Digital Europe Programme Project **QCI-CAT**
QCI: Proof of Concept – Secure Connectivity Austria
Digital Europe Work Programme 2021-2022
EU Secure Quantum Communication Infrastructure (DIGITAL-2021-QCI-01)
Project number: 101091642

Project starting date: fixed date: 1 January 2023
Project end date: 31 December 2025
Project duration: 36 months

Document:	**Deliverable**
Type:	Demonstrator pilot, prototype
Dissemination Level:	Public
Title:	**PQC-hardened Key Management Systems for QKD**
Work-Package	WP8
Document number:	**D8.3**
Document Owner:	Sebastian Ramacher (AIT)
Contributors:	Stephan Laschet (AIT)
Abstract:	This document describes the extensions of AIT's KMS with post-quantum secure key exchange protocols to provide end-to-end security guarantees.
Key words:	Muckle+, post-quantum key exchanges, FAEST
Pages	39
Delivery Date Planned	2025-05-31 (M29)

Revision History

Version	Revision Points	Author(s) & Organization	Date
V 1.0	Initial version	S. Ramacher (AIT)	2025-05-20
V 1.1	Include review comments	S. Ramacher (AIT)	2025-05-31

Author List

Organization	Name	E-Mail address
AIT	S. Ramacher	sebastian.ramacher@ait.ac.at
AIT	S. Laschet	stephan.laschet@ait.ac.at

Reviewer List

Organization	Name	E-Mail address
CANCOM	A. Neuhold	andreas.neuhold@cancom.com

Copyright Statement

Funding Acknowledgement:

This project has received funding from the European Union‘s Digital Europe Work Programme 2021-2022 under Project number: 101091642.

Table of content

List of Figures

List of Tables

Executive Summary

The Key Management System (KMS) of a quantum key distribution (QKD) network is the central component delivering key material from the QKD networks to security applications. For networks with more then two nodes, the KMS is also responsible for forwarding key material from the initiator node to the receiving node. While the forwarded keys are protected with keys from the local QKD nodes on a per link basis, end-to-end security of the keys cannot be guaranteed. Indeed, the intermediate nodes all observe and process the keys passing through. With the integration of additional security guarantees from post-quantum secure key exchanges, end-to-end security can be established. Depending on the type of interface, the combination with post-quantum cryptography can be achieved with different approaches and tools. In this deliverable we describe the post-quantum extensions integrated in the KMS deployed in the QCI-CAT network and research results helping to improve the efficiency of post-quantum secure key extension.

1. Introduction

Since the QCI-CAT network consist of multiple QKD links and peers that are not all directly connected, key forwarding requires the use of trusted nodes. The KMS of AIT integrates techniques well-established in classical cryptography with the use of post-quantum secure building blocks to provide end-to-end secure key material to security applications.

1.1. Purpose and scope of the document

This deliverable presents the extensions of the KMS with post-quantum secure protocols and primitives. In addition, the document also provides possible future extensions that future improve the security of the QKD network.

1.2. Target Audience

The document is targeted at developers of KMS, QKD network architects, and researchers.

1.3. Relation to other project work

The KMS described in this document is deployed in the QCI-CAT network as part of the work in work packages 5 and 7.

1.4. Structure of the document

In Section 2 we present the relevant interfaces of the KMS and the protocol for key forwarding. Section 3 presents the integration of post-quantum cryptography into the KMS. Sections 4, 5, and 6 describe improvements to the digital signatures, extensions of the key forwarding protocol and protocols for hybrid key exchange.

2. Key Management System

The Key Management System (KMS) is a central component of a Quantum Key Distribution Network (QKDN). The QKD modules generate point-to-point (P2P) keys, which are passed to the KMS. Applications request end-to-end (E2E) keys from the KMS between any pair of peers in the QKDN. The KMS' main tasks are:

- Receiving, managing, and providing keys.
- Creating E2E keys from P2P keys by forwarding them in the QKDN.
- Synchronizing databases, e.g., for the provision of different key sizes.
- Maintaining information-theoretic security (ITS) by selecting appropriate cryptographic methods, complemented by post-quantum cryptography (PQC) methods.

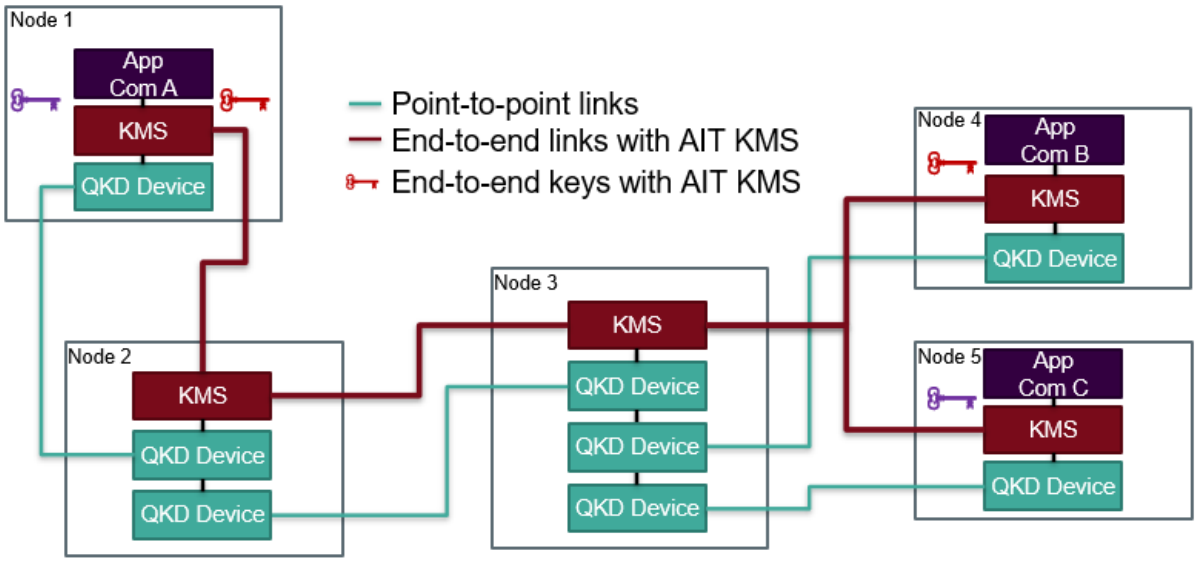

Figure 1: Schematic of a QKD network with KMS instances at every node supplying key material to applications.

The KMS[1] deployed in the QCI-CAT network is developed by AIT as a software solution that is pre-configured and pre-installed on servers at each node in the network. It aims to implement all relevant ETSI interfaces to interact with the other components of the network. These interfaces include ETSI GS QKD 004 [3], ETSI GS QKD 014 [4] and ETSI GS QKD 015 [5].

The design of the KMS is focused on its core functionalities and the required interfaces. Basically, any message received by any external interface is first handed over to a message queue. From there, it is dispatched to a set of components that provide based on the required action as part of the so-called KMS Worker. The KMS Worker itself consists of components for key lifecycle, key forwarding, QKD device management, key synchronization, key management, cryptographic operations, configuration management, database, and logging (c.f. Figure 2). In the following we discuss the design of the external interfaces and their messages.

[1] See https://qkd-kms.ait.ac.at/ for general information on the KMS. An academic discussion of some of the aspects of the KMS can also be found in [2].

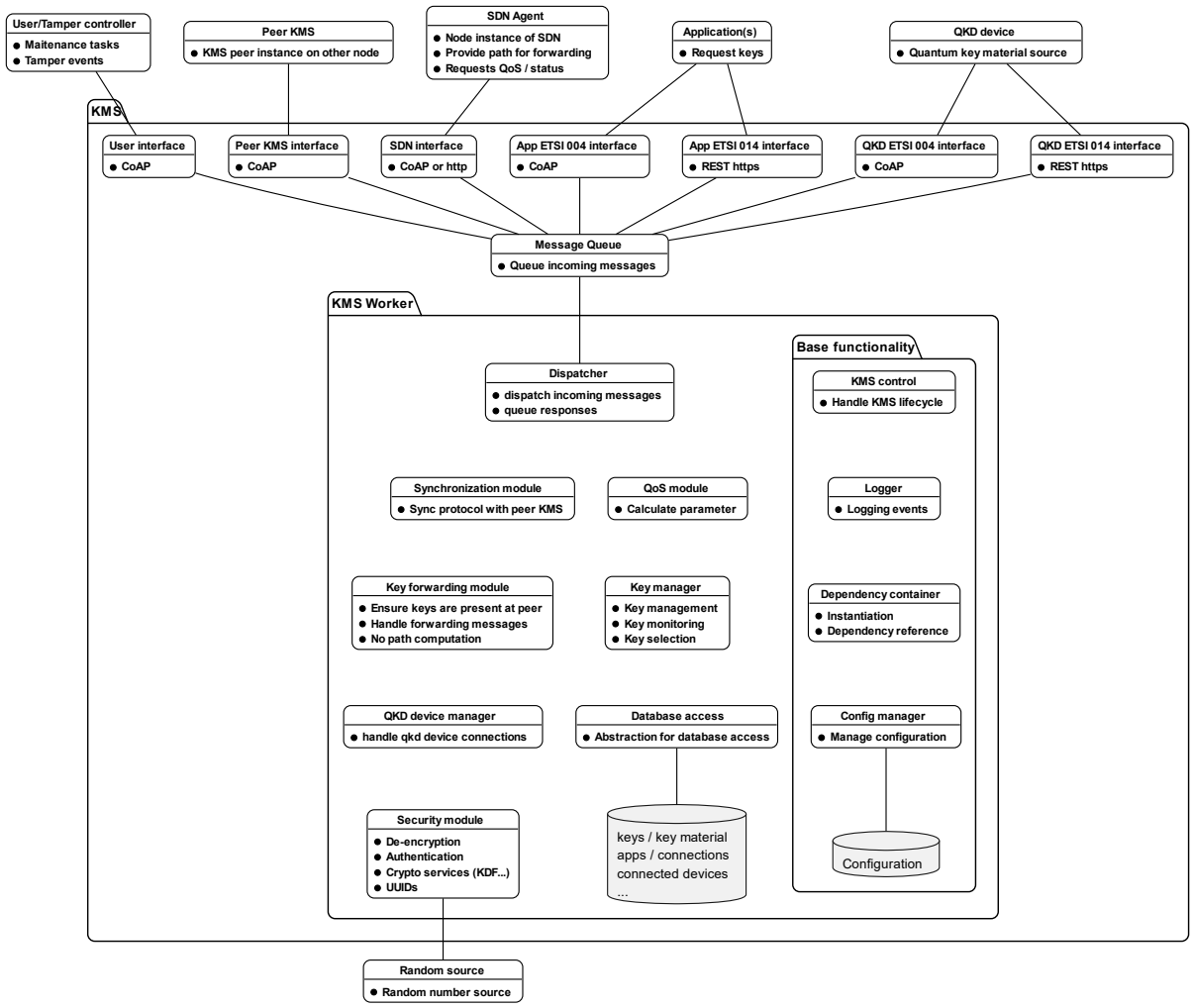

Figure 2: High level KMS design

2.1. Inter KMS Communication: Synchronization between Peer KMS Instances

The following subsections explain the available messages to exchange data between peer KMS instances of the QKD network. The messages in general follow a request/response paradigm with exceptions that do not have a response because they are not useful from a technical point of view.

