Quantum Key Distribution for Railway:

Evaluating Quantum-Secure TLS For Mobile Key Provisioning

Midterm Presentation

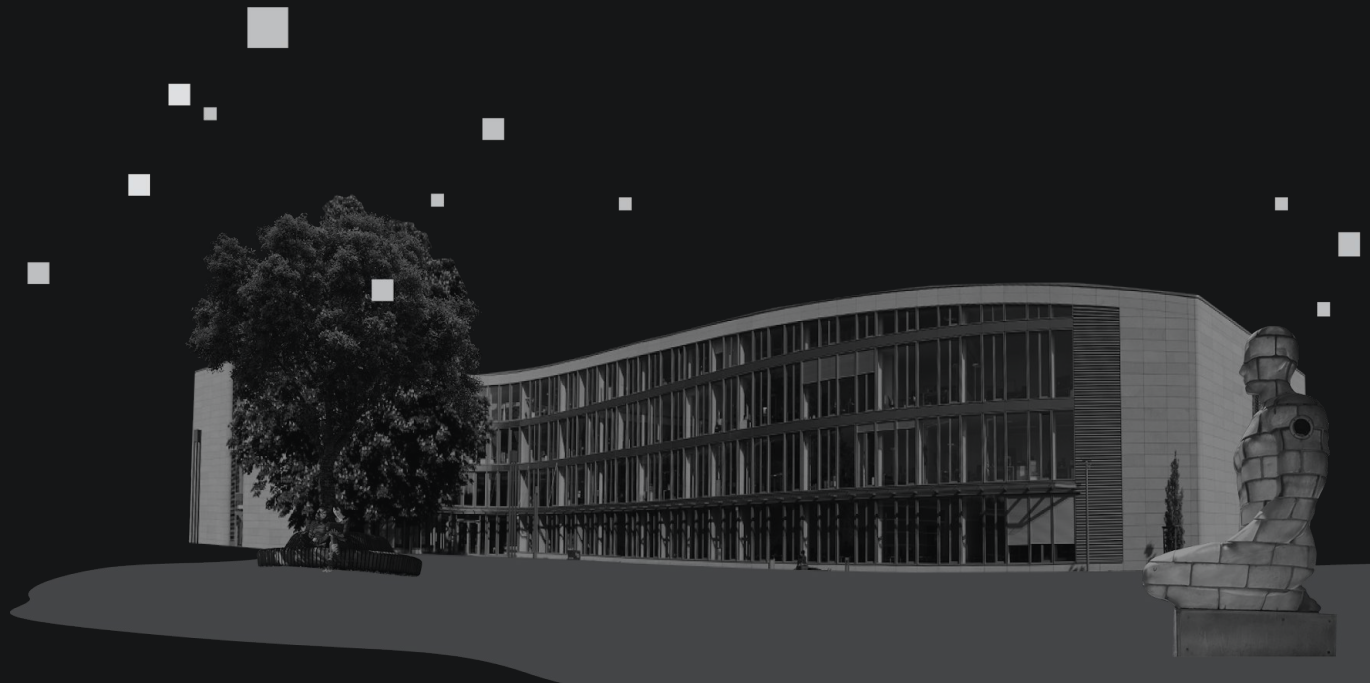
Sebastian Staib

16.06.2026

PROFESSOR	**Prof. Dr. Andreas Polze**
CHAIR	**Operating Systems and Middleware**
SUPERVISION	**Katja Assaf, Lukas Pirl**

Design IT.
Create Knowledge.

www.hpi.de



Agenda

1 Motivation, Background And Objectives

Quantum Threat | QKD/PQC Directions | Scope Of This Work

2 Post-Quantum Secure TLS Alternatives And Measurement Framework

TLS Variant Comparison | Measurement Framework | Showcase And Results

3 Mobile QKD Key Provisioning

Key Filling Station Architecture | Android Proof of Concept

4 Integrated View And Outlook

How Both Parts Connect | Future Measurements With Real QKD Hardware

Motivation, Background And Objectives

Quantum Threat | QKD/PQC Directions | Scope Of This Work

Motivation

Long-lived railway systems need migration paths before large quantum computers become practical

Rise Of Quantum Computers

Large-scale QC expected
Within planning horizons
For long-lived systems

Shor's Algorithm

Factorization
Discrete logarithms
RSA, DH, ECDH at risk

Harvest Now, Decrypt Later

Record traffic today
Decrypt later
Start migration now

Goal: Integrate quantum-secure key material into communication paths

Two Approaches to Quantum-Secure Systems

Post-Quantum Cryptography

- Software-oriented and based on hard mathematical problems
- Standards published by NIST in 2024:
  - ML-KEM, ML-DSA and SLH-DSA [1]
- Security assumptions include lattice-based and hash-based constructions

Quantum Key Distribution

- Relies on physics and is hardware-oriented
- Encoding information in quantum states [2]
- Generates information-theoretically secure key material
- Can provide perfectly secure ciphertexts, e.g. One-Time-Pad [3]

[1] https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards
[2] https://doi.org/https://doi.org/10.1016/j.tcs.2014.05.025
[3] https://doi.org/10.1002/j.1538-7305.1949.tb00928.x

Thesis Aim: Two Contribution Lines

TLS Variants And Measurement Framework

Protocol Side

TLS Variant Comparison

Classical TLS, PSK, Hybrid, QKD-assisted and KEMTLS paths

TLS Measurement Framework

Reproducible comparison of TLS, PQ, and QKD-based session setups

+

KFS Concept And Android Proof Of Concept

Mobile Key-Provisioning

Key Filling Station (KFS) Architecture

Secure-site QKD keys, train key reservoir and RBC-side TLS peer with matching QKD keys

Android KFS Proof Of Concept

Pixel client models the mobile endpoint and establishes TLS 1.3 PSK using provisioned key material

Combined Thesis Contribution:

The thesis connects TLS migration evidence with mobile QKD-key provisioning.

