A Practical and Scalable Implementation of the Vernam Cipher, under Shannon Conditions, using Quantum Noise

Results from an implementation of arbitrary-length key-distribution for a browser-integrated one-time pad

Adrian Neal
adrian.neal@oxfordscientifica.com

Oxford Scientifica

July 2024

Abstract

The one-time pad cipher is renowned for its theoretical perfect security, yet its practical deployment is primarily hindered by the key-size and distribution challenge. This paper introduces a novel approach to key distribution called *q-stream*, designed to make symmetric-key cryptography, and the one-time pad cipher in particular, a viable option for contemporary secure communications, and specifically, post-quantum cryptography, leveraging quantum noise and combinatorics to ensure secure and efficient key-distribution between communicating parties. We demonstrate that our key-distribution mechanism has a variable, yet quantifiable hardness of at least 504 bits, established from immutable mathematical laws, rather than conjectured-intractability, and how we overcome the one-time pad key-size issue with a localised quantum-noise seeded key-generation function, having a system hardness of at least 2304 bits, while introducing sender authentication and message integrity. Whilst the proposed solution has potential applications in fields requiring very high levels of security, such as military communications and large financial transactions, we show from our research with a prototype of *q-stream*, that it is sufficiently practical and scaleable for use in common browser-based web-applications, without any modification to the browser (i.e. plug-ins), running above SSL/TLS at the application level, where in tests, it achieved a key-distribution rate of around 7 million keys over a 5 minute surge-window, in a single (multi-threaded) instance of *q-stream*.

Keywords— one time pad, symmetric key distribution, quantum noise, combinatorics, forward secrecy

1 Introduction

In this paper, we present a novel method for symmetric-key distribution that has a quantifiable hardness of at least 504 bits, to which we refer to this scheme as *q-stream*, since it is based on the stream of a quantum random number generator (QRNG) . We demonstrate that this key-distribution mechanism makes the one-time-pad cipher[4], a viable option for general encryption requirements, due it's scalable nature, and introduce a simple modification to the one-time pad that addresses sender authentication and message integrity with a custom HMAC. We discuss the current state of public-key cryptography and how it is under threat from quantum computing and why current proposals for post-quantum (or quantum-safe) cryptography are failing to meet generally accepted levels of cryptographic robustness and how that is undermining forward secrecy.

Additionally, we argue that cryptographic solutions based upon conjectured intractability, rather than immutable laws of mathematics, will always carry the additional attack-vector that the intractability-conjecture may be false. So far, this has been the case for quantum-safe isogeny-based algorithms[1], and there are early signs[2] that this may soon be the case for quantum-safe lattice-based cryptography as well.

Finally, we demonstrate that attack-vectors for our key-distribution scheme are mitigated in its design and that the chosen mathematical principles upon which the scheme is based, are the primary source of the security hardness of the scheme, being by nature, immutable.

2 Motivation

The need for secure key distribution is critical as cyber threats continue to evolve. Current public-key cryptographic methods of key distribution (or encapsulation) face both classic and post-quantum threats, even for supposedly quantum-safe algorithms, raising the distinct possibility that public-key cryptography may become non-viable over the next decade, but quite possibly much sooner.

The logical upshot of this is that we may have to return to secret-key (symmetric) cryptography entirely, sooner or later, for which we will need new, secure and scalable key-distribution solutions. If a key distribution mechanism can overcome both the key-size distribution problem with one-time pads, as well as the inherent scalability problem, then it is worth pursuing.

3 Problems with Public-key based Key Exchange Mechanisms

3.1 The Quantum Threat

When the Cryptographically Relevant Quantum Computers (CRQC) era in quantum computing arrives, public-key cryptography will face the threat of quantum computers capable of executing algorithms such as Shor's algorithm[8], which are predicted to solve the problem of finding the prime factors of an integer in polynomial time, rendering current public key systems, including RSA and ECC, insecure. Initial estimates of the required number of qubits to execute Shor was $2n$, where n is the size, in bits, of the key being factored. Subsequent papers from Regev[6], followed by Vaikuntanathan and Ragavan[5], lowered this to $1.5n$, and then further with more efficient usage of memory, respectively.