It is up to the implementation to decide between piggybacked response model or separate request/response messages. Generally piggybacked responses are recommended if it can be assumed, that a response can be determined, given and transmitted within a reasonable time frame. If the response is assumed to be time-consuming or if the response can only be generated later it should be separate messages.

The Message Authentication Code (MAC) is a value generated with an ITS authentication algorithm by the security module. The algorithm takes the key, specified by the KeyID of the Authentication Key and authenticates the other data in the message, excluding the MAC itself.

The messages also state where the message is coming from by providing the UUID of the node, the KMS is running on. This is necessary, because the receiving entity only sees a new message at their listener and does not know where the message came from.

The status codes are individual for each message and to be documented as they are implemented. The general convention is: 0 = No Error. Usually, the error that occurred should be given an individual status code, but the 1 = Unspecified Error is reserved as a default error.

The status codes follow a paradigm, where the functional blocks have a number space, the messages have a subspace. The status code is a 32-bit unsigned integer, where the highest byte is 00, the next byte determines the functional group, the next byte the message and the lowest byte the status code. It therefore ranges from 0000 0000 to 00FF FFFF with the grouping 00FF MMEE with FF = functional group, MM = message, EE = Status code. The first status code (01) in each EE group is reserved for the invalid MAC error.

Some messages require to look up the primary/secondary role, which is noted at the beginning of the message.

2.1.1. Key Receive

This functional group is related to the process of receiving new key material from the QKD devices. This synchronization process ensures, that both KMS received the same key material and have consistent databases.

new_qkd_key_stream(): Requires Primary/Secondary negotiation.

A new ETSI 004 key stream is established (open_connect) by the primary KMS. The KS_ID must be communicated to the peer secondary KMS.

- It communicates the obtained KS_ID to the secondary KMS
- The secondary KMS will open the same key stream (open_connect)
- The secondary KMS will report back on the success.
- Both will immediately start getting keys, if the corresponding flag is set. The secondary after it sent the confirmation, the primary after it received the confirmation.

Table 1: New QKD key stream - Request

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Key_stream_ID	UUID_v4
Device ID	UUID_v4
Autostart key retrieval	bool

Table 2: New QKD key stream - Response

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Status Code	unsigned (32 bits)

start_key_retrieval(): Requires Primary/Secondary negotiation.

Synchronize to start key retrieval process via ETSI 004 get_key method. This is only viable if the previous open_connect was synchronized and if the autostart flag was NOT set.

- It communicates the open KS_ID for which to get the keys.

- The secondary KMS will start the key retrieval process for the key stream (get_key)
- The secondary KMS will report back on the success.
- The primary will start the key retrieval process

Table 3: Start key retrieval - Request

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Key_stream_ID	UUID_v4

Table 4: Start key retrieval - Response

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Status Code	unsigned (32 bits)

new_etsi014_keys(): Requires Primary/Secondary negotiation.

A new batch of ETSI014 keys received by the primary KMS must be fetched from the specified device.

Table 5:New ETSI014 key - Request

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
List of KeyIDs	List of UUID_v4
DeviceID	UUID_v4

Table 6: New ETSI014 key - Response

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Status Code	unsigned (32 bits)

new_key_batch(): Requires Primary/Secondary negotiation.

A new batch of keys that were received must be synchronized.

- It gives the Key IDs provided by the QKD module. This may be just an index or a UUID string.
- It gives new Key IDs for the key material. Each of the lists corresponds by position, meaning the first entry of the Key IDs corresponds with the first entry of the new Key ID.
- To eliminate the risk of inconsistent databases a MAC over the data entries to be synchronized is generated, namely the key material values.

Table 7: New key batch - Request

data explanation	type
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
List Device ID	list of UUID_v4
List of original KeyIDs/index values	list of Key IDs/index values
List of new KeyIDs	list of UUID_v4
KeyID of the Auth. Key for Keys	UUID_v4
MAC over Keys	string (base64)

Table 8: New key batch - Response

data explanation	type
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Status Code	unsigned (32 bits)
(if errors) invalid key IDs	list of UUID_v4

2.1.2. Key Provisioning

When keys are provided to the applications, the KMS instances must exchange some information.

new_etsi004_app(): Requires Primary/Secondary negotiation.

When a new etsi004 app registers to the node, some data are set such as the key stream ID, the QOS data and the connection information.

Table 9: New ETSI004 App - Request

data explanation	type
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
UUID of Application	UUID_v4
Source Address	string (URI)
Destination Address	string (URI)
Quality of Service (QoS)	QoS structure
Key_stream_ID	UUID_v4

Table 10: New ETSI004 App - Response

data explanation	type
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Status Code	unsigned (32 bits)

new_etsi014_app(): When an etsi014 master app sends an enc_keys request for a certain slave application for the first time, the KMS sets up internally an Etsi 014 app entry in the database (in order

to properly reserve keys for it). The new app data needs to be synchronized between the interested KMSes.

Table 11:New ETSI014 App - Request

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
UUID of Application	UUID_v4
Master SAE ID	UUID_v4
Slave SAE ID	UUID_v4
Key_stream_ID (optional)	UUID_v4

Table 12: New ETSI014 App - Response

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Status Code	unsigned (32 bits)

close_key_stream(): Requires Primary/Secondary negotiation.

When an app on one node closes a key stream, by design the application of the other node must close it too, after a certain TTL. It could be, that one App closes the connection, but due to some error the other side is not informed of that. Therefore, the KMS should also close the Key Stream ID after the TTL, so it does not continue to unnecessarily provide keys on one side even though the other side does not exist anymore.

Table 13: Close key stream - Request

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Key_stream_ID	UUID_v4

Table 14: Close key stream - Response

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Status Code	unsigned (32 bits)

2.1.3. Maintenance

Some Maintenance tasks require also a synchronization of the peer KMS.

get_status(): Get the status of the peer KMS. No response is expected, since the peer might be disconnected. If a response is received, it means the other node is in principle reachable and online, but the status code might indicate several other states such as "under maintenance".

Table 15:Get status - Request

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4

Table 16: Get status - Response

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Status Code	unsigned (32 bits)
Additional status info	data structure

post_status(): This message is sent if the sending KMS wants to report to the peer a status change. This message is for example sent when the KMS is shutting down to mark itself as disconnect. There is no need for a response, since if the KMS is disconnect afterwards for example the response is irrelevant.

Table 17: Post status - Request

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Status Code	unsigned (32 bits)
Additional status info	data structure

delete_keys(): Requires Primary/Secondary negotiation

The KMS may delete keys established with a peer. This could be because it notices that they are beyond a certain expiration date or the KMS detected a tamper event.

Table 18: Delet keys - Request

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
List of Key_IDs	list of UUID_v4

Table 19: Delete keys - Response

data explanation	**type**

Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Status Code	unsigned (32 bits)
(if errors) invalid key IDs	list of UUID_v4

2.1.4. Forwarding

The forwarding interface exists to forward key material that is not intended for the receiving node but to be forwarded. Errors during forwarding must also be communicated to the SDN.

forward_keys(): Does NOT require Synchronization, because by design this can only start at one node.

The key forwarding procedure is done when an application wants to generate a key with a peer, that is not directly connected, but reachable through intermediate nodes. Then a key available with the intermediate node is used to encrypt the end-to-end key and is sent with this method. It is foreseen, that not only a single key is forwarded but a batch of keys.

- The List of Key_IDs gives the IDs of the Keys used for encryption
- The List of encrypted keys corresponds by position and gives the encrypted values.
- ⚠ The Request is sent without an immediate response from the peer KMS. The Response is sent from the remote destination KMS.

Table 20: Forward keys - Request

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Key origin ID	UUID_v4
List of Key_IDs	list of UUID_v4
List of decryption Key_IDs	list of UUID_v4
List of encrypted keys	list of strings (base64)
Destination KMS ID	UUID_v4
Previous hop	URI
KS_ID	UUID_v4
Key application	string (external, internal)
App is Etsi004 (optional)	bool
Rest KeyID (optional)	UUID_v4

Table 21: Forward keys - Response

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Status Code	unsigned (32 bits)
(if errors) invalid key IDs	list of UUID_v4

2.1.5. Key Reformatting

Available key material may be reshaped to fit certain algorithms or key sizes requested by applications. Keys can be merged into larger ones and split into smaller ones.

reformat_keys(): The initiator of the message describes in the request which keys to use and how to reformat them. The initiator also specifies IDs for the new keys, their application and potential excess keys.

The logic is as follows:

- The initiator specifies the length and IDs for the new keys.
- It also specifies a list of key IDs that shall be used to generate those new keys. In this list the order matters and each original key is appended to the previous one, until a key stream is formed, that is equal or longer than the amount of new key material required.
- Then the new keys are formed by extracting blocks of keys from the concatenated key stream.
- Any leftover bytes are then assigned to the key ID given.