Post-Quantum Secure TLS Alternatives And Measurement Framework

TLS Variant Comparison | Measurement Framework | Showcase And Results

Why Compare TLS Variants?

QKD gives key material, but the protected channel still needs protocol integration.

QKD Gives Keys, but…

Symmetric key material
Not a complete channel

TLS Gives Channel

Authentication
Key Schedule
Protected Traffic

Different Integration points

Key Exchange
PSK Path
Authentication
KME Access

Comparable Criteria

Key schedule
Interoperability
Mobile Use-Case Fit

TLS Variants Reviewed in This Thesis

The variants are grouped by baseline role, post-quantum direction and QKD-assisted key use.

Baselines

Reference Points

- TLS 1.3 cert
- TLS 1.3 PSK

PQC / Hybrid

Quantum-Resistant TLS Paths

- Hybrid TLS
- KEMTLS

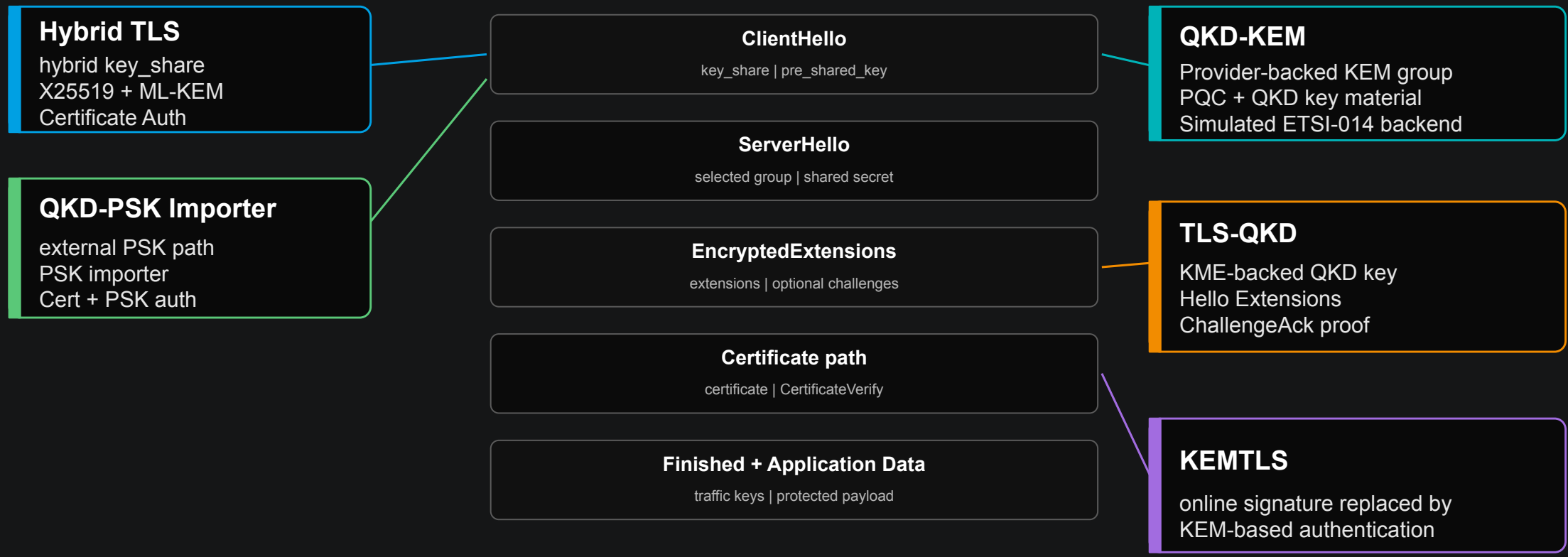
QKD-Assisted Paths

Pre-Provisioned Or KME-Backed Key Use

- QKD-PSK Importer
- QKD-KEM
- TLS-QKD

TLS Variant Hook-In Points

Simplified from Chapter 4: one TLS 1.3 orientation layer, separate adaptation points.

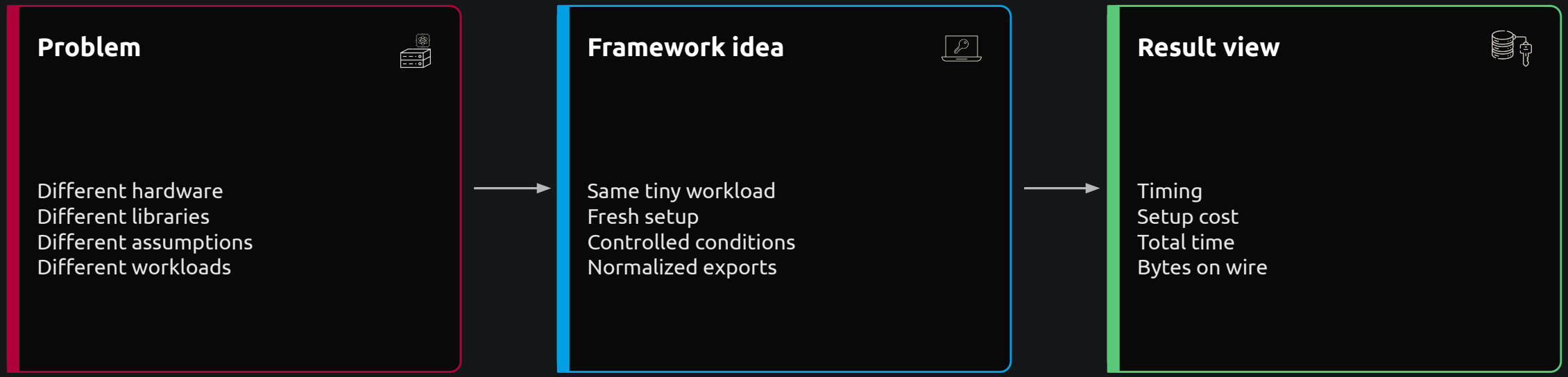


Separate variant paths: key_share / PSK schedule / KME extensions / authentication

The colors show hook-in points, not one combined TLS handshake.