3.2 Vulnerabilities in Proposed Quantum-Safe Algorithms

However, before the CRQC era, RSA and ECC will be replaced either partially or entirely, by quantum-safe algorithms, which should be capable of withstanding quantum cryptanalysis attack. While this is a positive step forward, these new quantum-safe algorithms are not immune to classic cryptanalysis, to which some of them have already failed[1], cryptographically, while others continue to display vulnerabilities to varying degrees[2][3]. In response to the quantum threat, the National Institute of Standards and Technology (NIST) have announced three new quantum-safe algorithms as the new standard for public-key cryptography. These are CRYSTALS–KYBER (FIPS-203[9]) for key encapsulation, together with CRYSTALS–Dilithium (FIPS-204[10]) and SPHINCS+ (FIPS-205[11]) for digital signatures.

Given the potential vulnerabilities and the impending quantum threat, there is a possibility that public-key cryptography could become non-viable as a secure form of cryptography. If this scenario materializes, symmetric-key cryptography might remain the only post-quantum alternative.

4 System Description of *q-stream*

Like Shannon's conditions[7] of the one-time pad, *q-stream* requires for its' security, that the output of the stream is only transmitted and used once (per recipient, per message). It operates under a registration system where users receive a one-time code (OTC) as a seed for generating a variable-length rotating user identifier (*rUID*). This *rUID* serves as a shared secret between *q-stream* and the user. Users also choose a public identity token, such as an email address, to facilitate identification by key requestors.

The system can be defined at a high level as a series of primary functions, which are User Registration, Requesting a Key and Receiving a Key. In reality, it is not the key that is sent or received, it is the key-generation material that will be used to create the key on the devices of the recipients.

4.1 User Registration

Before a user can request a key, they must first perform the registration process, whereby a unique *rUID* is created for one of their devices. Each device requires its own *rUID* and may hold multiple *rUID*s.

A users choose a public identity token ID_u such that:

$$\forall u \in U, \exists! \mathrm{ID}_u$$

Given:

- OTC is the one-time code of obituary length.
- U is the user-specific data, including the public identity token ID_u
- n is a counter
- H is a one-way hash function.
- LM is the last block of key generation material received from the *q-stream* service.

Step 1: Initialization

- Alice receives a one-time code OTC.
- Alice has user-specific data U.

Step 2: Generating the *rUID*

- Alice uses a counter n to keep track of the number of usages.
- Alice generates the *rUID* token using the formula:

$$\mathrm{rUID}_n = H(\mathrm{OTC} \| U \| LM \| n)$$

  where H is a one-way hash hash function.

Step 3: Usage and Rotation

- Every time Alice uses the *rUID* token, she increments the counter n to ensure the token is rotated.

4.2 Requesting a key

1. Alice (A) and Bob (B) are registered users with public identity tokens ID_A and ID_B.

2. Alice requests a key to communicate with Bob from *q-stream*.

3. *Q-stream* generates a random block R using a quantum random number generator (QRNG) with a size of 2,097,152 bits.

4. *Q-stream* generates 18 *rUID*s of each recipient in the key request.

5. *Q-stream* modifies R by inserting Alice's and Bob's *rUID*s at random positions, to which a 40-bit suffix P is concatenated, containing key generation material (*kGen*) location, size and ordinal position, to which a bitwise XOR operation is performed against a reserved part of the *rUID*, denoted as S, resulting in *PS*.

$$R = \text{QRNG}()$$

$$\forall i \in [1, 18], R_{A,i} = \text{insert}(R, rUID_A(t_i) \| PS)$$

$$\forall j \in [1, 18], R_{B,j} = \text{insert}(R, rUID_B(t_j) \| PS)$$

4.3 Receiving a key

1. The modified block R' is transmitted to both Alice and Bob.

2. Alice and Bob search, respectively, for their first *rUID*s and locate *PS*.

3. After performing an bitwise XOR operation of PS and S, the *kGen* location is obtained and then the *kGen* extracted.

4. Alice and Bob both rotate their respective *rUID*s in H and the process is repeated until the search for the next *rUID* fails.