Table 22: Reformat keys - Request

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
List of original Key_IDs	List of UUID_v4
List of new Key_IDs	List of UUID_v4
New key length	unsigned (32 bits)
Key application	string (internal/external)
AppID (optional)	UUID_v4
App is Etsi004 (optional)	bool
Rest KeyID (optional)	UUID_v4

Table 23: Reformat keys - Response

data explanation	**type**
Message Authentication Code (MAC)	string (base64)
KeyID of the Authentication Key	UUID_v4
UUID of this node	UUID_v4
Status Code	unsigned (32 bits)

Notes on data format:

- Soft delete the original keys by marking them as "resized", keeping the key length but delete the actual key bytes. (use the normal deletion method also applied if keys are used)
- Always use the **oldest** creation time of the original keys for the new keys
- Always use the **newest** creation time of the original keys for the rest ("rest_key_id"). The logic is, that since the keys are fetched by age and only as many are fetched to have enough key material, the last original key is the newest and only that one will contribute to the rest.
- Take any of the Device IDs contributing to a key as the origin of the key.

- Mark the source type as the one with the weakest guarantees:
    - if it is quantum and PSK, mark it quantum
    - if it is PSK and ITS_forwarded, mark it ITS_forwarded
    - It cannot be quantum (p2p) and ITS_forwarded (remote)

Notes on error handling:

There is a corner case, where the receiver fails to find the keys specified in the message. The entries could be missing or corrupt. Though this technically cannot really happen, because the keys were synchronized, and the synchronization process is designed to prevent database inconsistencies. But if that happens, the case is handled this way:

- At the receiver if original keys could not be properly fetched from the database, delete all original keys specified by the sender.
- Do not generate new keys.
- Return an error code to the sender.
- At the sender side, delete all new key entries.

The following Examples show different use-cases to explain the process:

2.1.6. Synchronization Strategy of New QKD Keys

The peer KMSs have to synchronize their keys, but for synchronizing the keys, they have to consume keys to generate the MAC. To reduce the internal key consumption, a batch of keys is synchronized and not each one individually. This will add a delay to the availability of keys.

The optimal batching size is to be determined during corresponding performance tests and to be configurable in the configuration. The trade-off is basically the following: The bigger the key batch, the fewer keys are consumed internally, but the bigger the key availability delay. The smaller the key batch, the bigger the internal key consumption and the smaller the key availability delay.

From a first engineering look, it is better to lean into the direction of bigger batches, because due to the buffering nature of the KMS the delay can be mitigated, but internally consumed keys reduce the effective key rate, which is a more critical factor.

Another trade-off is, that if one key in the batch is wrong (not available, not correct etc.) the whole batch is invalid. That means, that the bigger the batch the worse the outcome if a synchronization fails.

To mitigate the possibility, that synchronization fails just because there is a delay in receiving the key on the peer KMS, the following logic is implemented: The synchronization of one batch of keys is only initiated by the primary KMS, if a certain amount of the next batch is already available. This parameter is configurable, since the optimal number is to be determined with tests. The trade-off is here again an increased delay in key availability, which is mitigated by the buffering nature of the KMS.

If the secondary KMS notices a missing Key ID in the batch, it reports this back alongside a failure in synchronization, because it cannot calculate and verify the MAC over the key material of the existing keys. Then the primary can initiate the synchronization for all keys of the batch except the missing ones (resulting in a smaller batch). The missing Key ID is synchronized in the next batch, but if it was

not successful to synchronize a single key for a specified amount of tries, it is considered lost and deleted.

For the case that a batch has all keys available, but the MAC is wrong, indicating that one or more keys have different values, the KMS has to search for the wrong key. A simplified binary search algorithm is implemented, since at some point it simply is not worth it to continue searching (because each synchronization attempt consumes keys).

The batch is divided into 2 batches and a MAC is generated over both parts. If a MAC is valid, all keys corresponding to that MAC are synchronized. For the other batch, if it generates an invalid MAC (which it logically should do) the procedure is repeated. This split is repeated until the batch size is smaller than a lower threshold. The sequence is also not repeated if both MACS are wrong, indicating that there is more than one faulty key available, which would result in a too costly search. All remaining keys contributing to a wrong MAC are deleted.

Secondary unsynchronized key stacking occurs if for some reason the primary KMS drops keys, whereas the secondary KMS never drops keys. In this case, since the secondary KMS never initiates keys, they will never be synchronized. To some extent due to delays a small stack of unsynchronized keys is normal, but if this exceeds a certain threshold, something is wrong and a warning must be communicated to the log and an outside component in the network, since maintenance may be necessary.

From a technical point of view, unsynchronized keys are not a problem and useless, since we need a key pair at both endpoints. Due to the key lifecycle management, they will be deleted eventually.

2.2. ETSI 004

The Interface follows the ETSI 004 API only supporting the use-case, where the KS_ID is not known to applications a-priori, which is outlined in the following sequence:

open_connect: The open_connect message connects an application to the KMS. Conforming to the standard, the KMS will assign the key_stream_id (by obtaining it from the SDN) and can suggest alternative QoS due to network needs (as obtained by the SDN).

KMS resource accessed by an APP with the method POST.

Table 24: Open connect - Request

data explanation	**type**
source	string (IPv4:port)
destination	string (IPv4:port)
key_stream_id (opt)	string (UUID_v4)
qos	JSON object
qos.key_chunk_size	uint (Byte)
qos.max_bps	uint (bit)
qos.min_bps	uint (bit)
qos.jitter	uint
qos.priority	uint
qos.timeout	uint
qos.ttl	uint
qos.metadata_mimetype	string (mimetype)

Table 25: Open connect - Response

data explanation	type
source	string (URI)
destination	string (URI)
key_stream_id	string (UUID_v4)
qos	JSON object
qos.key_chunk_size	uint (Byte)
qos.max_bps	uint (bit)
qos.min_bps	uint (bit)
qos.jitter	uint
qos.priority	uint
qos.timeout	uint
qos.ttl	uint
qos.metadata_mimetype	string (mimetype)
status	uint

get_key: The get_key message retrieves a key from the KMS.

KMS resource accessed by an APP with the method GET in CoAP, while with POST in HTTP.

Table 26: Get key - Request

data explanation	type
key_stream_id	string (UUID_v4)
index	uint
metadata	JSON object

Table 27: Get key - Response

data explanation	type
index	uint
status	uint
key_buffer	string (base64)
metadata	JSON object

close: The close message disconnects an application from the KMS.

KMS resource accessed by an APP with the method PUT.

Table 28: Close - Request

data explanation	type
key_stream_id	string (UUID_v4)

Table 29: Close - Response

data explanation	type
status	uint

Status codes: The status value is according to the ETSI 004 standard. Additional custom status codes were added to communicate certain issues more specifically.

Table 30: Status codes

status	code
0	"Successful"
1	"Successful connection, but peer not connected"
2	"GET_KEY failed because insufficient key available"
3	"GET_KEY failed because peer application is not yet connected"
4	"No QKD connection available"
5	"OPEN_CONNECT failed because the KSID is already in use"
6	"TIMEOUT_ERROR The call failed because the specified TIMEOUT"
7	"OPEN failed because requested QoS settings could not be met, counter proposal included in return has occurred"
8	"GET_KEY failed because metadata field size insufficient. Returned Metadata_size value holds minimum needed size of metadata"
62	"success_already_open"
63	"success_already_open_remote_not_connected"
99	"not set"
2	"GET_KEY failed because insufficient key available"

Transport layer return codes: The following positive acknowledgment return codes are used.

Table 31: Positive acknowledgment return codes

message	response code OK CoAP	response code OK http
open_connect	2.01 Created	201 Created
get_key	2.05 Content	200 OK
Close	2.04 Changed	200 OK
key_material	2.01 Created	201 Created

The following negative acknowledgment return codes are used:

Table 32: Negative acknowledgment return codes

message	reason	error code CoAP	error Code http	message
open_connect	The key stream could not be opened (key stream ID already in use, etc.)	4.03 Forbidden	403 Forbidden	open_connect
	QoS could not be met	4.09 Conflict	409 Conflict	
get_key	The specified key stream ID is not registered	4.04 Not Found	404 Not Found	get_key
	Insufficient metadata size	4.00 Bad Request	400 Bad Request	
close	The specified key stream ID is not registered	4.04 Not Found	404 Not Found	close

The following negative acknowledgment return codes are also used for each etsi004 response:

Table 33: Error codes for ETSI 004 responses

error	error code CoAP	error Code http
User sent a malformed input	4.00 Bad Request	400 Bad Request
The KMS experienced an internal error	5.00 Internal Server Error	500 Internal Server Error
The message was re-transmitted (only CoAP)	4.29 Too Many Requests	--

Transport protocol: Three options for a transport protocol are available. HTTPs and HTTPS are mutual exclusive, since they are served by the same server instance, so either this server is configured to use HTTP or HTTPS.

- **CoAP:** This light-weight transport protocol allows for certain flexibility and an alternative for networks, where only UDP is supported. The used transport protocol is CoAP over UDP as specified by RFC 7252.[2]
- **HTTP:** Plain HTTP is supported as transport protocol by the KMS.
- **HTTPS:** The alternatively to http, is HTTPS is supported using TLS 1.2 with pre-shared certificate files.

2.3. SDN API

The SDN API is provided to satisfy the requirements of ETSI 015 and is used by the SDN agent to communicate with the KMS.

SKN and ESKN in the Link Performance Response: In response to a link performance request message, the key numbers, rather than the rates, are returned to the SDN Agent.

Table 34: Response

data explanation	type
link_id	string (UUID_v4)
secret_key_bytes	uInt32 (bytes)
effective_secret_key_bytes	uInt32 (bytes)
error_code	uInt32

where:

secret_key_bytes is the number of bytes of keys received from a QKD during a given interval (skn in the qos_module)

effective_secret_key_bytes is the number of bytes of keys in the KeyDB where life_cycle=synchronized, key_type=quantum & key_application=unassigned, i.e. the amount of key material available for an application.