Why a Shared Measurement Framework?

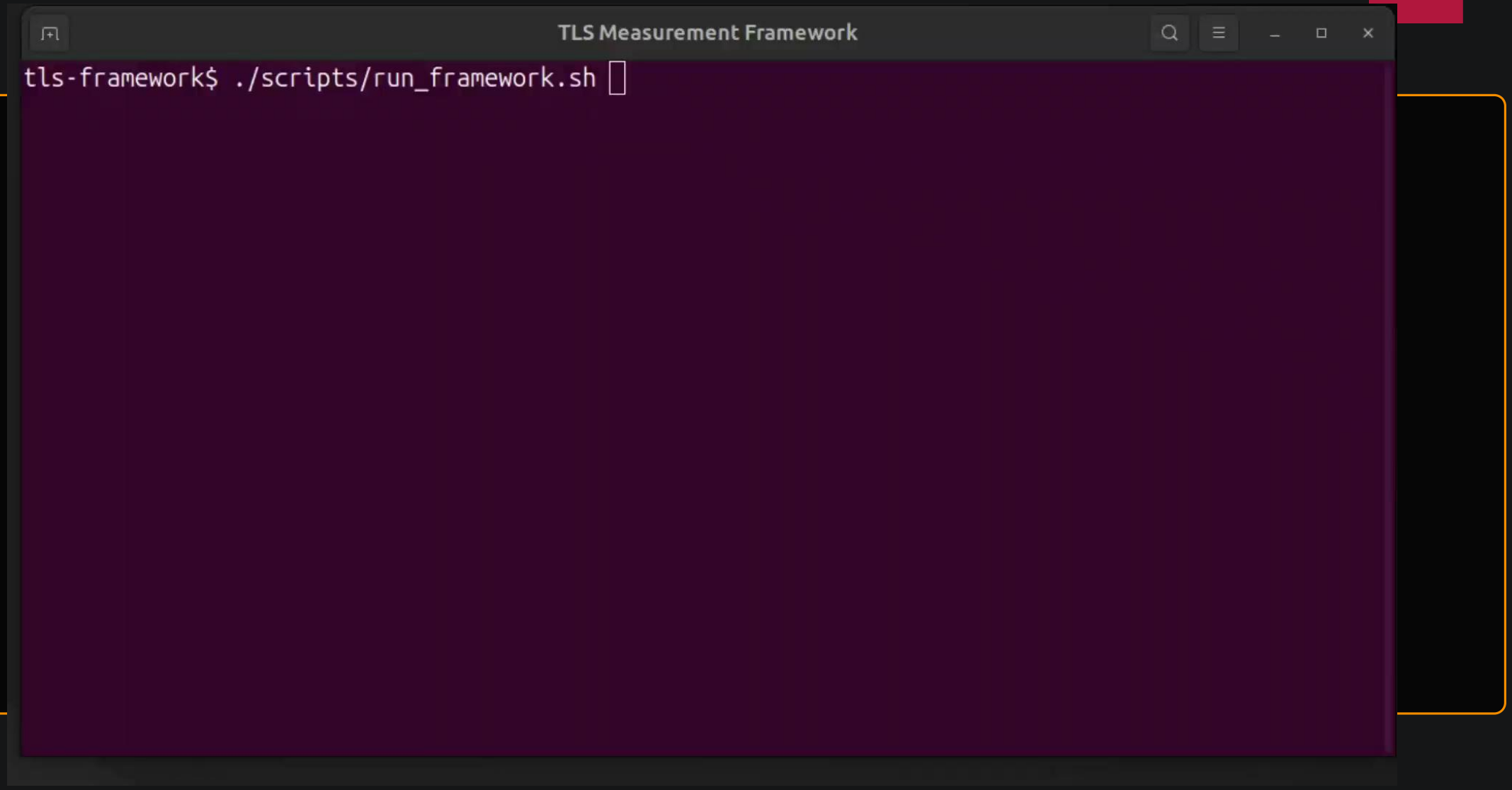
A qualitative TLS comparison is not enough for implementation behavior.



Goal: Repeatable evidence for selected session-establishment paths.

Framework is located at: https://gitlab.com/squirrl-project/tls-measurement-framework

TLS Measurement Framework Showcase



Clean Condition: Cost and Wire Footprint

Variant	TLS norm. p50	Setup p50	End-to-end p50	Wire Bytes per Handshake p50 / vs Cert
TLS 1.3 Cert	1.53 ms / 1.00x	-	2.02 ms / 1.00x	2,790 B / 1.00x
TLS 1.3 PSK	0.93 ms / 0.61x	0.04 ms	4.20 ms / 2.08x	1,701 B / 0.61x
Hybrid TLS	1.65 ms / 1.08x	-	2.17 ms / 1.08x	5,064 B / 1.82x
QKD-PSK Importer	3.91 ms / 2.55x	0.05 ms	4.73 ms / 2.34x	3,335 B / 1.20x
TLS-QKD	2.56 ms / 1.67x	149.81 ms	153.48 ms / 75.91x	2,371.5 B / 0.85x
QKD-KEM	8.60 ms / 5.61x	200.46 ms	209.88 ms / 103.80x	5,223 B / 1.87x
KEMTLS	0.64 ms / 0.42x	-	1.36 ms / 0.67x	11,717 B / 4.20x

Each comparable cell shows absolute value / factor. Timing and wire factors are relative to TLS 1.3 Cert. Setup is shown as absolute dependency cost because setup scopes differ across variants.