5. Alice and Bob leave their *rUID*s in the final state that failed, ready to begin the search for it when a new R is received (from a new key request).

6. Alice and Bob concatenate the *kGen* material according to the stated ordinal position given in P

$$\text{Key}_A = \text{generate\_key}(\{R'_{A,i}\})$$

$$\text{Key}_B = \text{generate\_key}(\{R'_{B,j}\})$$

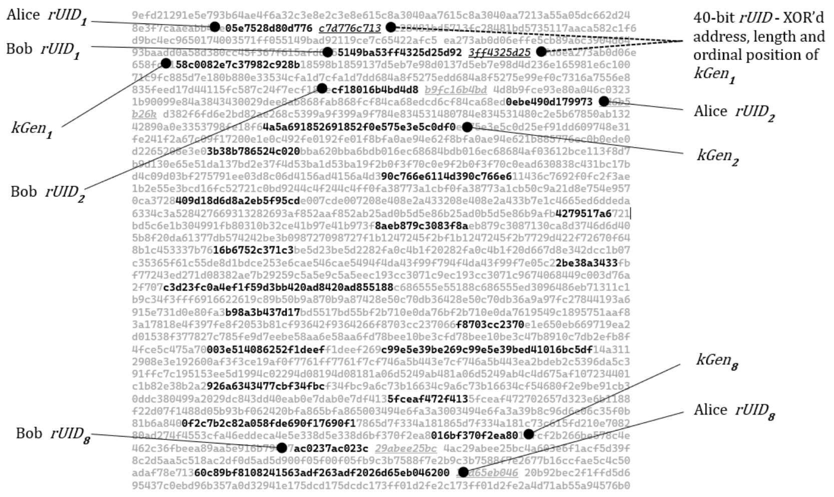

Figure 1: Visual part-rendering of q-stream

5 Protocol Definition for *q-stream* and One-Time Pad

1. **Registration**:
   - Alice (A) and Bob (B) register with *q-stream* and receive their OTCs.
   - Alice and Bob generate their $rUID$s: $rUID_A$ and $rUID_B$.
2. **Key Request**:
   - Alice requests a key for Bob.
   - *Q-stream* generates R using QRNG.
3. **Modification of** R:
   - *Q-stream* inserts Alice's current $rUID$ and rotate the $rUID$.
   - *Q-stream* repeats for 18 $rUID$s in total for both Alice and Bob, each.
4. **Transmission**:
   - *Q-stream* sends R' to both Alice and Bob.
5. **Key Extraction**:
   - Both idenitfy their own $rUID$s in R and derive $kGen$ block positions, sizes and concatenation order.
   - Both extract $kGen$ blocks and reorder them.
6. **Key Generation**

- Alice generates key K from the *kGen* material, of the same length as the message M that Alice intends to send to Bob.

7. **HMAC Generation**

   - Alice generates an HMAC of the message M using the key K_{HMAC}:

   $$\text{HMAC} = \text{HMAC}(K_{\text{HMAC}}, M)$$

   - The HMAC is a fixed-size output that ensures the authenticity and integrity of the message.

8. **Encryption**

   - Alice concatenates the message M and the HMAC to create M':

   $$M' = M \| \text{HMAC}$$

   - Alice performs a bitwise XOR operation between each bit of M' and the corresponding bit of the key K to produce the ciphertext C:

   $$C = M' \oplus K$$

   - Alice sends the ciphertext C to Bob.

9. **Decryption**

   - Bob receives the ciphertext C. He creates K in the same manner as Alice, to the length of C and performs a bitwise XOR operation to retrieve M':

   $$M' = C \oplus K$$

   - Bob separates M' into the message M and the HMAC:

   $$M' = M \| \text{HMAC}$$

10. **HMAC Verification**

    - Bob generates his own HMAC of the message M using the key K_{HMAC}:

    $$\text{HMAC}_B = \text{HMAC}(K_{\text{HMAC}}, M)$$

    - Bob compares his generated HMAC with the received HMAC. If they match, it verifies the integrity and authenticity of the message.