[2] https://datatracker.ietf.org/doc/html/rfc7252

3. Post-Quantum Crypto Integration

There are multiple layers that can be affected by the integration of methods and schemes that are post-quantum secure. Those layers are inherently affected by their handling of some form of key material or communication

- KMS to KMS communication: This communication includes the forwarding of keys between peers.
- ETSI 014 interface: This interface presents the sole access point of security applications to interact with the KMS. As such, all requested keys are handed to the security application via this interface.
- ETSI 015 interface: The KMS is controlled by the Software-Defined Network (SDN) layer via this interface.
- Shared key: While the key shared between the two peers is ITS secure, it fails to meet end-to-end security criteria due to trusted nodes.

Note that the communication between the KMS and its local QKD node is all done within the same node and local network. While one could ask whether this connection also needs to be secured, we consider a man-in-the-middle between the KMS and the local QKD node as a complete compromise of the QKD itself. Nevertheless, if the local communication is implemented using ETSI 014, then the same applies as to the ETSI 014 communication between security application and KMS. In fact, AIT's KMS with post-quantum secure ETSI 014 could act as reverse proxy on the QKD device to establish post-quantum secure channel between KMS and QKD device.

3.1. KMS-to-KMS communication

As discussed in Section 2.1, KMS-to-KMS communication is authenticated via an ITS-secure MAC. As such, the communication is inherently post-quantum secure at the cost of consuming QKD keys. We discuss in Section 6 an approach, that replaces the ITS MAC with unconditional hiding commitments.

3.2. ETSI 014 Interface

Recall that the ETSI 014 interface is a REST interface that is provided via HTTPS. As HTTPS – as currently standardized – is relying on classical cryptographic primitives (RSA, EdDSA, ECDHE, etc.) to ensure an authentic end-to-end secure key exchange, these primitives need to be replaced with post-quantum secure variants. There is ongoing work in different dimensions that will render HTTPS post-quantum secure.

From a standardization perspective, there are ongoing efforts at the IETF[3] which is responsible for the RFC for TLS, to standardize hybrid versions of the key exchange. The main focus of these efforts is the combination of the classical ECDHE-based key exchange with a post-quantum secure KEM, i.e., ML-KEM. Thereby the key exchange provides protection against store-new-decrypt-later attacks with possible future quantum computers. Notably, authentication is still performed with classical signatures. Adopting PQ signatures (or alternative PQ authentication methods such as KEMTLS) is currently deemed less important.[4]

From an implementation and deployment perspective, big software companies and projects implemented the proposed drafts and deployed them in production. These include Firefox 132, Chrome 131 , Edge 131 and other browsers, OpenSSL 3.5.0, GnuTLS 3.8.9, and others.

[3] https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/

[4] https://datatracker.ietf.org/doc/draft-reddy-uta-pqc-app/

The KMS from AIT relies on botan[5] for the HTTPS implementation. This library supports HTTPS with hybrid and standalone key exchanges with ML-KEM since version 3.7.0.[6] The KMS currently uses version 3.7.1, so it automatically gains support for ML-KEM. Note however, that botan currently does not support hybrid certificates. Hence, the KMS implements authorization via liboqs.[7]

3.3. ETSI 015 Interface

Similar to the security of the communication for ETSI 014, the security of the underlying transport protocol is also inherited. If NETCONF is used for network management, then protocols such as HTTPS or SSH are used as transport layer. So post-quantum security of these protocols directly translates to post-quantum security of the implementation of the ETSI 015 interface. The situation is different however if RESTCONF is used. In this case, the transport protocol is unsecured HTTP. Instead of introducing additional security measures on top of RESTCONF, we suggest implementers of the SDN components to switch to NETCONF-based implementations or provide post-quantum secured tunnels for RESTCONF messages.

3.4. Shared key

If we consider the shared key between two security applications, the key can be enhanced at multiple stages of its lifetime with post-quantum based key material. In parallel to the key forwarding of the QKD key from the initiator to the receiver KMS, the two KMS instances can also perform post-quantum based key exchange in parallel. After performing both steps, these two keys could be combined with a key combiner and then provided to the user. The resulting key inherits the ITS properties of the QKD keys whereas it also inherits the end-to-end security guarantees of the PQ key exchange. Thereby, as a side effect of the introducing the post-quantum security component, the resulting key also is protected against compromise of any of the trusted nodes.

Instead of such an ad-hoc approach, an alternative is the implementation of the Muckle+ protocol to produce end-to-end authenticated quantum-resistant keys that combines QKD, PQ and classical keys in a secure manner. We discuss the Muckle+ protocol in depth in Section 4.

4. Muckle+: Hybrid Authenticated-Key Exchange[8]

The Muckle+ protocol is a hybrid authenticated key exchange (HAKE) protocol designed to provide robust end-to-end security in the face of evolving cryptographic threats, particularly those posed by quantum computing. Building upon the foundational Muckle protocol [6], Muckle+ integrates post-quantum digital signatures to enhance authentication mechanisms, thereby eliminating the reliance on pre-shared keys and improving scalability and flexibility in diverse network environments.

[5] https://botan.randombit.net

[6] https://botan.randombit.net/news.html#version-3-7-0-2025-02-04

[7] https://openquantumsafe.org/liboqs/

[8] This section is based on [1]

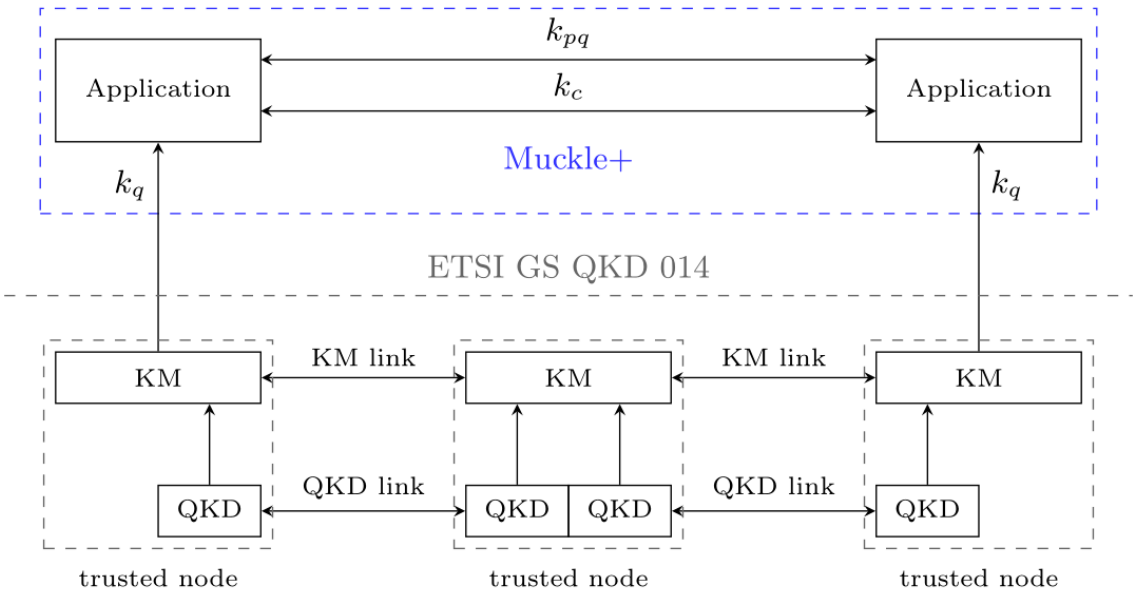

Figure 3: Architecture of a Muckle+ implementation wit a single intermediate node.

4.1. Hybrid Authenticated Key Exchange (HAKE)

Authenticated Key Exchange (AKE) protocols are fundamental to establishing secure communication channels, ensuring both end-to-end confidentiality and end-to-end authenticity. Hybrid Authenticated Key Exchange (HAKE) protocols combine multiple cryptographic primitives—classical, post-quantum, and quantum key distribution (QKD)—to enhance security and resilience.

The original Muckle protocol, introduced by Dowling, Brandt Hansen, and Paterson, utilized pre-shared keys within a modular HAKE framework. While effective, this approach was constrained by the inefficiencies associated with pre-shared key management, particularly in large-scale or dynamic networks. Muckle+ advances this framework by incorporating post-quantum digital signatures, thereby facilitating more efficient and scalable authentication without compromising security.

4.2. Core Components of Muckle+

Post-Quantum Digital Signatures. Muckle+ replaces pre-shared keys with post-quantum digital signatures for authentication. This shift enhances scalability and eliminates the need for prior key distribution, which is particularly beneficial in large or dynamic networks. However, integrating digital signatures into the HAKE framework required significant modifications to the underlying security models and proof techniques to ensure robustness against quantum adversaries.

Modular Integration of Cryptographic Primitives. The protocol is designed to flexibly incorporate various cryptographic components:

- **Classical Key Encapsulation Mechanisms (KEMs):** Established cryptographic methods providing efficient key exchange.
- **Post-Quantum KEMs:** Algorithms resistant to quantum attacks, ensuring long-term security.
- **Quantum Key Distribution (QKD):** Utilizes quantum mechanics principles to establish secure keys, offering information-theoretic security.

By combining these elements, Muckle+ achieves redundancy; the compromise of one component does not necessarily compromise the overall security of the communication channel.

Key Derivation and Session Key Establishment. Muckle+ employs a structured key derivation process that is designed to compatible with the use in TLS. Its goal is to integrate QKD keys in the TLS handshake for seamless integration in existing TLS setups. Hence, the key exchange itself is done in the classical way (with ECDHE) plus a post-quantum KEM. On top of that, also a QKD key is requested via the client's and server's local KMS instances.[9] All three keys are then combined via TLS' Key Derivation Function (KDF) to produce the final session key. The messages sent to perform the handshake are (mutually) authenticated via post-quantum secure digital signatures.

This process ensures that the session key is secure, even if one of the key exchange mechanisms is compromised. Indeed, if any of the exchanged keys is compromised, the security of the other two keys still ensures the overall security of the protocol.

