Qualitative Frame Entanglement (QFE): An Experimental Quantum Secure Communication Protocol

n12n, Gemini

Abstract

This document describes the current implementation of the Qualitative Frame Entanglement (QFE) experimental secure communication framework, realized in Rust. QFE establishes secure communication sessions by integrating standard, modern cryptographic primitives. Key establishment leverages the NIST standard post-quantum key encapsulation mechanism ML-KEM-1024 (Kyber) to generate a shared secret resistant to known quantum computing attacks. Following the key exchange, session keys are derived using the HKDF-SHA512 key derivation function with appropriate context separation. Communication security, encompassing confidentiality, integrity, and authenticity, is provided by the ChaCha20-Poly1305 Authenticated Encryption with Associated Data (AEAD) cipher. While the framework retains conceptual 'Frame' initialization deriving unique state via cryptographic hashing, its security foundation now rests upon the standard computational hardness assumptions underlying ML-KEM, HKDF, SHA-512, and ChaCha20-Poly1305. The library also includes experimental Zero-Knowledge Proof features (Schnorr protocol) bound to the session context. This paper details the updated architecture and cryptographic components of the QFE framework.

1 Introduction

The Qualitative Frame Entanglement (QFE) project provides an experimental Rust framework developed to simulate and explore secure communication protocols. Originally conceived from foundational principles exploring concepts like distinction, reference frames, and coherent interaction, the framework aimed to derive security properties directly from these conceptual underpinnings rather than relying on conventional computational hardness assumptions.

However, through analysis and development (as of April 7, 2025), the implementation has significantly evolved to ensure robust security guarantees aligned with modern cryptographic best practices. While retaining the 'Frame' abstraction to manage participant state and unique identifiers (now derived using cryptographic hashing), the core security mechanisms of QFE have been rebuilt using well-established, standardized primitives. Security in the current QFE implementation is based on the computational hardness assumptions of these underlying components, representing a pivot from the initial philosophical premise.

Specifically, QFE now utilizes:

- **ML-KEM-1024 (Kyber):** The NIST standard Key Encapsulation Mechanism for post-quantum secure key establishment, protecting the initial shared secret against attacks from both classical and future quantum computers.
- **HKDF-SHA512:** The standard HMAC-based Key Derivation Function (RFC 5869) to derive specific cryptographic keys (e.g., for AEAD) from the master shared secret generated by ML-KEM.

- **ChaCha20-Poly1305:** A standard, high-performance Authenticated Encryption with Associated Data (AEAD) cipher (RFC 8439) providing confidentiality, data integrity, and message authenticity for all communications subsequent to key establishment.
- **SHA-512:** Used for cryptographic hashing during Frame initialization, key derivation (within HKDF), and potentially within Zero-Knowledge Proof components.

The framework also incorporates experimental Zero-Knowledge Proof features (e.g., Schnorr protocol) to demonstrate advanced cryptographic capabilities within the QFE context.

This document details the revised architecture, cryptographic components, and security basis of the current QFE implementation. It reflects a pragmatic approach, leveraging standard, vetted cryptography to provide strong security guarantees while utilizing the QFE 'Frame' concept primarily for state management and participant representation within the simulation.

2 Core Cryptographic Mechanisms

The security of the QFE framework, in its current implementation, is achieved through the layered application of standard, well-vetted cryptographic primitives, rather than the emergent properties described in its initial conceptualization. This section outlines the core mechanisms employed.

2.1 Frame Initialization and State

Each participant in the communication is represented by a 'Frame' structure. While drawing conceptual inspiration from distinct reference frames, the implementation ensures unique initialization state through deterministic cryptographic hashing. Upon calling 'Frame::initialize' with a participant ID and an initial seed value, unique internal identifiers (representing the conceptual 'DistinctionNode' and 'ReferenceFrame') and an initial state are derived using SHA-512 with appropriate domain separation. This ensures that distinct initial parameters lead to distinct Frame states, providing a basis for participant representation and context management within the cryptographic protocols.