Delay/Jitter: Operational Impact

Injected Network Conditions: +20 ms delay, 5ms jitter

Variant	TLS norm. p50	Setup p50	End-to-end p50	End-to-end p95
TLS 1.3 Cert	50.19 ms / 1.00x	-	172.55 ms / 1.00x	198.79 ms / 1.00x
TLS 1.3 PSK	41.32 ms / 0.82x	0.04 ms	168.33 ms / 0.98x	189.32 ms / 0.95x
Hybrid TLS	47.15 ms / 0.94x	-	170.93 ms / 0.99x	193.31 ms / 0.97x
QKD-PSK Importer	53.76 ms / 1.07x	0.04 ms	176.42 ms / 1.02x	206.55 ms / 1.04x
TLS-QKD	110.00 ms / 2.19x	483.65 ms	716.24 ms / 4.15x	880.87 ms / 4.43x
QKD-KEM	49.03 ms / 0.98x	200.34 ms	372.94 ms / 2.16x	396.26 ms / 1.99x
KEMTLS	47.47 ms / 0.95x	-	174.84 ms / 1.01x	198.77 ms / 1.00x

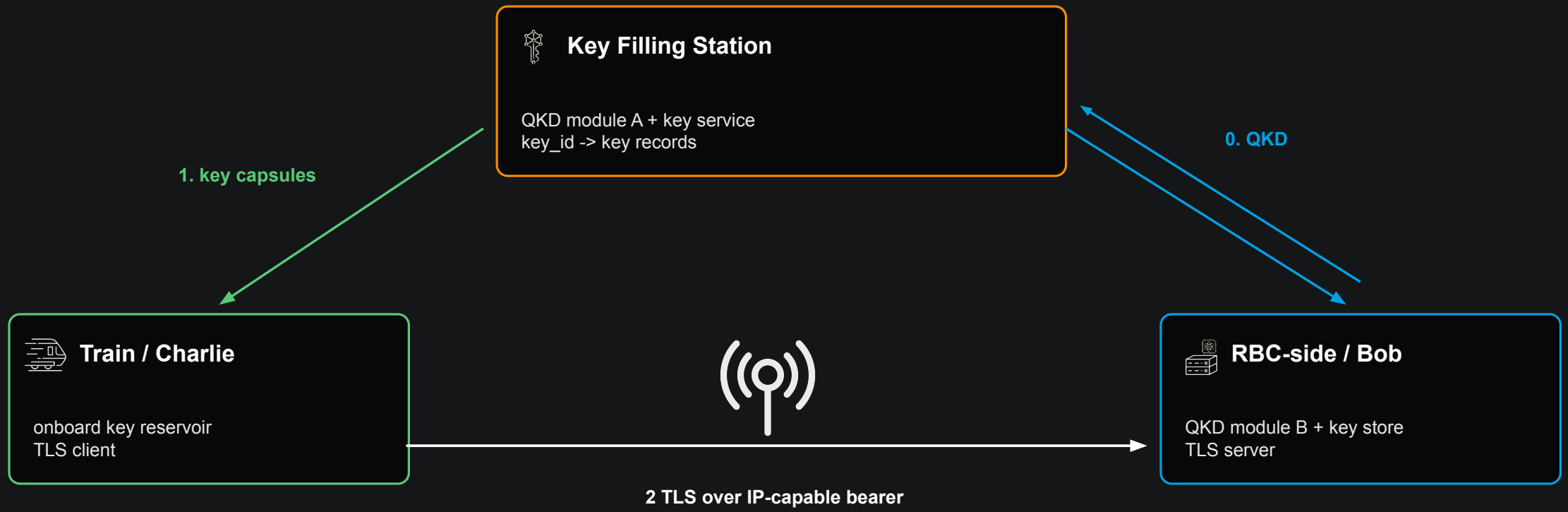
Each comparable timing cell shows absolute value / factor relative to TLS 1.3 Cert under the same delay/jitter condition. Wire-footprint values are not used as a main claim under impairment.

Mobile QKD Key Provisioning

Key Filling Station Architecture | Android Proof of Concept

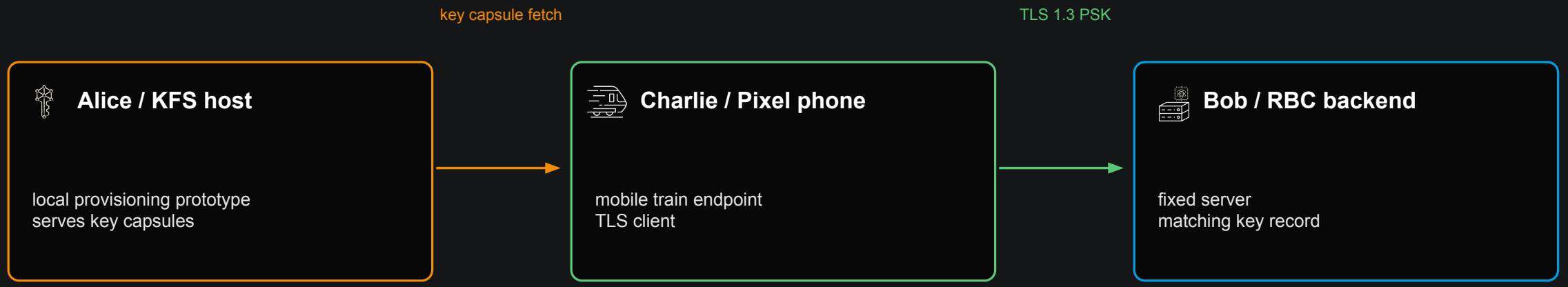
Key Filling Station Architecture

No QKD optics on the train: the train consumes provisioned key material.