4.3. Security Considerations

Muckle+ is designed with several security objectives:

- **Forward Secrecy:** Ensures that the compromise of long-term keys does not compromise past session keys.
- **Post-Quantum Security:** Incorporates cryptographic primitives resistant to quantum attacks, safeguarding against future threats.
- **Redundancy:** Combining multiple key exchange methods ensures that the failure of one does not lead to a total security breakdown.

The protocol's security proofs involve complex models and adapted proof techniques to account for the integration of post-quantum digital signatures within the HAKE framework.

5. FAEST: Improved Post-Quantum Secure Signatures

Digital signatures can be constructed one-way functions (OWF) which itself can be built from symmetric primitives such as block ciphers. As such, signature schemes are directly constructable from them too. Initial work to show the practicality of such constructions was Picnic [7] which was submitted to the NIST standardization effort on post-quantum signatures. The main idea of Picnic consists of proving the pre-image of LowMC-based OWF in zero-knowledge. By applying the Fiat-Shamir transform to such an interactive proof, one can obtain a signature. Picnic provides reasonable runtime performance but the signatures themselves are large in comparison to other symmetric signatures such as SPHINCS+ [8]. Furthermore, the NIST evaluation highlighted the use of a relatively new block cipher such as LowMC instead of well-studied ones such as AES.

FAEST[10] follows a similar approach but focuses on OWFs instantiated from AES. Additionally, it switches the proof paradigm of MPC-in-the-head with VOLE-in-the-head. The latter provides more efficient proofs both in terms of runtime costs as well as signature size. Hence, it addresses multiple concerns raised during the first iteration of the NIST PQC project. FAEST was submitted to the additional call for signatures in June 2023. In the following, we discuss two improvements that were added to further improve FAEST during the NIST process.

[9] The key ID obtained from the KMS on the client side can be communicated to the server via an optional extension of the TLS handshake.

[10] https://feast.info

5.1. One Tree to Rule Them All: Optimizing GGM Trees and OWFs for Post-Quantum Signatures[11]

Abstract. *The use of MPC-in-the-Head (MPCitH)-based zero-knowledge proofs of knowledge (ZKPoK) to prove knowledge of a preimage of a one-way function (OWF) is a popular approach towards constructing efficient post-quantum digital signatures. Starting with the Picnic signature scheme, many optimized MPCitH signatures using a variety of (candidate) OWFs have been proposed. Recently, Baum et al. (CRYPTO 2023) showed a fundamental improvement to MPCitH, called VOLE-in-the-Head (VOLEitH), which can generically reduce the signature size by at least a factor of two without decreasing computational performance or introducing new assumptions. Based on this, they designed the FAEST signature which uses AES as the underlying OWF. However, in comparison to MPCitH, the behavior of VOLEitH when using other OWFs is still unexplored.*

In this work, we improve a crucial building block of the VOLEitH and MPCitH approaches, the so-called all-but-one vector commitment, thus decreasing the signature size of VOLEitH and MPCitH signature schemes. Moreover, by introducing a small Proof of Work into the signing procedure, we can improve the parameters of VOLEitH (further decreasing signature size) without compromising the computational performance of the scheme. Based on these optimizations, we propose three VOLEitH signature schemes FAESTER, KuMQuat, and MandaRain based on AES, MQ, and Rain, respectively. We carefully explore the parameter space for these schemes and implement each, showcasing their performance with benchmarks. Our experiments show that these three signature schemes outperform MPCitH-based competitors that use comparable OWFs, in terms of both signature size and signing/verification time.

The emergence of quantum computing poses a significant threat to classical cryptographic systems, particularly digital signatures. In response, the cryptographic community has been developing post-quantum (PQ) digital signature schemes based on assumptions believed to be resistant to quantum attacks. The U.S. National Institute of Standards and Technology (NIST) initiated a standardization process in 2017, resulting in the selection of three schemes: SPHINCS+[8], Dilithium [10], and FALCON [11]. While Dilithium and FALCON rely on lattice-based hardness assumptions, NIST recognized the need for diversity in cryptographic foundations, prompting further exploration of alternative approaches.

One promising direction involves constructing digital signatures from zero-knowledge proofs of knowledge (ZKPoKs). These are interactive protocols that allow a prover to demonstrate knowledge of a secret (e.g., a private key) without revealing it. When combined with the Fiat-Shamir transformation, these interactive proofs can be converted into non-interactive digital signatures.

A particularly effective method for building such proofs is the MPC-in-the-Head (MPCitH) [12] paradigm, which simulates a secure multi-party computation protocol inside the prover's mind. While MPCitH is efficient for small to medium-sized circuits, its proof size scales linearly with the circuit size, which can be a limitation for larger applications.

VOLE-ZK and the FAEST Scheme. To address these limitations, recent works developed VOLE-ZK proofs [13], which use vector oblivious linear evaluation (VOLE) to achieve linear complexity and scalability. These proofs initially supported only designated verifiers, but recent work introduced VOLE-in-the-Head (VOLEitH) [14], enabling public verifiability while retaining efficiency.

[11] This section is based on [9].

The FAEST signature scheme is based on VOLEitH and uses AES as its one-way function (OWF). Compared to MPCitH-based schemes (e.g, [15], [16], [17], [18]), FAEST achieves smaller signatures (at least 2× smaller) and comparable or better performance in signing and verification, making it a strong candidate for post-quantum cryptography.

Optimizations and New Techniques: In this work ,we introduce new optimizations for FAEST:

1. **Batch All-But-One Vector Commitments (BAVC):** To reduce the overhead of generating and opening multiple vector commitments in VOLEitH, we introduce BAVC, a new abstraction that allows interleaving commitments. This significantly reduces the opening size and improves efficiency. Although this approach introduces rejection sampling, it does not weaken security—instead, it acts as a proof of work, increasing the cost for potential attackers.
2. **FAESTER – An Optimized Variant of FAEST:** Building on BAVC, we propose FAESTER, which incorporates a technique called grinding [19]. Grinding reduces the entropy of the challenge space, allowing for further reductions in signature size and computational cost. Despite a slight reduction in the theoretical security of the proof, the overall scheme maintains its security level due to the added proof-of-work burden. FAESTER achieves a signature size of 4KB, the smallest among AES-based schemes, while maintaining or improving performance.
3. **FAEST-d7:** We also present a new method of proving AES in VOLE-ZK proof systems, using degree-7 constraints over the binary field. Compared with the degree-2 constraints over the 8-degree binary extension field used in the original FAEST, we halve the witness size in the ZK proof. Although proving higher-degree constraints does come with some extra costs, we show that signature sizes can be up to 5% smaller in FAEST-d7.

Exploring Alternative One-Way Functions. While AES is a well-established and conservative choice, it was not designed for VOLEitH-style applications. Hence, we also explore the design space of alternative OWFs, including Rain [18] and multivariate quadratic (MQ) [20] PRFs, which may offer better performance or smaller signatures. Thereby, we introduce two new schemes: MandaRain (based on Rain PRF) and KuMQuat (based on MQ PRF). These schemes leverage the same commitment optimizations and achieve signature sizes as small as 2.6KB, the smallest among all VOLEitH and MPCitH-based schemes. For a comparison see Figure 4 and Figure 5.

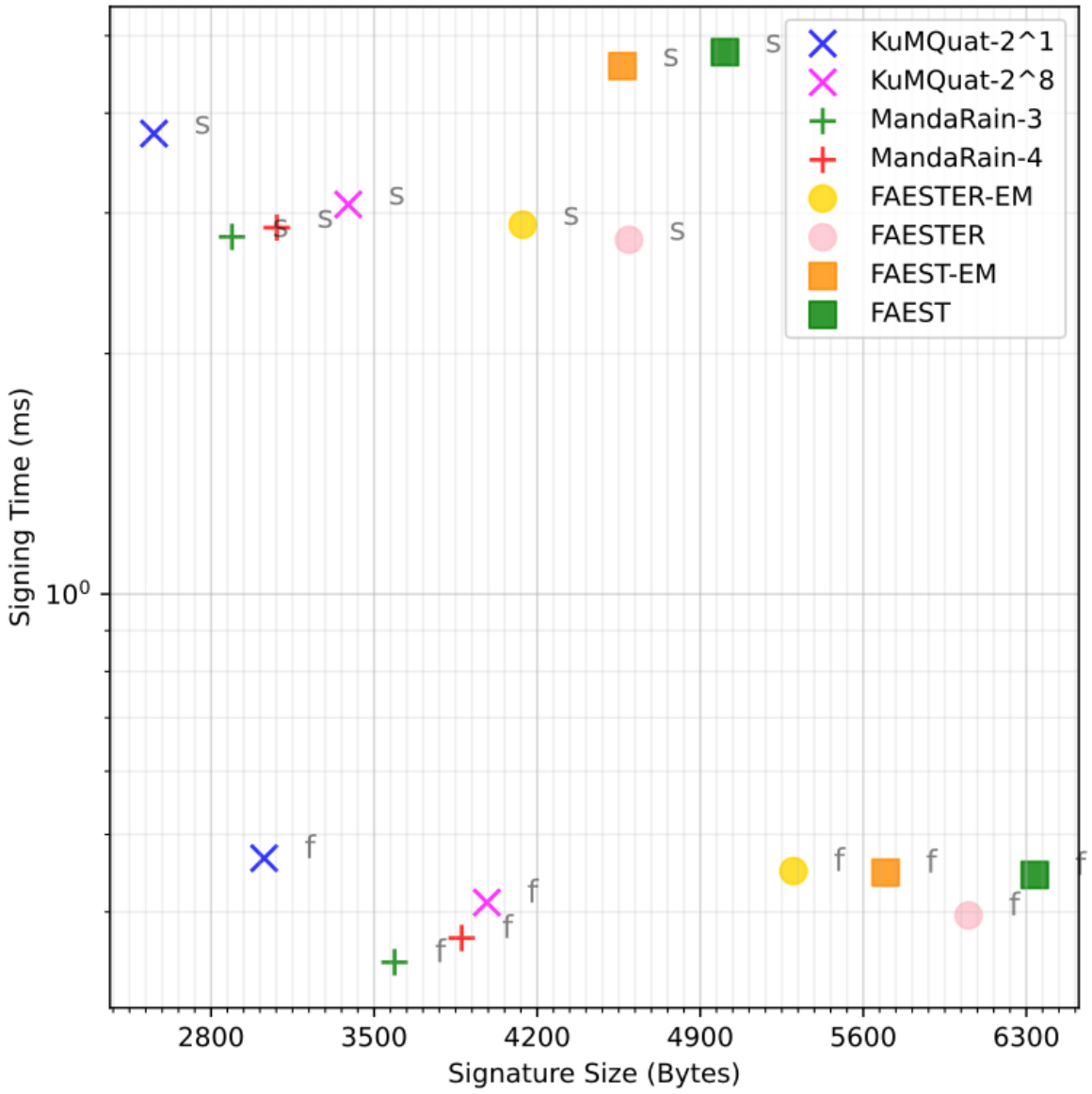

Figure 4: Signature size and runtime trade-off comparison between the proposed signature schemes with FAEST and FAEST-EM. The slow and fast versions are denoted with sand f respectively. The fast version offers smaller signing and verification time, however, comes with a larger signature size. Similarly, in the slower version, the signature size is smaller but both signing and verification timings are larger.