6 Algorithmic vs Foundational Hardness

6.1 Definitions

In cryptographic systems, the concept of security hardness can be understood from two perspectives:

- **Algorithmic hardness**, which we define as intractability-conjecture, with examples such as cryptographic algorithms like RSA and AES, and
- **Foundational hardness**, which we define as immutable pure mathematical laws, with the example of Vernam's' Cipher[12], or one-time pad, for which Shannon introduced the conditional notion of 'perfect secrecy'[7].

6.2 Algorithmic Hardness

Algorithmic hardness refers to the security provided by cryptographic algorithms, which rely on the computational difficulty of certain mathematical problems. For instance, the security of RSA encryption is based on the difficulty of factoring large prime numbers, while elliptic-curve cryptography (ECC) relies on the hardness of the elliptic-curve discrete logarithm problem.

However, the algorithmic hardness of these cryptographic methods inherently carries a non-zero risk of undiscovered weaknesses. Advances in mathematical research or computational power, such as the development of quantum computers, could potentially expose weaknesses in these algorithms. Consequently, while current cryptographic algorithms are deemed secure based on contemporary knowledge and technology, they are always susceptible to future breakthroughs that may render them obsolete or vulnerable.

6.3 Foundational Hardness

Foundational hardness, on the other hand, is grounded in principles of pure mathematics and number theory, which are considered immutable. In the context of *q-stream*, foundational hardness is derived from the use of quantum randomness and a combinatorial explosion.

6.3.1 Quantum Randomness

The unpredictability of quantum random number generators (QRNG) ensures that the initial random block R is fundamentally secure. QRNGs leverage quantum mechanical phenomena, which are inherently random and cannot be predicted or reproduced, providing a robust source of entropy.

6.3.2 Permutations

The process of breaking the *kGen* into the 18 blocks of random length, that are then inserted into the random block R at random positions, creates, given the chosen bounds of *kGen* and R, a combinatorial explosion of permutations, where finding the single correct permutation of *kGen* blocks is computationally infeasible.

6.4 Comparison

Algorithmic hardness is reliant on the current understanding and computational infeasibility of certain mathematical problems, which always carries a risk of being compromised by future discoveries. In contrast, foundational hardness, in the context of *q-stream*, is based on the intrinsic properties of randomness and combinatorial permutations, which are not subject to the same risks of computational advancements.

7 *Q-stream* Security Definition

We define the security of *q-stream* as a set of propositions that must be true for the system to be intrinsically secure:-

Let:

- Q represent the use of QRNG.
- R represent the random block generated by the QRNG.
- C represent the combinatorial explosion of permutations.
- D represent the random assignment of positions and lengths of *kGen* blocks.
- S represent the security of the system.
- P represent the property of unpredictability.

The security, based on both quantum randomness and the combinatorial explosion of permutations can be defined thus:

$$(Q \rightarrow P) \wedge ((P \wedge D) \rightarrow C) \wedge (C \rightarrow S)$$

Q-stream leverages foundational hardness by utilizing QRNGs and combinatorial permutation-explosion principles, ensuring that the security is not dependent on the assumed difficulty of mathematical problems but rather on the fundamental properties of randomness and mathematical theory. This distinction is critical, as it provides a more robust and enduring form of security that is less likely to be compromised by future technological advancements.

8 Claims of Hardness

To calculate the security hardness, we consider the combinatorial complexity of the *kGen*s placements and their sizes.

8.1 Combinatorial Complexity

For each of the 18 *kGen* blocks, the placement can vary across n bits, and each block can vary in length between 128 and 256 bits. The total number of possible placements is the number of ways to choose 18 positions out of 2,097,152, considering the varying lengths of each block:

$$\text{Combinations} = \frac{n!}{(n-18)!}$$

Additionally, considering the block lengths, we need to factor in the range for each block:

$$\text{Total Combinations} = \text{Combinations} \times 129^{18}$$

8.1.1 Overall Security Hardness

Given the complexity, the total hardness H_{total} is:

$$H_{total} = \log_2(\text{Total Combinations})$$

Assuming $n = 2,097,152$ bits:

$$\text{Combinations} = \frac{2,097,152!}{(2,097,152-18)!} \approx 2,097,152^{18}$$

$$\log_2(2,097,152^{18} \times 129^{18}) = 18\log_2(2,097,152 \times 129)$$

$$= 18\log_2(270,528,608) \approx 18 \times 28.0 \approx 504$$

$$H_{total} \approx 504 \text{ bits}$$

Thus, the security hardness of *q-stream*, for a QRNG block of 2097152 bits, is approximately 504 bits, assuming all bits contribute equally to the combinatorial complexity of the system.