2.2 Post-Quantum Key Establishment and Derivation (SQS)

Secure communication requires establishing a shared secret context, represented by the 'Sqs' (Shared Qualitative Structure) struct, between communicating Frames (e.g., Alice and Bob). This is achieved via a standard post-quantum key establishment protocol followed by key derivation:

Key Encapsulation (ML-KEM-1024): To establish an initial shared secret secure against both classical and known quantum attacks, QFE employs the ML-KEM-1024 (Kyber) algorithm, selected by NIST for post-quantum standardization. The process involves:

1. The initiator (Alice) generates an ML-KEM key pair (pk_A, sk_A).
2. Alice transmits her public key pk_A to the responder (Bob).
3. Bob uses pk_A to perform the ML-KEM encapsulation operation. This generates a shared secret (ss_B) and a ciphertext (ct).
4. Bob transmits the ciphertext ct back to Alice.

5. Alice uses her secret key sk_A to perform the ML-KEM decapsulation operation on ct, yielding her view of the shared secret (ss_A).

Upon successful completion, both parties hold an identical shared secret ($ss_A = ss_B$), typically 32 bytes, established securely.

Key Derivation (HKDF-SHA512): The raw shared secret (ss) obtained from ML-KEM is used as Input Keying Material (IKM) for the standard HMAC-based Key Derivation Function (HKDF, RFC 5869) using SHA-512 as the hash function.

- A salt for HKDF is first derived by hashing participant IDs (sorted lexicographically) and a provided context string using SHA-512, ensuring the derived keys are bound to the specific session participants and context.
- HKDF-Extract is applied using the salt and the IKM (ss) to produce a strong Pseudorandom Key (PRK).
- HKDF-Expand is then used with the PRK and distinct "info" strings to derive the necessary cryptographic material stored within the ‘Sqs‘ struct:
  - A 64-byte general-purpose secret byte sequence (‘Sqs.components‘), derived with info ‘"QFE_SQS_COMPONENTS_V2"‘.
  - A 32-byte specific key (‘Sqs.aead_key‘) for use with ChaCha20-Poly1305, derived with info ‘"QFE_AEAD_KEY_V1"‘.

The resulting ‘Sqs‘ struct contains this derived key material and participant identifiers, forming the secure context for the session. Fields related to the original QFE concepts of shared phase lock and resonance frequency are no longer present or used in this security model.

2.3 Authenticated Encryption for Communication

All subsequent communication requiring confidentiality and integrity between Frames possessing a shared ‘Sqs‘ context utilizes the ChaCha20-Poly1305 AEAD cipher (RFC 8439).

- The cipher is keyed using the specific ‘Sqs.aead_key‘ derived via HKDF.
- For each message encryption, a unique 12-byte nonce must be generated (typically randomly). This nonce is transmitted in plaintext alongside the ciphertext. **Nonce uniqueness per key is critical for security.**
- The AEAD operation provides strong guarantees for:
  - **Confidentiality:** Plaintext is encrypted into ciphertext.
  - **Integrity:** Any modification to the ciphertext or associated data during transmission will be detected upon decryption.
  - **Authenticity:** Successful decryption verifies that the message originated from a party holding the shared ‘Sqs.aead_key‘.
- Optional Associated Data (AD) can be included, which is authenticated along with the ciphertext but remains unencrypted.

The output of encryption is packaged (e.g., as ‘nonce‘ and ‘ciphertext+tag‘) for transmission. Decryption takes these components and returns the original plaintext only if all checks (including tag verification) pass.

2.4 Zero-Knowledge Proofs (Optional)

The framework includes experimental support for Zero-Knowledge Proofs (ZKPs), such as the Schnorr protocol implemented using the Ristretto group. When used, the Fiat-Shamir transformation derives challenges by hashing relevant public proof components (e.g., public key, commitment) along with session context identifiers (such as sorted participant IDs from the ‘Sqs‘) to ensure proofs are bound to the specific QFE session and context.

3 The QFE Algorithm Steps

The QFE protocol simulation involves the following distinct stages, now utilizing standard cryptographic primitives for session establishment and communication:

3.1 Frame Initialization

Participant Frames, representing entities like a sender (A) and a receiver (B), are initialized using the `Frame::initialize(id, initial_seed)` function. Each Frame is instantiated with a unique state derived deterministically from its identifier (`id`) and seed using SHA-512. This process generates internal identifiers (conceptually representing $\Omega(x)$ and $R(\Omega)$) ensuring distinct initial states but does not contribute directly to session key secrecy. A Frame begins in a valid state but without an established shared context (`Sqs`).