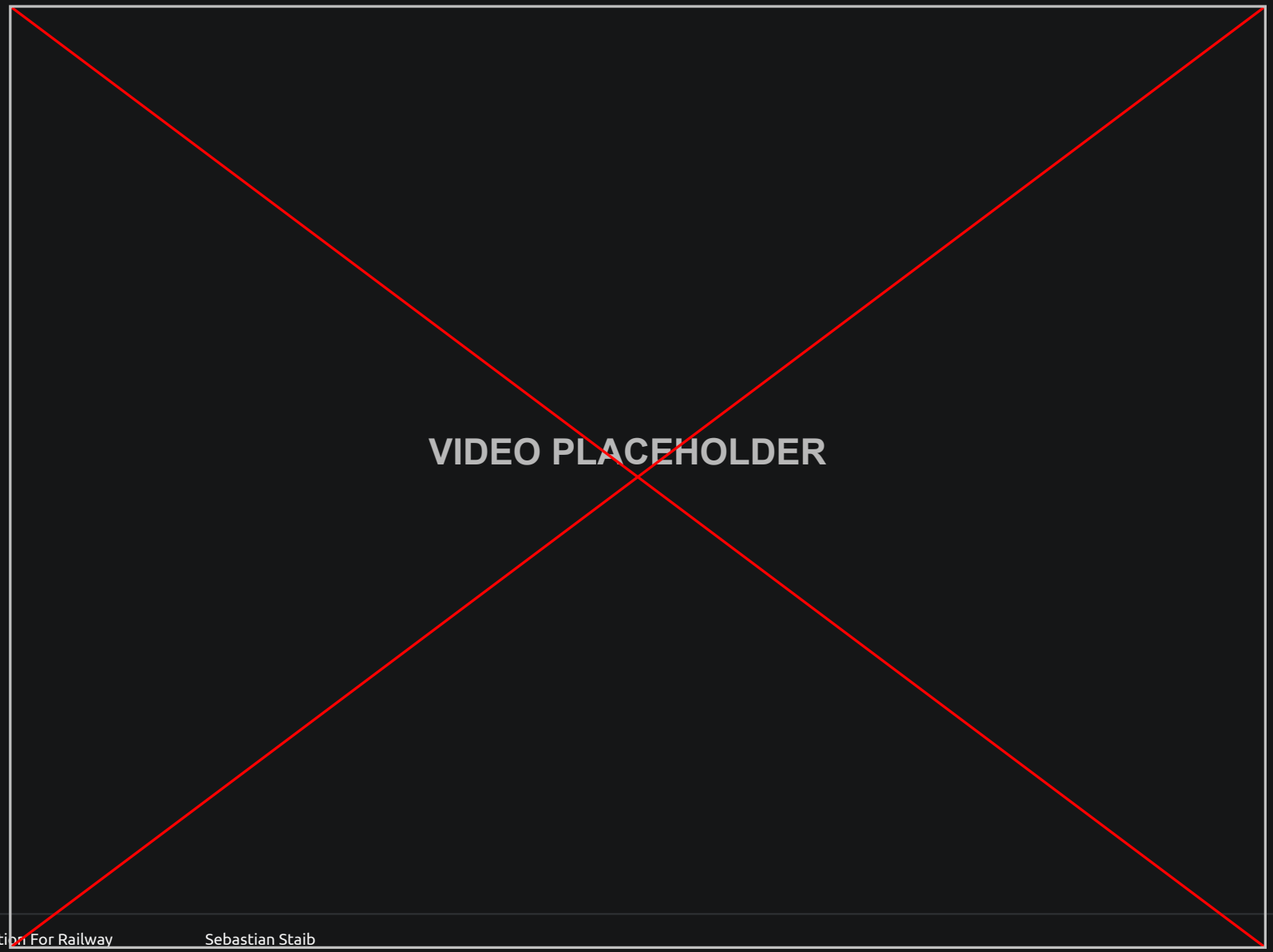
Presentation version of the Chapter 5 KFS target state.

Android KFS PoC - Role Mapping



key_id = PSK identity / lookup metadata | key bytes = external PSK secret

Android KFS PoC - Showcase



Integrated View And Outlook

How Both Parts Connect | Future Measurements With Real QKD Hardware

Integrated View: How Both Artefacts Fit Together

TLS Measurement Framework

compares session-establishment paths
under one run contract

future evaluation path

Android KFS PoC

validates mobile key consumption
on a real Pixel phone

Bridge

measure the prototype path and compare it with the framework variants

Current status: connected conceptually, not yet a production railway QKDN.