Or to quantify an attack, the attacker would be required to calculate all possible sets of 18 *kGen* blocks within the QRNG block, having a computational complexity of 504 bits, thus being computationally infeasible.

8.2 Key Generation Process and Complexity

Initialization

Initialize the master hash as an empty string:

$$M_0 = ""$$

Step 1: Order the 18 kGen blocks

Order the 18 kGen blocks as per the original order in P:

$$\mathbf{B}_P = (B_{P(1)}, B_{P(2)}, \ldots, B_{P(18)})$$

Step 2: Hash the concatenation of the ordered kGen blocks and the master hash

For $n = 0$:

$$C_0 = B_{P(1)} \,||\, B_{P(2)} \,||\, \ldots \,||\, B_{P(18)}$$
$$H_0 = H(C_0 \,||\, M_0)$$

Step 3: Concatenate the new hash from step 2 with the master hash

$$M_1 = M_0 \,||\, H_0$$

Step 4: Repeat if necessary

If the master hash is smaller than the required key size K, reorder the 18 kGen blocks in a predetermined sequence and return to Step 2.

For $n \geq 1$:

$$\text{If } \mathrm{len}(M_n) < K, \text{ then:}$$
$$\mathbf{B}_{P_n} = \mathrm{reorder}(\mathbf{B}_P, n)$$
$$C_n = B_{P_n(1)} \,||\, B_{P_n(2)} \,||\, \ldots \,||\, B_{P_n(18)}$$
$$H_n = H(C_n \,||\, M_n)$$
$$M_{n+1} = M_n \,||\, H_n$$

Repeat steps 2 to 4 until:

$$\mathrm{len}(M_n) \geq K$$

In summary, the process can be expressed as:

1. Initialize:

$$M_0 = ""$$

2. For $n = 0$:

$$C_0 = B_{P(1)} \,||\, B_{P(2)} \,||\, \ldots \,||\, B_{P(18)}$$
$$H_0 = H(C_0 \,||\, M_0)$$
$$M_1 = M_0 \,||\, H_0$$

3. For $n \geq 1$:

$$\text{If } \mathrm{len}(M_n) < K, \text{ then:}$$
$$\mathbf{B}_{P_n} = \mathrm{reorder}(\mathbf{B}_P, n)$$
$$C_n = B_{P_n(1)} \,||\, B_{P_n(2)} \,||\, \ldots \,||\, B_{P_n(18)}$$

$$H_n = H(C_n \,||\, M_n)$$

$$M_{n+1} = M_n \,||\, H_n$$

4. Until:

$$\text{len}(M_n) \geq K$$

8.2.1 Measuring Key Predictability

For arbitary-length keys, we state that any knowledge of part of the key does not lead to knowledge of either any other part of the key, or any of the key-generation material.

While a long key will be the concatenation of multiple hashes, in general, one hash cannot be used to directly determine another hash, especially for cryptographic hash functions such as SHA-1, SHA-256, or others. This is due to several fundamental properties of cryptographic hash functions, which ensure their security and integrity.

8.2.2 Preimage Resistance

Given a hash output h, it should be computationally infeasible to find any input x such that $H(x) = h$. Formally, this is expressed as:

$$\forall h \in \text{Range}(H), \Pr[\text{find } x \text{ such that } H(x) = h] \leq \epsilon$$

where ϵ is a very small probability. Furthermore, in this particular scenario, it is not enough to find *any* input, as the attack needs to find a specific input. Given that the input, for any segment of the generated key, is at least 2304 bits, any and all keys of any length will have a key-generation hardness of over 2304 bits, usually much more.