3.2 Session Establishment (SQS via KEM and HKDF)

A secure shared context (`Sqs`) is established between two Frames (e.g., A and B) using the `establish_sqs_kem` function, which simulates a Post-Quantum secure key exchange and derives session keys. This process replaces the previous custom interaction logic. The simulated steps are:

1. **KEM Key Generation:** The initiating Frame (A) generates an ML-KEM-1024 key pair (pk_A, sk_A) using `mlkem1024::keypair()`.
2. **KEM Encapsulation:** The responding Frame (B) conceptually receives pk_A and uses it to encapsulate a shared secret via `mlkem1024::encapsulate(&pk_A)`, obtaining a shared secret (ss_B) and a ciphertext (ct).
3. **KEM Decapsulation:** Frame A conceptually receives ct and uses its secret key sk_A to decapsulate via `mlkem1024::decapsulate(&ct, &sk_A)`, obtaining its view of the shared secret (ss_A).
4. **Verification:** The implementation verifies that ss_A and ss_B are identical. A mismatch indicates a critical error. The resulting verified shared secret is denoted ss.
5. **Salt Derivation:** A salt for HKDF is derived using SHA-512, hashing a domain separator, the sorted participant IDs (from `frame_a` and `frame_b`), and an application-provided context string.
6. **HKDF-Extract:** HKDF-Extract (using SHA-512) is applied to the KEM shared secret ss (as IKM) and the derived salt, producing a Pseudorandom Key (PRK).
7. **HKDF-Expand (Components):** HKDF-Expand (using SHA-512) is applied to the PRK with the info string `"QFE_SQS_COMPONENTS_V2"` to derive 64 bytes of general-purpose session key material $(SQS_{\text{components}})$.

8. **HKDF-Expand (AEAD Key):** HKDF-Expand is applied again to the *same* PRK but with the distinct info string `"QFE_AEAD_KEY_V1"` to derive the specific 32-byte AEAD key (k_{AEAD}) for ChaCha20-Poly1305.
9. **SQS Creation:** An `Sqs` struct is created containing the derived $SQS_{\text{components}}$, k_{AEAD}, participant IDs, and validation status.
10. **SQS Storage:** An atomically reference-counted pointer (`Arc<Sqs>`) to the created `Sqs` instance is stored within both `frame_a` and `frame_b`, establishing their shared context.

After successful execution, it is recommended (though simulated here) that participants compare SQS fingerprints out-of-band to mitigate potential Man-in-the-Middle attacks during the conceptual exchange of pk_A and ct.

3.3 Authenticated Encryption (AEAD Encode)

To send a message confidentially and with integrity, the sending Frame (A) uses the `encode_aead(plaintext, associated_data)` method:

1. Retrieve the shared `Sqs` context, specifically the AEAD key k_{AEAD}. Fail if no SQS is established or the Frame is invalid.
2. Generate a unique 12-byte random nonce N using a cryptographically secure random number generator. Nonce uniqueness per key is mandatory.
3. Instantiate the ChaCha20-Poly1305 AEAD cipher using k_{AEAD}.
4. Encrypt the `plaintext` using the cipher, nonce N, and optional `associated_data` (AD). This operation produces the ciphertext C and a 16-byte authentication tag T.
5. Package the nonce N and the combined ciphertext and tag $C||T$ into a `QfeEncryptedMessage` structure.
6. Return the `QfeEncryptedMessage`.

3.4 Transmission

The sender (A) transmits the `QfeEncryptedMessage` (containing the nonce and the ciphertext+tag) to the receiver (B) over the communication channel.

3.5 Authenticated Decryption (AEAD Decode)

Upon receiving a `QfeEncryptedMessage`, the receiving Frame (B) uses the `decode_aead(&encrypted_message, associated_data)` method:

1. Retrieve the shared `Sqs` context and the AEAD key k_{AEAD}. Fail if no SQS is established or the Frame is invalid.
2. Extract the nonce N and the combined ciphertext+tag $C||T$ from the received `encrypted_message`. Verify nonce length (12 bytes).
3. Separate the combined $C||T$ into the ciphertext C and the 16-byte tag T. Fail if the combined length is less than 16 bytes.

4. Instantiate the ChaCha20-Poly1305 AEAD cipher using k_{AEAD}.
5. Attempt to decrypt and verify the ciphertext C using the cipher, nonce N, the same optional `associated_data` (AD) used during encryption, and the tag T.
6. If the tag T is valid for the key, nonce, ciphertext, and AD, the decryption succeeds and the original `plaintext` is returned.
7. If the tag verification fails (indicating tampering, wrong key, wrong nonce, or wrong AD), the decryption fails, an error (`QfeError::DecodingFailed`) is returned, and the receiving Frame's validation status is set to false.

