Defense-in-Depth with PQC and QKD

Xinhua (Frank) Ling, Ph.D., CISSP

Amazon Web Services
2024-11-06

Quantum Resistant Cryptography

- Post Quantum Cryptography (PQC) – mathematical algorithm based
  - Key-encapsulation: FIPS 203, …
  - Digital signature: FIPS 204, 205, …

- Quantum Key Distribution (QKD) – quantum physics based
  - Only for symmetric key establishment: prepare-and-measure, entanglement-based, …
  - Standards: ETSI GS QKD series, ISO/IEC 23837, ITU-T Y.3800 series, IEEE P1913, …

- Defense-in-depth – deploy multiple layers of diverse security controls with independent failure modes to enhance the security posture

Hybrid Symmetric and Asymmetric Keys

NATIONAL SECURITY AGENCY
CENTRAL SECURITY SERVICE

COMMERCIAL SOLUTIONS for CLASSIFIED (CSfC)

Symmetric Key Management Requirements Annex V2.1

Symmetric Pre-Shared Keys (PSKs) should be used instead of or in addition to asymmetric public/private key pairs to provide quantum resistant cryptographic protection of classified information within CSfC solutions. For CSfC customers who have a requirement to protect long-life[1] classified information, at least one of the two CSfC solution tunnels must use PSKs to provide the required quantum resistant cryptographic protection for that information. Both tunnels should use PSKs when possible to provide quantum resistant protection to the entire CSfC solution, however at least one tunnel must use asymmetric public/private key pairs for mutual authentication per the requirements of the applicable CP and the *CSfC Key Management Requirements Annex*.

ETSI TS 103 744 V1.1.1 (2020-12)

ETSI

TECHNICAL SPECIFICATION

CYBER;
Quantum-safe Hybrid Key Exchanges

Process:

1) Form $secret = psk \,||\, k_1 \,||\, k_2 \,||\, ... \,||\, k_n$.
2) Set $f\_context = f(context, MA, MB)$, where f is a context formatting function.
3) $key\_material = KDF(secret, label, f\_context, length)$.
4) Return *key_material*.

Output:

key_material - derived key material.

NOTE: For a given set of key exchange mechanisms, the lengths of the k_i's are independent of the execution of the protocol, i.e. k_1 will be $length_1$, k_2 will be $length_2$, etc.

A pre-shared key, *psk*, for this method and the cascade method below may be established using a previous session or an alternative key-establishment method like QKD.

GSMA™

IG.18 Opportunities and Challenges for Hybrid (QKD and PQC) Scenarios

Version 1.0

20 October 2024

International Telecommunication Union

ITU-T
TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Technical Report

(24 November 2021)

ITU-T Focus Group on Quantum Information Technology for Networks (FG QIT4N)