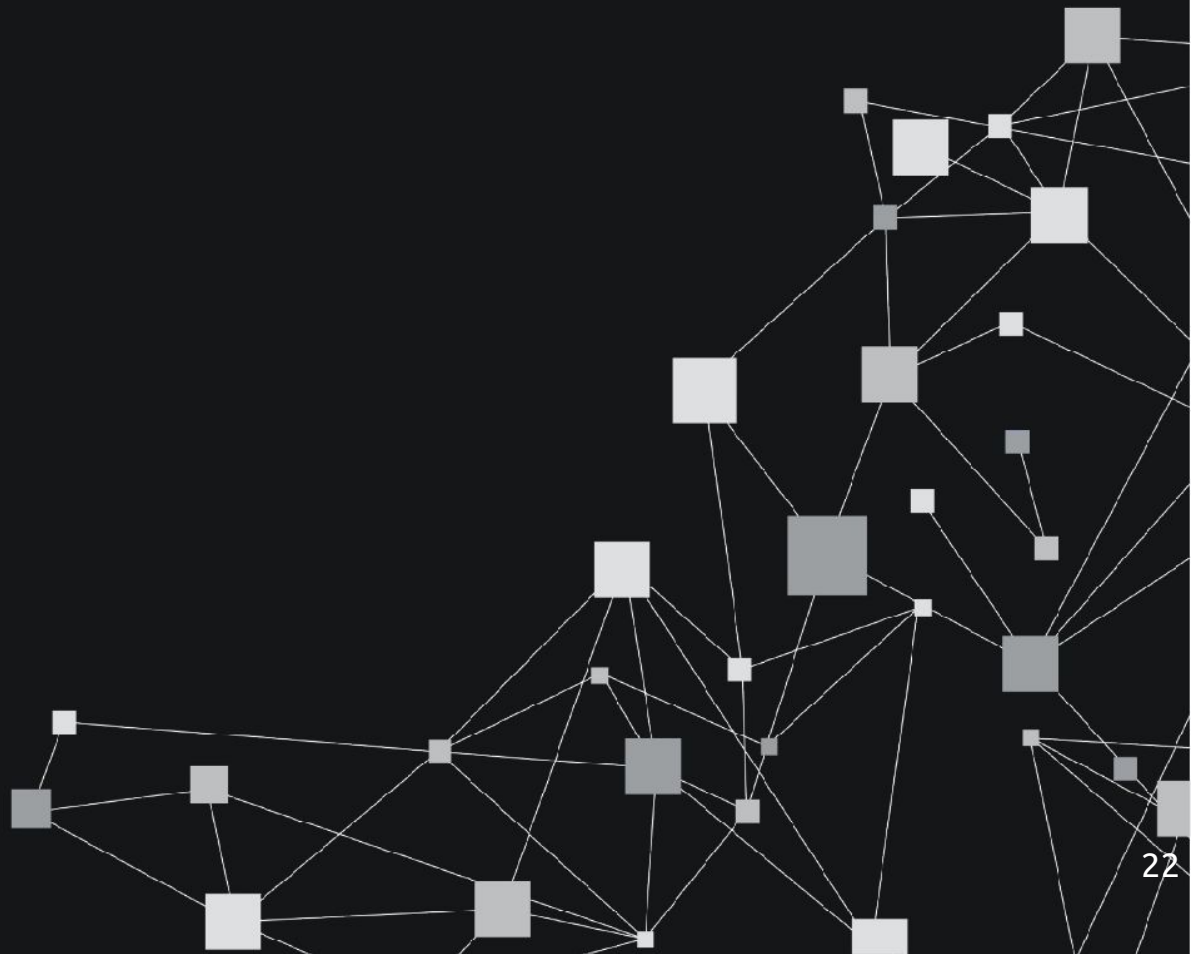
Summary and Outlook

- TLS Variant Comparison helped to select PoC-Design for KFS
- The TLS Measurement Framework makes different quantum-secure TLS paths comparable.
- The KFS Architecture maps QKD-derived keys to a mobile train endpoint without QKD optics on the train.
- The Android PoC validates mobile TLS 1.3 PSK key consumption on a real device.

Next Step

Feed real QKD-derived key material into the mobile Android prototype and measure with framework

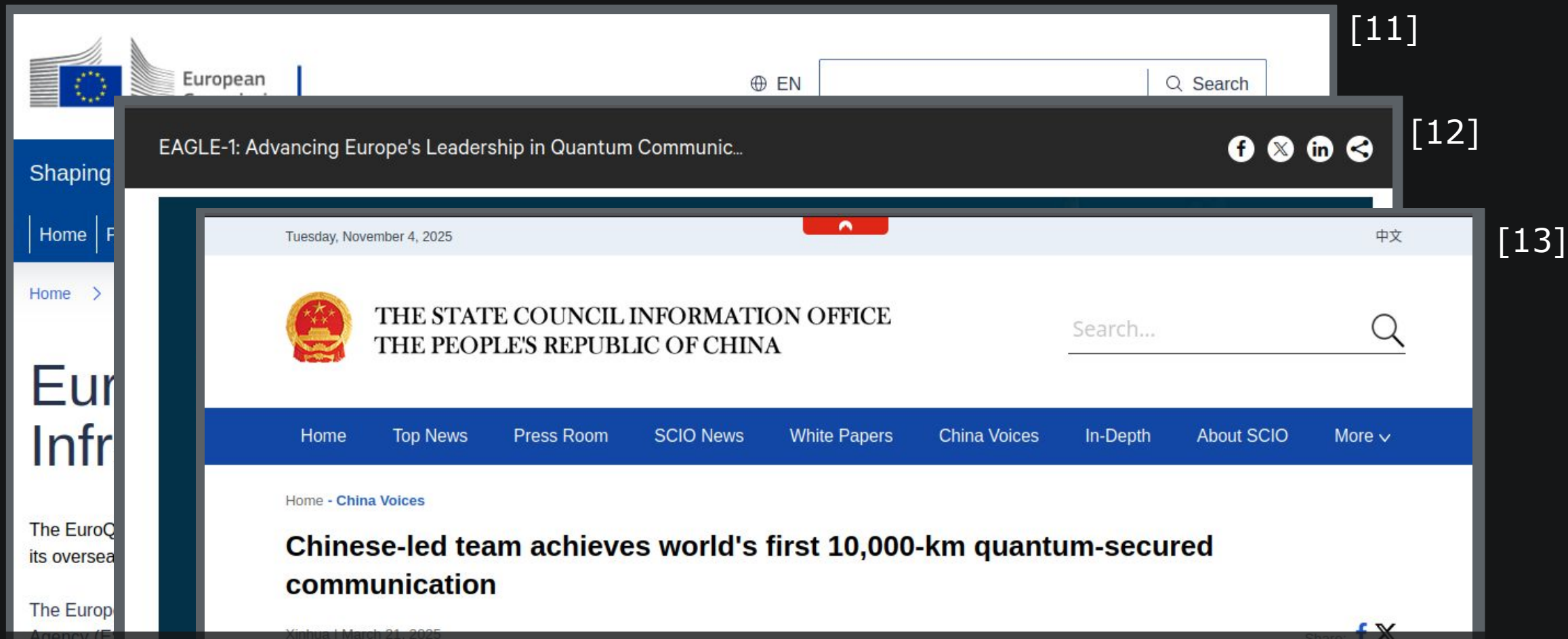
Backup Slides



Design IT.
Create Knowledge.

www.hpi.de

Backup: Recent QKD Developments



[11]

[12]

[13]

[11] https://digital-strategy.ec.europa.eu/en/policies/european-quantum-communication-infrastructure-euroqci
[12] https://www.ses.com/newsroom/eagle-1-advancing-europes-leadership-quantum-communications
[13] http://english.scio.gov.cn/chinavoices/2025-03/21/content_117778711.html