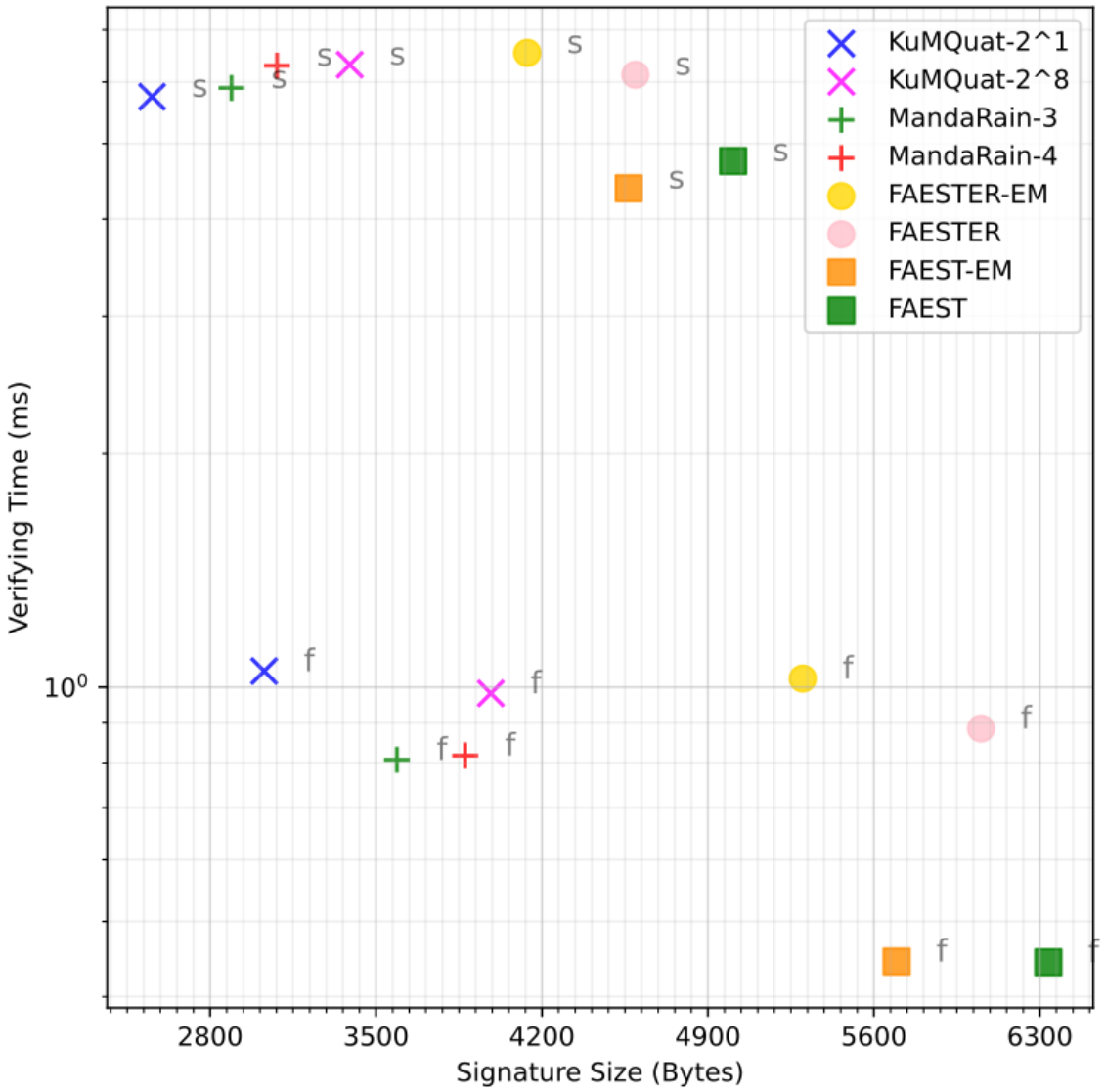

Figure 5: Signature size and runtime trade-off comparison between the proposed signature schemes with FAEST and FAEST-EM. The slow and fast versions are denoted with sand f respectively. The fast version offers smaller signing and verification time, however, comes with a larger signature size. Similarly, in the slower version, the signature size is smaller but both signing and verification timings are larger.

5.2. Shorter, Tighter, FAESTer: Optimizations and Improved (QROM) Analysis for VOLE-in-the-Head Signatures[12]

Abstract. *In the past decade and largely in response to the NIST standardization effort for post-quantum cryptography, many new designs for digital signatures have been proposed. Among those, the FAEST digital signature scheme (Baum et al., CRYPTO 2023) stands out due to its interesting security-performance trade-off. It only relies on well-tested symmetric-key cryptographic primitives, as it constructs a digital signature from a Zero-Knowledge (ZK) proof of knowledge of an AES key. To achieve this, it uses the VOLE-in-the-head ZK proof system which relies only on Pseudorandom generator (PRG) and hash function calls. FAEST simultaneously has relatively small signature size and competitive sign and verify times.*

[12] This section is based on [21].

In this work, we improve both the security and practical efficiency of FAEST. We improve the main computational bottleneck of the original construction by replacing hash function calls in the underlying vector commitment scheme with calls to an AES-based PRG. At the same time, we also improve the signature size by revisiting the evaluation of the AES block cipher in ZK. We use observations from Galois Theory to compress the size of the witness (and thus signature), due to the algebraic nature of the AES S-Box. We implemented our new construction, and our benchmarks show that its sign and verify times reduce up to 50% over the state-of-the-art while achieving the same security and smaller signatures.

Finally, we analyze our resulting signature scheme both in the Quantum Random Oracle Model (QROM) and its classical analogue. To achieve concretely good security bounds, we devise a new classical proof for FAEST based on Renyi divergence techniques. We construct a QROM analogue and present a new Fiat Shamir transform which is applicable to VOLE-in-the-head-based signature schemes.

Digital signatures are a core component of cryptographic systems, ensuring authenticity and integrity in digital communications. However, traditional signature schemes like RSA and those based on discrete logarithms are vulnerable to attacks by quantum computers. In response, the U.S. National Institute of Standards and Technology (NIST) has begun standardizing post-quantum signature schemes. While three schemes have already been selected, NIST has acknowledged their limitations and issued a call for additional candidates. One such candidate is FAEST [9], [14], a signature scheme that, like SPHINCS+ [8], relies solely on symmetric cryptographic primitives such as AES and SHAKE, avoiding public-key assumptions that may be less thoroughly analyzed in the quantum context.

FAEST uses the VOLE-in-the-Head (VOLEitH) zero-knowledge proof system to prove knowledge of an AES key. This makes it a conservative and secure option. Compared to SPHINCS+, FAEST offers better performance in most areas except for verification time. However, when compared to lattice-based schemes like Dilithium [10], FAEST is less competitive, particularly in terms of computational efficiency. This is largely due to the performance bottlenecks inherent in the VOLEitH and MPC-in-the-Head [7], [12] paradigms, especially the generation and reconstruction of vector commitments using hash-based constructions. These operations are computationally expensive, especially when compared to AES, which benefits from hardware acceleration.

Another challenge for FAEST is its signature size. Although it significantly improves upon earlier schemes like Picnic [7], it still lags behind lattice-based alternatives. This is partly due to the way AES is represented in the zero-knowledge proof, which is not optimized for compactness. Additionally, while FAEST is claimed to be secure against quantum attacks, its security proof in the quantum random oracle model (QROM) is heuristic and relies on an unproven conjecture, with a large reduction loss that weakens its theoretical guarantees.

To address these issues, we introduce three major improvements to FAEST and the VOLEitH framework. First, they enhance the vector commitment scheme by replacing most hash-based commitments with a statistically binding scheme based on a pseudorandom generator and a universal hash function. This approach, inspired by Naor's commitment scheme, significantly improves computational efficiency. The new scheme is statistically secure in the random oracle model, though it has a computationally inefficient extractor. We adapt the security proof to accommodate this change, using a technique that reduces existential unforgeability to soundness rather than knowledge soundness.

Second, we optimize the zero-knowledge proof for AES evaluation. In previous schemes, proving AES required committing to every S-box input, which was computationally intensive. A recent idea suggested committing only to every second round's S-box outputs [9], but this required complex

degree-7 constraints. We propose a simpler alternative using Galois Theory, committing to full S-box outputs every second round and using smaller representations for intermediate rounds. This reduces the required constraint degree to three, making the circuit more efficient and practical for VOLEitH.