8.2.3 Second Preimage Resistance

Given an input x_1 and its hash $H(x_1)$, it should be computationally infeasible to find a different input x_2 such that $H(x_1) = H(x_2)$. Formally:

$$\forall x_1, \Pr[\text{find } x_2 \neq x_1 \text{ such that } H(x_1) = H(x_2)] \leq \epsilon$$

8.2.4 Collision Resistance

It should be computationally infeasible to find two different inputs x_1 and x_2 such that $H(x_1) = H(x_2)$. Formally:

$$\Pr[\text{find } x_1 \neq x_2 \text{ such that } H(x_1) = H(x_2)] \leq \epsilon$$

9 Attack Vectors

The primary attack vector in the *q-stream* system is the compromise of the *rUID* for any recipient. Access to the *rUID* compromises all future communication for that recipient because the *rUID* is used in the generation of subsequent keys. However, prior communication remains secure because the rotation of the *rUID* each time it is used, utilises a one-way hash function that cannot be reversed. This is because the rotation function intentionally loses information, making it infeasible to reconstruct the original *rUID* from its hashed form.

9.1 One-Way Hash Functions

One-way hash functions are mathematical functions that take an input (or 'message') and return a fixed-size string of bytes. The output is typically a 'digest' that is unique to each unique input. The key property of one-way hash functions is that they are designed to be irreversible. It should be computationally infeasible to reverse the process and obtain the original input given only the output. This is achieved by ensuring that the function loses information during the hashing process, creating a scenario where many different inputs could produce the same output, but it is not possible to determine which one was the original.

In *q-stream*, this property is used to ensure that once an *rUID* is rotated and hashed, the previous *rUID* cannot be deduced from the new one, securing past communications against future compromise.

9.2 User Registration

Since the user registration involves the exchange of a one-time code (OTC) and the setup of the initial *rUID*, it is a single one-time event. In a general harvesting attack vector, the likelihood of correlating the registration communication to future key requests is exceptionally low, with additional mitigation's possible, such as out-of-band registration, if this was deemed to pose a risk.

9.3 Traffic Harvesting for Later Decryption

9.3.1 Using *q-stream* with Common Symmetric Cryptographic Algorithms

Attackers may attempt to harvest encrypted traffic with the intention of decrypting it at a later date, when they potentially have more computational power or new cryptographic breakthroughs. This will always remain a risk.

9.3.2 Using *q-stream* with the One-Time Pad

Traffic harvesting is not a threat when *q-stream* is used with the one-time pad, as described in this paper with the addition of the HMAC.

9.3.3 Attacking *q-stream* Directly with Traffic Harvesting

If *q-stream* traffic is the target of the traffic harvesting, we have already calculated that it has a hardness of 504 bits; harder than most symmetric algorithms, but not as hard as the one-time pad.

10 Protocol Variations

10.1 Variation 1: Secret Inclusion in Key Generation

In the first variation, the key generation function includes a secret that only the recipients know, but not the *q-stream* service. This secret could be a pre-shared word, group of words or phrase, between the recipients that is combined with the *kGen* blocks extracted from R'.

1. **Registration**:
    - Alice (A) and Bob (B) register with *q-stream* and exchange a secret S securely.
2. **Key Generation**:
    - Alice and Bob extract the *kGen* blocks from R' as usual.
    - They combine these blocks with the secret S using a secure key derivation function (*KDF*):

$$\text{Key} = KDF(\{R'_{A,i}\}, S)$$

This ensures that the final key is known only to Alice and Bob, and not to the *q-stream* service.

10.2 Variation 2: Agreed *kGen* Block Locations

In the second variation, the recipients agree between themselves on the chosen locations in R for the *kGen* blocks. This agreement can be done securely using an initial pre-shared secret or an out-of-band communication method.

1. **Registration**:
    - Alice and Bob agree on the locations in the R for the *kGen* blocks.
2. **Key Request**:
    - Alice requests a key for Bob from *q-stream* without disclosing the agreed locations.
3. **Key Generation**:
    - *Q-stream* generates R as usual and sends it to both Alice and Bob.