4 Security Analysis

The security guarantees of the revised Qualitative Frame Entanglement (QFE) framework are now predicated on the well-established security properties of the standard cryptographic primitives employed, rather than the conceptual principles of the original QFE proposal. The analysis assumes correct implementation and proper usage (e.g., nonce management) of these primitives.

4.1 Key Establishment (ML-KEM-1024)

The establishment of the initial shared secret (ss) relies on the ML-KEM-1024 (Kyber) algorithm.

- **Security Goal:** To establish a shared secret between two parties (Alice and Bob) such that an eavesdropper observing the exchanged messages (Alice's public key pk_A, Bob's ciphertext ct) cannot compute the secret ss. This security should hold even against adversaries equipped with quantum computers.
- **Security Property:** ML-KEM is designed to achieve IND-CCA2 (Indistinguishability under Adaptive Chosen Ciphertext Attack) security. This is the standard security definition for Key Encapsulation Mechanisms.
- **Underlying Assumption:** The security of ML-KEM relies on the computational hardness of the Module Learning With Errors (Module-LWE) problem over specific algebraic structures. Module-LWE is widely believed to be resistant to efficient attacks by both classical and known quantum algorithms (including Shor's algorithm).
- **Result:** Assuming the hardness of Module-LWE, the shared secret ss established via the ML-KEM exchange is computationally secure against passive and active attackers (in the IND-CCA2 model) on the key exchange messages, including quantum attackers.

4.2 Key Derivation (HKDF-SHA512)

The shared secret ss from ML-KEM is used as Input Keying Material (IKM) for HKDF-SHA512 to derive session-specific keys (`Sqs.components` and `Sqs.aead_key`).

- **Security Goal:** To derive multiple strong cryptographic keys from a single initial shared secret, such that the derived keys are computationally indistinguishable from random strings and independent of each other.

- **Security Property:** HKDF (RFC 5869) is designed as a secure Key Derivation Function (KDF). When instantiated with a secure hash function like SHA-512, it acts as a Pseudo-Random Function (PRF). The Extract step concentrates potentially non-uniform entropy from the IKM into a fixed-size Pseudorandom Key (PRK), while the Expand step generates the required output keying material.
- **Underlying Assumption:** Relies on the security of the underlying hash function (SHA-512) and the PRF-security of the HMAC construction used internally by HKDF.
- **Result:** The derived keys (`Sqs.components`, `Sqs.aead_key`) are cryptographically strong and suitable for their intended purposes (general session context, AEAD key). The use of distinct context-specific "info" strings during HKDF-Expand ensures cryptographic separation between the derived keys. An attacker without knowledge of the initial secret ss cannot compute these keys.

4.3 Communication Security (ChaCha20-Poly1305 AEAD)

Messages are protected using the ChaCha20-Poly1305 AEAD cipher, keyed by `Sqs.aead_key`.

- **Security Goal:** To provide confidentiality (preventing eavesdroppers from reading message content), integrity (preventing undetected modification of messages), and authenticity (ensuring messages originate from the key holder).
- **Security Property:** ChaCha20-Poly1305 provides AEAD security, satisfying standard notions like IND-CPA (or stronger) for confidentiality and INT-CTXT for integrity/authenticity.
- **Underlying Assumption:** Relies on the assumed security of the ChaCha20 stream cipher as a pseudorandom function and the Poly1305 authenticator as a secure message authentication code (MAC). A critical operational assumption is the **uniqueness of the nonce** used for each encryption operation performed with the same key. Nonce reuse catastrophically breaks both confidentiality and authenticity. The implementation uses randomly generated 12-byte nonces, which provides high probability of uniqueness.
- **Result:** Messages encrypted with `encode_aead` are protected against eavesdropping and tampering. Successful decryption with `decode_aead` assures the receiver of the message's confidentiality, integrity, and origin from the party holding the corresponding session key, provided nonces are managed correctly.

4.4 Limitations and Further Considerations

- **Man-in-the-Middle (MitM) during Key Establishment:** The base ML-KEM protocol, like Diffie-Hellman, does not inherently protect against an active MitM attacker during the exchange of the public key (pk_A) and ciphertext (ct). An attacker could potentially intercept and replace these messages. Preventing this requires either performing the exchange over a pre-authenticated channel or verifying the established session context out-of-band (e.g., using the SQS fingerprint derived from the final `Sqs.components`).
- **Forward Secrecy:** Since `establish_sqs_kem` generates fresh ML-KEM key pairs for each call via `mlkem1024::keypair()`, the resulting session keys (`Sqs.components`, `Sqs.aead_key`) benefit from forward secrecy. Compromise of long-term identity keys (if any existed) would not compromise past session keys established this way.

