior).

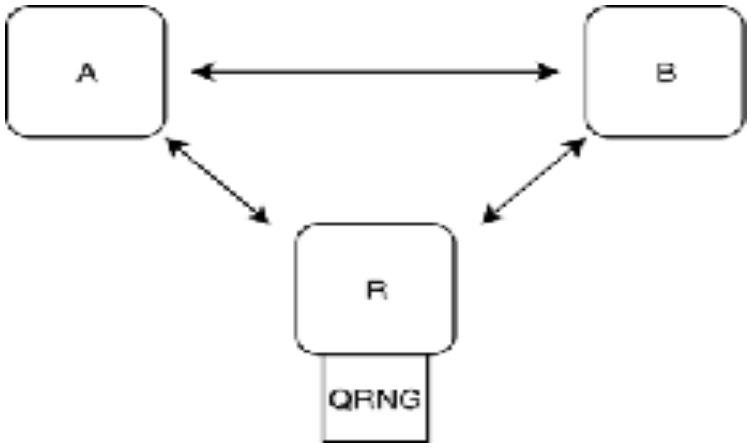

Figure 1: Schematic of the topology used for key generation

The above protocol is used to generate 2 (or more) independent vectors for each client. Then the following steps are taken to encrypt and transmit the message from A to B. Note that the secondary encryption on the node R is only used if the node is required to relay data between A and B. If a direct peer-to-peer connection can be established without a relayer (e.g. TURN server), we can skip the secondary encryption step described below.

1. When a client is registered for the first time, a random vector is generated between this client and R. This key is stored in the database on R and locally on the client. It is associated with unique device ID assigned to each client during software installation. This key may be periodically re-generated as needed, for example when the client software is reinstalled. We call this vector |A1> (for client A) and |B1> for client B.
2. For every new session or transaction, a new random vector is generated, which expires as soon as said session or transaction ends. It is temporarily stored on A and B until transaction is completed. We call this vector |AB>.
3. Multilayer cellular automata (CA) evolution is used to encrypt the data. Each CA layer uses a separate vector as initial conditions. For example, in the case of two vectors, A uses |A1> for first layer and |AB> for second layer.
4. Because A or B only share |AB> and A does not know |B1>, nor B knows |A1>, the message is relayed though the R server, which partially decrypts message using |A1> and then re-encrypts it using |B1>. Usually message is sent in chunks, so each chunk undergoes this re-encryption process on R independently.
5. When B receives the message relayed by R, it decrypts it using the |B1> and |AB> vectors.

A schematic of the transformations the data undergoes assuming a 2-layered cellular automata is shown in figure 2, to illustrate the process.

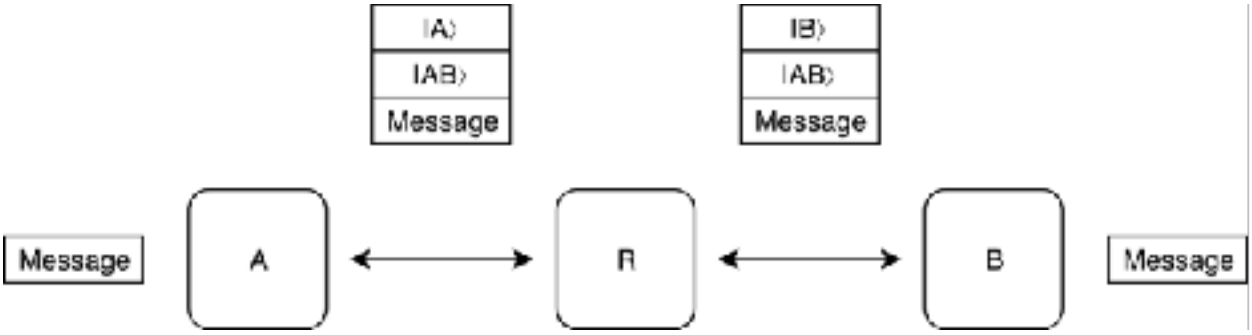

Figure 2: Schematic of the encryption steps a message undergoes during transport, as specified by the protocol described in this work. The message is partially decrypted and re-encrypted in the relayer server R.

Conclusions

With the current unavailability of fully quantum networks, protected from eavesdroppers by the no-cloning theorem, alternative setups must be designed to ensure protection from message and key interception. The key exchange step in any cryptographic system is an intrinsically weak point which can be used to intercept sufficient information to decrypt the protected message.

In this paper we detail an encryption protocol based on vectors intrinsically random numbers obtained using Quantum1Net's QRNG. Mutually authenticating encryption keys are generated from streams of random numbers generated from photon detection coincidences. The actual encryption keys are never explicitly transmitted over the network, as keys are generated locally at A and B following the steps detailed above. As keys are never transported from A to B, the only object left for a potential eavesdropper to intercept is the cypher text corresponding to the message, which without additional information cannot be decrypted.

This protocol, in combination with the agility and low overhead with which new keys can be generated with Quantum1Net's system, is designed to allow for secure data transmission now, and in a future where quantum computers exist which can surpass widely used encryption systems such as RSA or elliptic-curve cryptography. On the one hand, there is no underlying algorithm associated with the generation of the QEKG random numbers and derived encryption keys which can be exploited. On the other, they keys are never exchanged over the network and hence never exposed to eavesdroppers.

References

[1] Y. Wang, *"Public Key Cryptography Standards: PKCS"*, arxiv:1207.5446 (2012)
[2] M. Ebrahim, S. Khan and U.B. Khalid, "*Symmetric Algorithm Survey: A Comparative Analysis*", arxiv:1405.0398 (2013)
[3] W. K. Wootters and W. H. Zurek, "*A single quantum cannot be cloned*", Nature 299, 802-803 (1982)
[4] E. Diamanti, H. K. Lo, B. Qi and Z. Yuan, "*Practical challeges in quantum key distribution*", npk Quantum Information 2 16025 (2016)
[5] Y. Shih, "*Entangled biphoton source – property and perparation*", Rep. Prog. Phys. 66, 1009-1044 (2003)
[6] Quantum1Net, "*Cryptography for a Post-Quantum World Addressing the Security Challenges Ahead" (whitepaper)*, 2018, https://quantum1net.com/Q1N%20white%20paper.pdf
[7] Quantum1Net", "*Introducing a Non-Algorithmic, Computationally Irreducible, Scalable Data Encryption Framework" (yellow paper)*, 2017, https://quantum1net.com/

Q1N%20yellow%20paper.pdf [8] J. von Neumann, "*Mathematical Foundations of Quantum Mechanics*" Princeton University Press (1955)
[9] J. G. Ren et. al, "*Ground-to-satellite quantum teleportation*", Nature 549, 70-73 (2017)
[10] R. J. Hughes et. al, "*Practical quantum key distribution over a 48-km optical fiber network*", arXiv:quant-ph/9904038
[11] https://www.cqc2t.org/research/QuantumRepeater