- Alice and Bob extract the *kGen* blocks from the agreed locations within R:

$$\text{Key} = \text{generate\_key}(\{R_{A,\text{agreed\_locations}}\})$$

In both cases, the *q-stream* service has no knowledge of the final keys generated, presenting a complete point-to-point key distribution scheme where only the intended recipients have knowledge of the final key.

11 Varying the Computational Hardness

The specific computational hardness of *q-stream* can be varied by changing the size of R or the count of *kGen* blocks, or it's length and size bounds.

11.1 Variation 1: Increasing the size of R

If we double the size of the QRNG block R, then:-

Assuming $n = 4,194,304$ bits:

$$\text{Combinations} = \frac{4,194,304!}{(4,194,304-36)!} \approx 4,194,304^{18}$$

$$\log_2(4,194,304^{36} \times 129^{36}) = 36\log_2(4,194,304 \times 129)$$

$$= 36\log_2(541,085,216) \approx 36 \times 29.0 \approx 522$$

$$H_{total} \approx 522 \text{ bits}$$

11.2 Variation 2: Increasing the Number of *kGen* Blocks

If we further double the number of *kGen* blocks, then:-

Assuming $n = 4,194,304$ bits:

$$\text{Combinations} = \frac{4,194,304!}{(4,194,304-36)!} \approx 4,194,304^{36}$$

$$\log_2(4,194,304^{36} \times 129^{36}) = 36\log_2(4,194,304 \times 129)$$

$$= 36\log_2(541,085,216) \approx 36 \times 29.0 \approx 1044$$

$$H_{total} \approx 1044 \text{ bits}$$

11.3 Variation 3: Increasing the Lower and Upper Bounds of *kGen* Blocks

If we then further double the distance between the lower and upper bounds of the size of the *kGen* blocks, then:-

Assuming $n = 4,194,304$ bits:

$$\text{Combinations} = \frac{4,194,304!}{(4,194,304 - 36)!} \approx 4,194,304^{36}$$

$$\log_2(4,194,304^{36} \times 257^{36}) = 36 \log_2(4,194,304 \times 257)$$

$$= 36 \log_2(1,078,409,408) \approx 36 \times 30.0 \approx 1080$$

$$H_{total} \approx 1080 \text{ bits}$$

11.4 Summary of Variation Impact

Where the increase of hardness doubles from 522 bits to 1044 bits, indicates that it is the number of *kGen* block in the scheme that has the most impact on the overall hardness. Comparatively, for variations 1 and 3, the increase was relatively minimal.

12 Results from SSL/TLS Browser Prototype Implementations

Prototype implementations of *q-stream* have shown that its performance makes it feasible for many applications. In tests using AWS as the host of *q-stream*, the rate of key-distribution was approximately 25,000 keys per second. Because a web-browser session requires only one call to *q-stream* to be able to generate unique keys for each transmission during a session, a single *q-stream* instance can support a significant number of users simultaneously.

The integration of *q-stream* and it's modified one-time pad implementation into a web-browser did not require any modification of the browser, such as additional plugins, as maintaining user key-state was accomplished using existing out-of-the-box browser capabilities, with *q-stream* being implemented at the web-application level, on top of, and without interference to the existing SSL/TLS communications layer.

The result of using q-stream in this way means that such traffic has an additional lay of encryption, should either classic or quantum-safe cryptography become vulnerable to attack.

A typical example of expected performance, would be the number of key requests that it could support over a 5 minute period when there is a surge of users, such as certain times in the morning when messages and emails are checked. In our tests, a single instance of *q-stream* was capable of supporting over 7 million users within that 5 minute surge-window, although probably a lot more with implementation refinements.

13 Conclusion

Q-stream presents a novel approach to symmetric-key distribution by leveraging quantum randomness and rotating user identifiers. This system offers enhanced security, achieving a hardness significantly greater than traditional methods.

The distinction between algorithmic and foundational hardness further underscores the robustness of *q-stream*, as it relies on immutable mathematical principles rather than the assumed difficulty of computational problems.