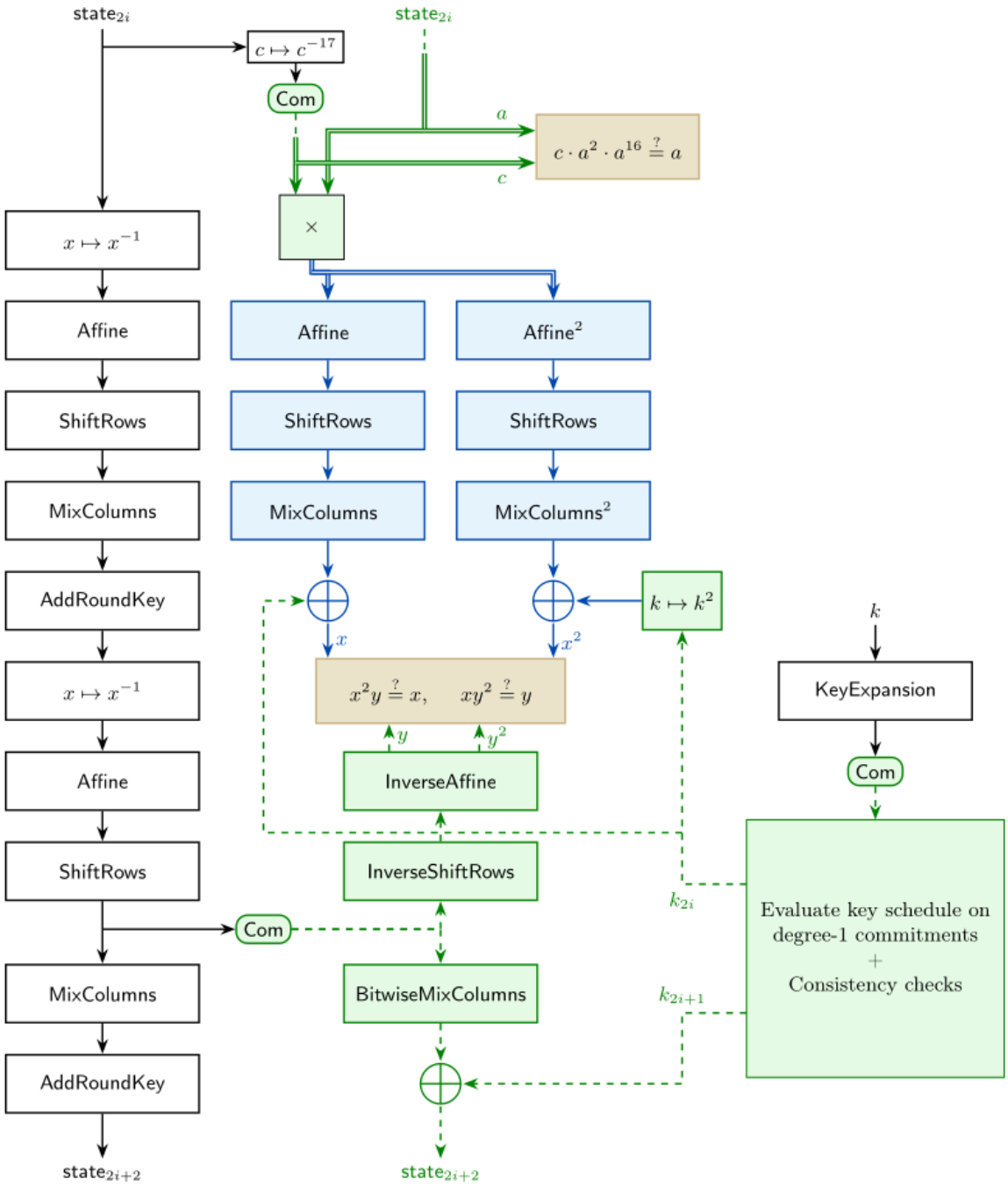

Figure 6: Optimized circuits to decrease the size of the signatures.

Third, we provide a rigorous QROM proof of FAEST's security. They reduce the EUF-CMA security of FAEST to the security of AES and SHAKE, achieving a small constant reduction loss and only minor additive losses due to Grover's algorithm. This proof introduces three technical innovations that are presented in a general form to facilitate reuse in other cryptographic constructions.

In summary, this work significantly improves the efficiency, security, and theoretical foundations of the FAEST signature scheme. By optimizing vector commitments, simplifying AES proofs, and providing a tighter quantum security proof, we enhance FAEST's viability as a post-quantum digital signature

candidate. These contributions also advance the broader development of symmetric-key-based post-quantum cryptography.

Variant		|sk|	|pk|	Fast			Small		
				$\|\sigma\|$	Sign	Verify	$\|\sigma\|$	Sign	Verify
128	[BBM+24]	32	32	6 052	878	802	4 594	6 485	5 790
	this	32	32	5 924	524	432	4 506	3 878	3 041
192	[BBM+24]	56	64	16 100	2 149	1 996	12 028	18 545	14 598
	this	40	48	14 948	2 030	1 759	11 260	15 038	11 475
256	[BBM+24]	64	64	28 084	3 321	3 226	21 752	25 169	25 364
	this	48	48	26 548	3 067	2 901	20 696	24 716	24 726
EM-128	[BBM+24]	32	32	5 444	852	786	4 170	6 389	6 077
	this	32	32	5 060	455	337	3 906	2 809	2 253
EM-192	[BBM+24]	48	48	13 532	1 944	1 899	10 108	17 423	16 800
	this	48	48	12 380	1 467	1 333	9 340	11 401	10 576
EM-256	[BBM+24]	64	64	26 036	3 260	3 119	19 744	25 382	24 553
	this	64	64	23 476	2 615	2 431	17 984	17 537	17 004

Figure 7: Comparison with previous work [9] showing the overall improvements in runtime and signature size of this work.

6. Secure Key Forwarding for Large-Scale quantum Key Distribution Networks[13]

Abstract. *Secure communication in large-scale quantum key distribution (QKD) networks relies on the key management layer to establish a key between any two nodes in the network. For non-adjacent nodes key forwarding is applied and secure keys are one-time pad encrypted with key material obtained from individual QKD links on a hop-by-hop basis from the initiator to the receiver. Notably, in a distributed setting, this requires that the nodes along the path be fully trusted, since they are given plaintext access to the secret during re-encryption.*

In this work, we propose a security model for the key forwarding process that provides confidentiality of the key with respect to all nodes that are not on the selected path from initiator to receiver. The model also captures the authenticity of the forwarded key in a sense that the key at the receiver is the same as the one sent by the initiator. Furthermore, in case the authenticity check fails, the receiver can identify the link that violated the authenticity guarantees by accident or on purpose. Finally, we provide a protocol for secure authentic key forwarding based on unconditionally hiding commitment schemes and unforgeable signature schemes that achieves confidentiality of the transmitted keys in an information theoretic sense of security whereas authenticity requires computationally secure primitives.

Quantum Key Distribution (QKD) enables the secure distribution of symmetric random bit strings with proven security even against eavesdroppers with unbounded computational power, providing

[13] This section is based on [22].

information-theoretic security. As computing technologies rapidly advance, QKD is expected to play an important role in securing the transmission of critical data, finding applications beyond the typical QKD-enabled VPN use case. Its practical relevance has also been demonstrated in many testbeds. However, QKD systems face challenges due to limited transmission distances between directly connected endpoints, typically limited to a few hundred kilometers, as quantum signals degrade over long distances due to noise and interference.

To overcome this limitation in large-scale networks, a Quantum Key Distribution Network (QKDN) employs Trusted Nodes (TN) as intermediaries. These nodes relay keys across the network by decrypting and re-encrypting key material, enabling secure key forwarding over long distances. This process is managed by the QKD Key Management System (KMS), which coordinates key distribution throughout the network. In a typical QKD network with a decentralized KMS, to establish an end-to-end shared key, the source KMS generates a random value to serve as the final symmetric key, encrypts it using the QKD-generated key for the next node, and forwards it. Each intermediate node KMS decrypts the key using its own corresponding QKD-generated key and re-encrypts it for the next node before forwarding it. This process continues until the destination node KMS is reached, enabling secure key sharing between distant endpoints. The use of a One-Time Pad (OTP) ensures that the symmetric keys shared between the two end-nodes remain secure for application use, thus facilitating end-to-end encrypted communication over large distances.

The use of Trusted Nodes, however, imposes a strong security assumption on these nodes, since they see all secret keys in plain. If communicating parties do not have a direct quantum channel between them, the desired goal of end-to-end security can only be achieved by using additional protocols. Examples of such protocols are Muckle [6] and Muckle+ [1], which reduce the trust requirements by combining QKD with techniques from authenticated key exchanges for TLS-like protocols.

Aside from the end-to-end security aspect, one may ask how the receiving node can verify the authenticity of the received secret. No such functionality is currently envisioned in the recommendations for key management systems of QKD networks. Indeed, cases where a transmission error occurs, an intermediate node changes the forwarded key, an adversary gains access to a link and modifies some part of the key, or other cases where the transmitted key is altered are impossible for the destination node to detect. While protocols such as Muckle+ would be able to detect modified keys, the overall system would not be able to detect which link or node was responsible for the transmission error, thus providing easy ways to mount denial-of-service attacks in the QKD network.

In this work we tackle the problems stated above in the following way: We introduce a formal security model for the process of key forwarding called authenticated key forwarding which covers passive and active adversaries in a QKD network based on their capabilities of interacting with the protocol. For passive adversaries we consider honest-but-curious parties of the network that are able to observe the communication of the network without having access to the QKD keys used to encrypt the traffic along a specific link. In this case we are concerned with the ability to derive any knowledge on the secret key from the observed communication. In the second case, we consider active adversaries which may have compromised nodes along the path of a key forwarding process. While in this scenario the adversary is aware of the transmitted secret, we want to protect the receiver from being sent an incorrect key. In the case that the receiver receives an invalid key, the receiver should also detect the link where an error was introduced (either by accident or maliciously).