- **Implementation Security:** This analysis assumes the underlying cryptographic libraries (`pqcrypto`, `hkdf`, `chacha20poly1305`, `sha2`) are correctly implemented and that the QFE integration uses them properly (e.g., nonce uniqueness, correct handling of inputs/outputs). Implementation vulnerabilities (side channels, buffer overflows, etc.) are outside this scope.

In conclusion, the security of the revised QFE framework now rests on the well-understood foundations of standard, modern cryptography, including post-quantum key establishment via ML-KEM and authenticated encryption via ChaCha20-Poly1305. The previous security arguments based on phase modulation and coherence are no longer applicable.

5 Conclusion

The Qualitative Frame Entanglement (QFE) framework, as described in this document reflecting its implementation as of April 7, 2025 (April 7, 2025), provides an experimental platform in Rust for simulating secure communication protocols. It demonstrates the successful integration of standard, high-assurance cryptographic components to achieve robust security guarantees suitable for modern applications, including consideration for post-quantum threats.

Secure sessions within QFE are now established using the NIST standard post-quantum Key Encapsulation Mechanism ML-KEM-1024 (Kyber), ensuring that the foundational shared secrets resist attacks even from adversaries possessing quantum computers. These secrets are processed using the standard HKDF-SHA512 Key Derivation Function to generate specific, context-bound keys for communication. Subsequent message exchange is protected by the ChaCha20-Poly1305 Authenticated Encryption with Associated Data (AEAD) scheme, providing strong guarantees of confidentiality, data integrity, and message authenticity. The framework also incorporates experimental Zero-Knowledge Proof capabilities, demonstrating the potential for advanced cryptographic features within the QFE structure.

This architecture represents a significant evolution from the initial QFE concepts that sought to derive security properties from foundational or philosophical principles related to coherence and phase modulation. While the 'Frame' abstraction is retained for managing participant state and identity (using cryptographic hashing during initialization), the security of the communication protocol now firmly relies on the well-understood computational hardness assumptions associated with the underlying standard cryptographic primitives: ML-KEM, HKDF, SHA-512, and ChaCha20-Poly1305.

Consequently, the revised QFE provides post-quantum secure key establishment and strong authenticated encryption, aligning with current cryptographic best practices. Future work could involve exploring alternative PQC KEMs or AEAD schemes, further developing the ZKP features, formally analyzing the security of the integrated protocol flow, and designing robust mechanisms for the authenticated exchange of initial KEM messages to prevent Man-in-the-Middle attacks.

In its current state, QFE serves as a valuable experimental framework demonstrating the practical integration of post-quantum key establishment with modern authenticated encryption within a simulated, stateful communication context.

References

[1] National Institute of Standards and Technology (NIST). *FIPS PUB 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard.* August 2023. Available: `https://csrc.nist.gov/pubs/fips/203/final`

[2] National Institute of Standards and Technology (NIST). *FIPS PUB 180-4: Secure Hash Standard (SHS)*. August 2015. Available: `https://csrc.nist.gov/pubs/fips/180-4/final`

[3] Krawczyk, H. and Eronen, P. *RFC 5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. IETF, May 2010. Available: `https://www.rfc-editor.org/info/rfc5869`

[4] Nir, Y. and Langley, A. *RFC 8439: ChaCha20 and Poly1305 for IETF Protocols*. IETF, June 2018. Available: `https://www.rfc-editor.org/info/rfc8439`

[5] Hamburg, M., et al. *draft-irtf-cfrg-ristretto255-04: The Ristretto Prime-Order Group*. Internet Research Task Force (IRTF) Crypto Forum Research Group (CFRG), Work in Progress, October 2023. Available: `https://datatracker.ietf.org/doc/draft-irtf-cfrg-ristretto255/`

[6] Schnorr, C. P. *Efficient signature generation by smart cards*. Journal of Cryptology, 4(3):161–174, 1991.

[7] National Institute of Standards and Technology (NIST). *FIPS PUB 198-1: The Keyed-Hash Message Authentication Code (HMAC)*. July 2008. Available: `https://csrc.nist.gov/pubs/fips/198-1/final`