Consequently, q-stream is capable of providing reliable forward-secrecy of messages, if the encryption algorithm for which the keys have been generated, is also cryptographically robust. It will increase the security of most symmetric cryptographic schemes, but will have the most value if deployed with the variation of the one-time pad as described in this paper.

Its practicality and scalabilty was demonstrated in its simple integration into common web-browsers, providing elevated security for highly confidential transmissions, neutralising traffic-harvesting threats against forward-secrecy.

Future work will focus on practical implementations and further analysis of the protocol's resilience against various attack vectors. Additionally, exploring protocol variations can further enhance security by ensuring that only the intended recipients have knowledge of the final generated keys, making *q-stream* a versatile and robust solution for secure key distribution in the quantum era.

References

[1] Wouter Castryck and Thomas Decru. *An efficient key recovery attack on SIDH*. Cryptology ePrint Archive, Paper 2022/975. `https://eprint.iacr.org/2022/975`. 2022. URL: `https://eprint.iacr.org/2022/975`.

[2] Yilei Chen. *Quantum Algorithms for Lattice Problems*. Cryptology ePrint Archive, Paper 2024/555. `https://eprint.iacr.org/2024/555`. 2024. URL: `https://eprint.iacr.org/2024/555`.

[3] Emre Karabulut and Aydin Aysu. "FALCON Down: Breaking FALCON Post-Quantum Signature Scheme through Side-Channel Attacks". In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 2021, pp. 691–696. DOI: `10.1109/DAC18074.2021.9586131`.

[4] Thomas Lugrin. "One-Time Pad". In: *Trends in Data Protection and Encryption Technologies*. Ed. by Valentin Mulder et al. Cham: Springer Nature Switzerland, 2023, pp. 3–6. ISBN: 978-3-031-33386-6. DOI: `10.1007/978-3-031-33386-6_1`. URL: `https://doi.org/10.1007/978-3-031-33386-6_1`.

[5] Seyoon Ragavan and Vinod Vaikuntanathan. *Space-Efficient and Noise-Robust Quantum Factoring*. Cryptology ePrint Archive, Paper 2023/1501. `https://eprint.iacr.org/2023/1501`. 2023. URL: `https://eprint.iacr.org/2023/1501`.

[6] Oded Regev. *An Efficient Quantum Factoring Algorithm*. `https://arxiv.org/abs/2308.06572`. 2024. arXiv: `2308.06572 [quant-ph]`. URL: `https://arxiv.org/abs/2308.06572`.

[7] Claude Shannon. *Communication Theory of Secrecy Systems*. Bell System Technical Journal. `https://www.cs.virginia.edu/~evans/greatworks/shannon1949.pdf`. 1949. URL: `https://www.cs.virginia.edu/~evans/greatworks/shannon1949.pdf`.

[8] P.W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134. DOI: `10.1109/SFCS.1994.365700`.

[9] National Institute of Standards and Technology. *Module-Lattice-based Key-Encapsulation Mechanism Standard*. Tech. rep. Federal Information Processing Standards Publication (FIPS PIBS) 203. Washington, D.C.: U.S. Department of Commerce, 2024. DOI: `10.6028/NIST.FIPS.203.ipd`.

[10] National Institute of Standards and Technology. *Module-Lattice-based Key-Encapsulation Mechanism Standard*. Tech. rep. Federal Information Processing Standards Publication (FIPS PIBS) 204. Washington, D.C.: U.S. Department of Commerce, 2024. DOI: `10.6028/NIST.FIPS.204.ipd`.

[11] National Institute of Standards and Technology. *Module-Lattice-based Key-Encapsulation Mechanism Standard*. Tech. rep. Federal Information Processing Standards Publication (FIPS PIBS) 205. Washington, D.C.: U.S. Department of Commerce, 2024. DOI: `10.6028/NIST.FIPS.205.ipd`.

[12] Gilbert Vernam. *One-Time Pad (OTP)*. Cryptomuseum.com. Archived from the original on 2014-03-14. Retrieved 2014-03-17. `https://web.archive.org/web/20140314175211/http://www.cryptomuseum.com/crypto/otp.htm`. 1917. URL: `http://www.cryptomuseum.com/crypto/otp.htm`.