Pre phase

if $S = \perp$ then A will randomly get $S \xleftarrow{\$} \mathcal{K}$
A runs $\mathsf{Commit}(H_K(S); r) \rightarrow ct, o$
A runs $\mathsf{Sign}(ct, \mathsf{sk}_A) \rightarrow \boldsymbol{\sigma}_0$
for $j = 0, \ldots, k$ **do**
  $K_{i_j, i_{j+1}} \leftarrow \mathsf{QKGen}(i_j, i_{j+1}, 1^{m+n})$
end for

Transmission$_j$:

Node P_{i_j} has S, o, ct and signature vector $\boldsymbol{\sigma}$.
P_{i_j} computes $\boldsymbol{\mathcal{S}}_j \leftarrow o \| S \oplus K_{i_j, i_{j+1}}$
if $j = 0$ **then** A computes $\boldsymbol{\sigma}_0 \leftarrow \mathsf{Sign}(\mathsf{sk}_A, ct)$
else
  P_{i_j} computes $\boldsymbol{\sigma}_j \leftarrow \mathsf{Sign}\left(\mathsf{sk}_{i_j}, ct \| \boldsymbol{\sigma}_{j-1}\right)$
P_{i_j} sends $m_0 = \boldsymbol{\mathcal{S}}_j$, $m_1 = ct$, $m_2 = \boldsymbol{\sigma}$ to $P_{i_{j+1}}$
Upon receiving m_0, m_1 and m_2, $P_{i_{j+1}}$ sets:
$o \| S \leftarrow m_0 \oplus K_{i_j, i_{j+1}}$, $ct \leftarrow m_1$, $\boldsymbol{\sigma} \leftarrow m_2$
$P_{i_{j+1}}$ computes $\mathsf{Open}(ct, H_K(S), o)$, **if** it is 0
  then return corruptedLink $\leftarrow j$

Figure 8: Pre-phase and transmission algorithms of the proposed protocol.

We provide a construction of the protocol based on unconditionally hiding commitment schemes and unforgeable digital signature schemes. Despite the use of a computationally secure cryptographic primitive, we show that the security against a passive adversary holds in an information-theoretic sense. Interestingly, this observation allows us to securely instantiate the protocol with a commitment scheme such as the one of Pedersen [23] which relies on the hardness of the discrete logarithm problem. If security against an active adversary is required in a quantum-resistant sense, then the use of a post-quantum signature (for example a signature scheme that is constructed from symmetric primitives only such as [7], [9]) and commitment scheme (e.g. as in [24]) is required. However, even this case a quantum adversary would be required to break binding property of the commitment scheme in seconds which would be infeasible even if a quantum computer powerful enough to break the discrete logarithm problem would exist.

Summary

Extending the Key Management System with techniques from key exchange protocols and other areas of classical cryptography with post-quantum secure instantiations provides end-to-end security and additional security guarantees to the keys of a QKD network. Most importantly, the ITS security of the ITS keys is not compromised by combining them with PQ key material. Instead, one obtains a system that is resilient to different kind of cryptographic breaks.

Appendix A - List of Acronyms

- QKD: Quantum Key Distribution
- KMS: Key Management System
- ITS: Information-theoretic security
- MPCitH: MPC-in-the-Head
- VOLEitH: VOLE-in-the-Head
- MPC: Multi-party computation
- VOLE: Vector oblivious linear equation
- QROM: Quantum random oracle model
- IP: interactive proof
- PRG: pseudo random generator
- ZK: zero-knowledge
- GGM: Goldreich-Goldwasser-Micali
- TN: Trusted Node
- PSK: pre-shared key
- p2p: peer-to-peer
- e2e: end-to-end
- ZKPoK: zero-knowledge proof of knowledge
- MAC: message authentication code
- SDN: software defined network
- OWF: one way function
- PRF: pseudro-random function
- FS: Fiat-Shamir
- PQ: post-quantum
- PQC: post-quantum cryptography

Appendix B – Bibliography

[1] S. Bruckner, S. Ramacher, and C. Striecks, "Muckle+: End-to-End Hybrid Authenticated Key Exchanges.," in *Post-Quantum Cryptography - 14th International Workshop, PQCrypto 2023, College Park, MD, USA, August 16-18, 2023, Proceedings*, 2023, pp. 601–633. doi: 10.1007/978-3-031-40003-2_22.

[2] P. James, S. Laschet, S. Ramacher, and L. Torresetti, "Key Management Systems for Large-Scale Quantum Key Distribution Networks.," in *Proceedings of the 18th International Conference on Availability, Reliability and Security, ARES 2023, Benevento, Italy, 29 August 2023- 1 September 2023*, 2023, p. 126:1-126:9. doi: 10.1145/3600160.3605050.

[3] *ETSI GS QKD 004: Quantum Key Distribution (QKD); Application Interface*, 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/QKD/001_099/004/02.01.01_60/gs_qkd004v020101p.pdf

[4] *ETSI GS QKD 014: Quantum Key Distribution (QKD); Protocol and data format of REST-based key delivery API*, 2019. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/QKD/001_099/014/01.01.01_60/gs_qkd014v010101p.pdf

[5] *ETSI GS QKD 015: Quantum Key Distribution (QKD); Control Interface for Software Defined Networks*, 2022.

[6] B. Dowling, T. B. Hansen, and K. G. Paterson, "Many a Mickle Makes a Muckle: A Framework for Provably Quantum-Secure Hybrid Key Exchange," in *Post-Quantum Cryptography*, J. Ding and J.-P. Tillich, Eds., Cham: Springer International Publishing, 2020, pp. 483–502.

[7] M. Chase *et al.*, "Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives.," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, 2017, pp. 1825–1842. doi: 10.1145/3133956.3133997.

[8] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The SPHINCS$^+$ Signature Framework," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, London United Kingdom: ACM, Nov. 2019, pp. 2129–2146. doi: 10.1145/3319535.3363229.

[9] C. Baum *et al.*, "One Tree to Rule Them All: Optimizing GGM Trees and OWFs for Post-Quantum Signatures.," in *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part I*, 2024, pp. 463–493. doi: 10.1007/978-981-96-0875-1_15.

[10] L. Ducas *et al.*, "CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, pp. 238–268, Feb. 2018, doi: 10.46586/tches.v2018.i1.238-268.

[11] P.-A. Fouque *et al.*, "Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU," 2020. [Online]. Available: https://falcon-sign.info/falcon.pdf

[12] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Zero-knowledge proofs from secure multiparty computation," *SIAM J Comput*, vol. 39, 2009, doi: 10.1137/080725398.

[13] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai, "Compressing Vector OLE," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, Toronto Canada: ACM, Oct. 2018, pp. 896–912. doi: 10.1145/3243734.3243868.

[14] C. Baum *et al.*, "Publicly Verifiable Zero-Knowledge and Post-Quantum Signatures from VOLE-in-the-Head," in *Advances in Cryptology – CRYPTO 2023*, vol. 14085, H. Handschuh and A. Lysyanskaya, Eds., in Lecture Notes in Computer Science, vol. 14085. , Cham: Springer Nature Switzerland, 2023, pp. 581–615. doi: 10.1007/978-3-031-38554-4_19.

[15] C. D. De Saint Guilhem, L. De Meyer, E. Orsini, and N. P. Smart, "BBQ: Using AES in Picnic Signatures," in *Selected Areas in Cryptography – SAC 2019*, vol. 11959, K. G. Paterson and D. Stebila, Eds., in Lecture Notes in Computer Science, vol. 11959. , Cham: Springer International Publishing, 2020, pp. 669–692. doi: 10.1007/978-3-030-38471-5_27.

[16] C. Delpech De Saint Guilhem, E. Orsini, and T. Tanguy, "Limbo: Efficient Zero-knowledge MPCitH-based Arguments," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and*

Communications Security, Virtual Event Republic of Korea: ACM, Nov. 2021, pp. 3022–3036. doi: 10.1145/3460120.3484595.

[17] C. Baum, C. D. De Saint Guilhem, D. Kales, E. Orsini, P. Scholl, and G. Zaverucha, "Banquet: Short and Fast Signatures from AES," in *Public-Key Cryptography – PKC 2021*, vol. 12710, J. A. Garay, Ed., in Lecture Notes in Computer Science, vol. 12710. , Cham: Springer International Publishing, 2021, pp. 266–297. doi: 10.1007/978-3-030-75245-3_11.

[18] C. Dobraunig, D. Kales, C. Rechberger, M. Schofnegger, and G. Zaverucha, "Shorter Signatures Based on Tailor-Made Minimalist Symmetric-Key Crypto," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, Los Angeles CA USA: ACM, Nov. 2022, pp. 843–857. doi: 10.1145/3548606.3559353.

[19] W. Beullens, T. Kleinjung, and F. Vercauteren, "CSI-FiSh: Efficient Isogeny Based Signatures Through Class Group Computations," in *Advances in Cryptology – ASIACRYPT 2019*, vol. 11921, S. D. Galbraith and S. Moriai, Eds., in Lecture Notes in Computer Science, vol. 11921. , Cham: Springer International Publishing, 2019, pp. 227–247. doi: 10.1007/978-3-030-34578-5_9.

[20] R. Benadjila, T. Feneuil, and M. Rivain, "MQ on my Mind: Post-Quantum Signatures from the Non-Structured Multivariate Quadratic Problem," in *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*, Vienna, Austria: IEEE, Jul. 2024, pp. 468–485. doi: 10.1109/EuroSP60621.2024.00032.

[21] C. Baum *et al.*, "Shorter, Tighter, FAESTer: Optimizations and Improved (QROM) Analysis for VOLE-in-the-Head Signatures," in *CRYPTO 2025*, 2025.

[22] N. Franzoi, S. Krenn, T. Lorunser, and S. Ramacher, "Secure Key Forwarding for Large-Scale Quantum Key Distribution Networks," in *2025 International Conference on Quantum Communications, Networking, and Computing (QCNC)*, Apr. 2025, pp. 486–493. doi: 10.1109/QCNC64685.2025.00082.

[23] T. P. Pedersen, "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing," in *Advances in Cryptology — CRYPTO '91*, vol. 576, J. Feigenbaum, Ed., in Lecture Notes in Computer Science, vol. 576. , Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 129–140. doi: 10.1007/3-540-46766-1_9.

[24] C. Baum, I. Damgård, V. Lyubashevsky, S. Oechsner, and C. Peikert, "More Efficient Commitments from Structured Lattice Assumptions," in *Security and Cryptography for Networks*, vol. 11035, D. Catalano and R. De Prisco, Eds., in Lecture Notes in Computer Science, vol. 11035. , Cham: Springer International Publishing, 2018, pp. 368–385. doi: 10.1007/978-3-319-98113-0_20.