QKD Architecture

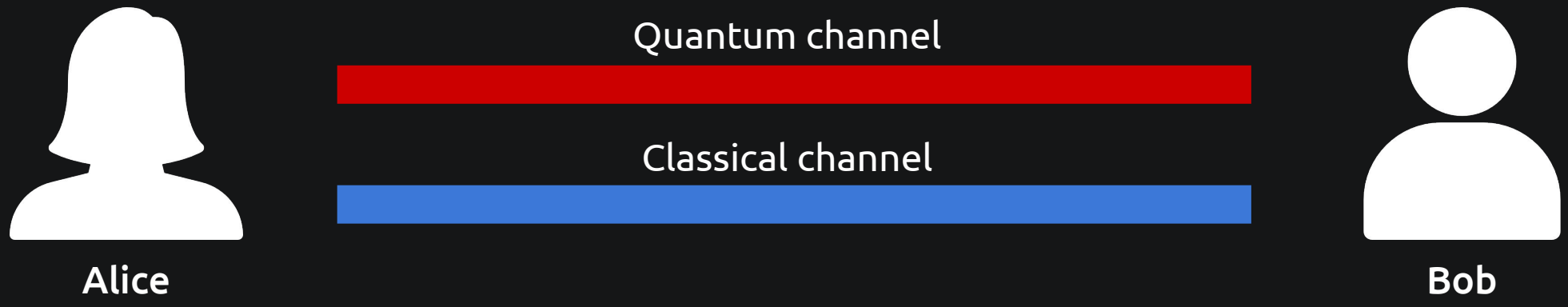


Fig 1. QKD setting. Alice and Bob are connected through a quantum channel and a classical channel [4]

QKD still needs authentication and key management around the physical link.

[4] based on Fig 1 in https://doi.org/10.1103/RevModPhys.81.1301

Backup: TLS Variants Overview

- **Hybrid TLS** changes the TLS key_share by combining X25519 with ML-KEM, while keeping certificate-based server authentication.
- **QKD-PSK** Importer uses the pre_shared_key path instead of key_share. The key material is locally available, so the measurement focuses on the importer path.
- **TLS-QKD** retrieves QKD key material through the KME side, adds QKD metadata in Hello extensions and uses ChallengeAck to prove shared key possession.
- **KEMTLS** replaces the online CertificateVerify signature with KEM-based implicit authentication, while certificates still bind the authentication material.
- **QKD-KEM** also works at the key-exchange layer. For TLS, it appears as a separate KEM group through an OpenSSL provider and combines PQC with simulated QKD key material.

Backup: TLS Measurement Framework - Implementation

- **Measurement flow:** Local measurement campaigns with validation and CSV export generation
- **Harness:** Fresh TCP connection, fresh TLS/TLS-like setup and one fixed protected PING/PONG exchange per run
- **Variants:** TLS Cert, TLS PSK, Hybrid TLS, QKD-PSK Importer, TLS-QKD, QKD-KEM and KEMTLS
- **Network conditions:** Clean baseline and 20 ms delay / 5 ms jitter using Linux tc netem
- **Capture:** Client timing split, setup/dependency cost, success status and per- run pcap traces
- **Trace analysis:** Packet count, wire bytes, selected TCP stream, TLS record/ handshake bytes, ClientHello and server-flight size
- **Scope:** Session-establishment comparison between local protocol paths, not application throughput or production deployment
- **Capture**: Timing split, setup/dependency cost, success status, pcap traces, packet count, wire bytes, selected TCP stream and TLS handshake-size fields

Backup: TLS Measurement Framework - Architecture

Layer	Own framework work	External mechanisms
Harness	Bash/Python orchestration	-
Variant path	wrappers, adapters, fixtures	OpenSSL, wolfSSL, QURSA, rustls-QKD, KEMTLS
Analysis	capture parsing, validation, exports	-

Backup: What the Results Separate

A single timing metric is misleading for QKD-related variants.

TLS normalized

TLS-near setup cost
after separating known setup work

Setup / dependency

local PSK setup
QKD API work
KME key acquisition

End-to-end

system view until
first protected payload

Wire

bytes and packets
when traces are valid

Clean: baseline cost + wire footprint

Delay/jitter: robustness + tail latency

Backup: TLS Variant Overview: Key Source and Hook-In Point

Variant	Key material / security idea	TLS integration point	Measured implementation path
TLS 1.3 cert	ECDHE + cert	reference handshake	OpenSSL
TLS 1.3 PSK	external PSK	pre_shared_key path	OpenSSL
Hybrid TLS	PQC + classical	key_share	OpenSSL 3.5.6
QKD-PSK Importer	local PSK fixture	external PSK importer	wolfSSL
QKD-KEM	simulated QKD + PQ	provider group	QURSA/OpenSSL
TLS-QKD	fixture-backed KME	KME path + extensions	rustls-QKD
KEMTLS	KEM certificates	authentication path	upstream KEMTLS

Message: Variants are structurally different, so the framework normalizes the workload around them.

Backup: TLS Measurement Framework

A reproducible harness for comparing TLS session-establishment paths.

What it is

Docker-based harness
seven TLS paths
one shared run contract

Why it exists

different libraries
different assumptions
one comparable setup

What it does

fresh TCP connection
fresh TLS setup
protected PING/PONG

What it exports

timing metrics
setup/dependency cost
packet + CSV artifacts

Key message: controlled local measurements, not a production benchmark.