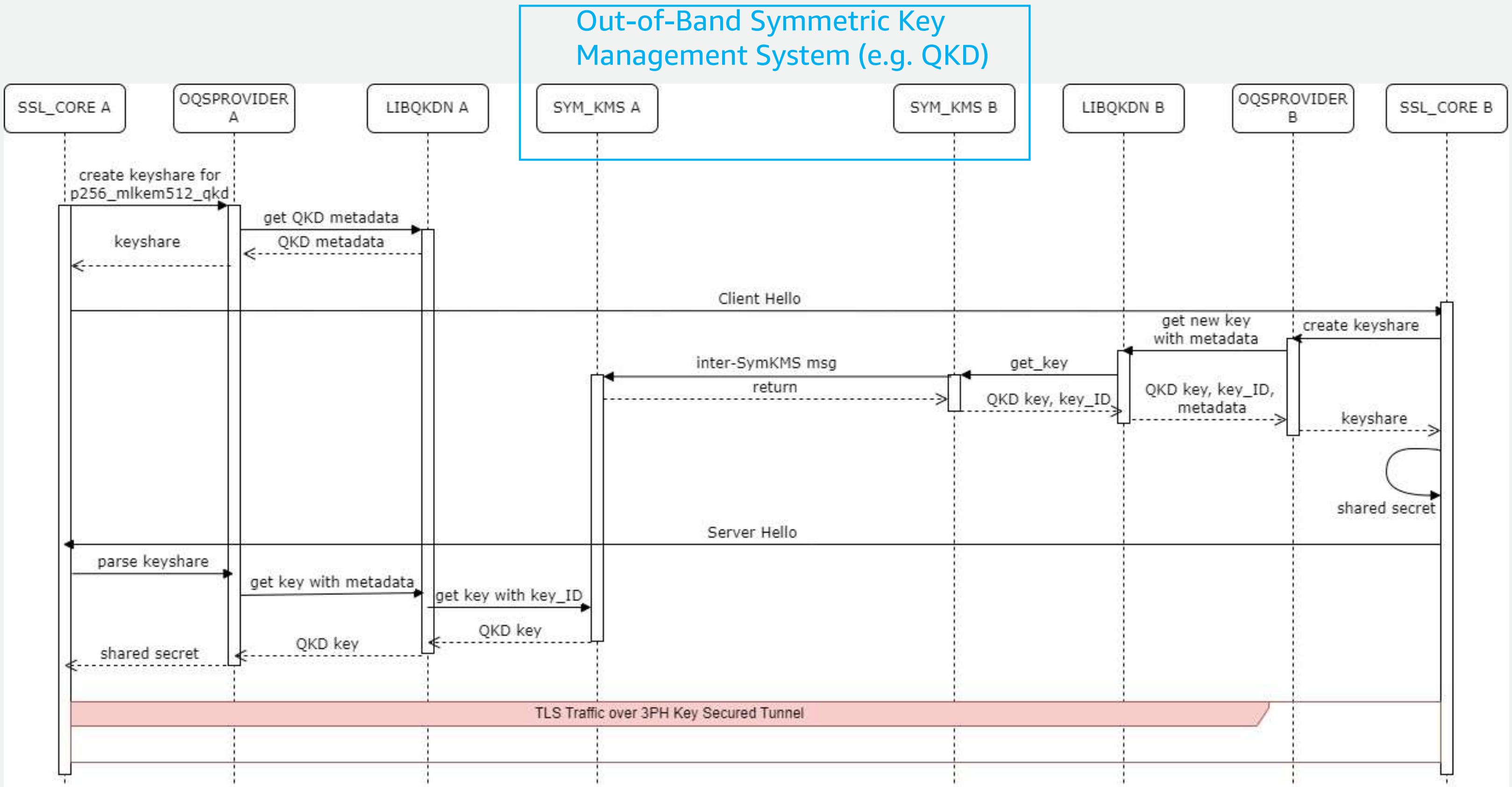
FG QIT4N D2.2
Quantum information technology for networks use cases: Quantum key distribution network

Example: 3-Party-Hybrid Key in OpenSSL v3.x

- Build on top of the 2-party-hybrid (e.g. P256-MLKEM512) in *oqsprovider*
  - Some moderate code changes in *oqsprovider*
  - No code change needed in *OpenSSL core*, nor in *liboqs*
    - Except for adding some supported group names (e.g. P256_MLKEM512_QKD)

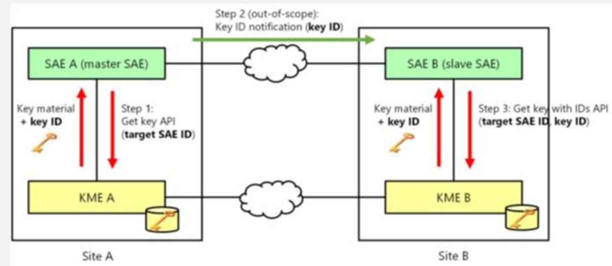
- Hybrid method: concatenate a QKD key as the third part to the hybrid secret
  - Approved “hybrid” technique, Sec. 2, NIST SP 800-56Cr2
  - Approved method 3, Sec. 6.3, NIST SP 800-133r2

3-Party-Hybrid Key in OpenSSL v3.x – Cont'd



Current Work on ETSI QKD APIs

- ETSI GS QKD 014 “Protocol and data format of REST-based key delivery API” v1.1.1 (2019-02)
  - sample implementation in C (i.e. *libqkdn*) by AWS Quantum Network Engineering team, to be open sourced upon Amazon internal approval




```c
#define QKDN_H

int qkdn_get_ctos_string(char** strtoserver);
int qkdn_get_local_config(char* kmename, char* saeid);
int qkdn_new_key(char** newkey, char**stoc_string, char* secondarysae);
int qkdn_get_key(char** key, char* strfromserver);

#endif
```

- ETSI GS QKD 004 “Application Interface” v2.1.1 (2020-08)
  - sample implementation in C by a Universidad Politécnica de Madrid (UPM) team, open source hosted at https://forge.etsi.org/rep/qkd/gs004-app-int

```c
#ifndef ETSI_GS_QKD_004_INTERFACES_H
#define ETSI_GS_QKD_004_INTERFACES_H

#include "qkd/models.h"

#ifdef __cplusplus__
extern "C" {
#endif

extern const version_t QKD_004_VERSION;

/**
 * @brief This is the OPEN_CONNECT call of the ETSI QKD 004.
 *
 * @param source         (in) the source identifier for the stream.
 * @param destination    (in) the destination identifier for the stream.
 * @param qos            (in/out) Quality of Service parameters, an empty qos marks end of list.
 * @param key_stream_id (in/out) key stream id.
 * @param status (out) status code of this call (see qkd_status_codes enum).
 * @param app_context_data   (in/out) service shared data (non-standard parameter). Used to share data between different processes.
 */
void
qkd_open_connect(qkd_uri_t source, qkd_uri_t destination, qkd_qos_t *qos, uuid_t *key_stream_id,
                 qkd_status_t *status, void *app_context_data);


/**
 * @brief This is the GET_KEY call of the ETSI QKD 004.
 *
 * @param key_stream_id (in) the key stream id, must be retrieved by a previous qkd_open.
 * @param index          (in/out) the index within the key stream id buffer.
 * @param key_buffer     (out) the key material buffer to be filled.
 * @param metadata       (in/out) the qkd_metadata for the key.
 * @param status         (out) status code of this call (see qkd_status_codes enum).
 * @param app_context_data   (in/out) service shared data (non-standard parameter). Used to share data between different processes.
 */
void qkd_get_key(uuid_t key_stream_id, uint32_t *index, qkd_key_buffer_t *key_buffer, qkd_metadata_t *metadata,
                 qkd_status_t *status, void *app_context_data);

/**
 * @brief This is the CLOSE call of the ETSI QKD 004.
 *
 * @param key_stream_id (in) the key stream to close.
 * @param status         (out) status code of this call (see qkd_status_codes enum).
 * @param app_context_data   (in/out) service shared data (non-standard parameter). Used to share data between different processes.
 */
void qkd_close(uuid_t key_stream_id, qkd_status_t *status, void *app_context_data);
```

Planned Future Work

- More robust error/exception handling for existing projects
  - What if the QKD key is not available on the first or second peer?
  - What if the QKD keys are different on the two sides?
  - …
- PSK in TLS v1.3 with QKD
- Integrate hybrid keys into other protocols, e.g.,
  - IKEv2 (Internet Key Exchange protocol) – RFC8784
  - MKA (MACsec Key Agreement protocol) – IEEE Std 802.1AE-2018
- Implement in other programming languages (GO …)

The Request

- PQCA to host an “experimental track” project on hybridizing keys from PQC and QKD (and classic algorithms) in a library with built-in defense-in-depth capability

Thank you!

Xinhua (Frank) Ling
lingxinh@amazon.com