Backup: TLS Measurement Framework - Procedure

Fresh TCP

one connection

Fresh setup

TLS or TLS-like

Protected App Data

PING / PONG

Capture

timing + packets

Export

normalized CSVs

Default Comparison: No session resumption | No 0-RTT | Fixed protected exchange

Backup: TLS Post-Quantum Secure Variants - Comparison Table

Criterion	TLS 1.3	Hybrid TLS	QKD-PSK Importer	TLS-QKD	QKD-KEM	KEMTLS
QKD material in key schedule	×	×	✓	✓	✓	×
External PSK support through standard handshake messages	✓	×	✓	×	×	×
PQ KEM contribution to key establishment	×	✓	×	×	✓	✓
Online QKD key-management dependency at session setup[1]	×	×	×*	✓	✓	×
Non-standard handshake modifications[2]	×	✓*	×	✓	✓	✓
Hardware requirements for QKD infrastructure	×	×	✓	✓	✓	×
Interoperation with unmodified TLS 1.3 peers[3]	✓	✓*	✓*	✓*	×	×
Mobility without continuous wide-area access to remote QKD infrastructure[4]	✓	✓	✓*	✓*	✓*	✓
QR goal for HNDL confidentiality[5]	×	✓	✓	✓	✓	✓*

Backup: Clean Protocol-Size Snapshot

Variant	Wire bytes	TLS hs. bytes	Packets	Valid runs
TLS 1.3 cert	2790 B	1534 B	15	100/100
TLS 1.3 PSK	1701 B	491 B	15	100/100
Hybrid TLS	5064 B	3808 B	15	100/100
QKD-PSK Importer	3335 B	1563 B	23	100/100
TLS-QKD	2372 B	1015 B	16	100/100
QKD-KEM	5223 B	3769 B	18	100/100
KEMTLS	11717 B	9435 B	18	100/100

Backup: PSK-Focused View: KFS-Relevant Comparison

Both paths rely on pre-provisioned key material.

Metric	TLS 1.3 PSK	QKD-PSK Importer	Ratio
TLS norm. p50	0.93 ms	3.91 ms	4.21x
Setup p50	0.04 ms	0.05 ms	1.18x
End-to-end p50	4.20 ms	4.73 ms	1.13x
Wire bytes p50	1701 B	3335 B	1.96x

Interpretation

local setup is small
visible TLS cost delta
wire footprint grows
KFS-relevant baseline

Use current validated exports only; do not generalize to all PSK implementations.

Clean Condition: Full Trace and Handshake Footprint

Detailed p50 view from 100 clean runs. Timing and byte factors are relative to TLS 1.3 Cert.

Variant	TLS / Cipher / Group	Auth	TLS Norm p50	Setup p50	E2E p50	HS bytes	Record bytes	Wire bytes	IP bytes	Pkts	CH / SH / EE	Cert / CV / Fin	Srv Flight
TLS 1.3 Cert	TLSv1.3 TLS_AES_256_GCM_SHA384 x25519	cert	1.53 ms 1.00x	-	2.02 ms 1.00x	1 534 B 1.00x	1 731 B	2 790 B 1.00x	2 580 B 1.00x	15	205 118 2	853 260 96	1 329 B
TLS 1.3 PSK	TLSv1.3 TLS_AES_256_GCM_SHA384 x25519	psk	0.93 ms 0.61x	0.04 ms	4.20 ms 2.08x	491 B 0.32x	646 B	1 701 B 0.61x	1 491 B 0.58x	15	269 124 2	0 0 96	222 B
Hybrid TLS	TLSv1.3 TLS_AES_256_GCM_SHA384 X25519MLKEM768	cert	1.65 ms 1.08x	-	2.17 ms 1.08x	3 808 B 2.48x	4 005 B	5 064 B 1.82x	4 854 B 1.88x	15	1391 1206 2	853 260 96	2 417 B
QKD-PSK Importer	TLSv1.3 TLS13-AES256-GCM-SHA384 X25519	cert+ext PSK	3.91 ms 2.55x	0.05 ms	4.73 ms 2.34x	1 563 B 1.02x	1 758 B	3 335 B 1.20x	3 013 B 1.17x	23	248 96 2	861 260 96	1 315 B
TLS-QKD	TLSv1_3 TLS13_AES_256_GCM_SHA384 -	tls_qkd	2.56 ms 1.67x	149.81 ms	153.48 ms 75.91x	1 015 B 0.66x	1 107 B	2 372 B 0.85x	2 148 B 0.83x	16	278 203 6	405 75 48	737 B
QKD-KEM	TLSv1.3 TLS_AES_256_GCM_SHA384 qkd_kyber768	cert+QKD-KEM	8.60 ms 5.61x	200.46 ms	209.88 ms 103.80x	3 769 B 2.46x	3 966 B	5 223 B 1.87x	4 971 B 1.93x	18	1369 1190 2	852 260 96	2 400 B
KEMTLS	TLSv1.3 TLS13_AES_256_GCM_SHA384 Kyber512	kemtls	0.64 ms 0.42x	-	1.36 ms 0.67x	9 435 B 6.15x	10 458 B	11 717 B 4.20x	11 465 B 4.44x	18	1224 854 6	7351 0 0	8 211 B

Legend: HS = decoded TLS handshake bytes. Wire = selected TCP stream frame bytes. CH/SH/EE = ClientHello / ServerHello / EncryptedExtensions. CV = CertificateVerify.

Backup: KFS Lifecycle: Fuel, Roam, Refill

Scope boundary

The PoC validates key loading, local cache behavior and TLS use. Full lifecycle enforcement remains future work.

Backup: KFS Android PoC - Implementation

- **Alice/KFS:** Local HTTP provisioning prototype (serves normalized key capsules)
- **Bob/RBC:** TLS 1.3 external-PSK server; matches `key_id`, sends `PROTECTED_PAYLOAD_OK`
- **Charlie:** Pixel 2 / Android 11 key-consuming endpoint, not a QKD node
- **Kotlin**: UI, Bob/RBC settings, capsule fetch and app-private key capsule cache
- **Java Native Interface (JNI)**: Passes `key_id` and 32 byes / 256-bit PSK from Kotlin to native C
- **Native OpenSSL:** TLS 1.3 PSK-only handshake, no certificate authentication
- **Scope**: Validates mobile key consumption and TLS PSK use, not production KFS security
- **Boundary**: Local demo provisioning and prototype cache

Backup: Standards and Role Terms

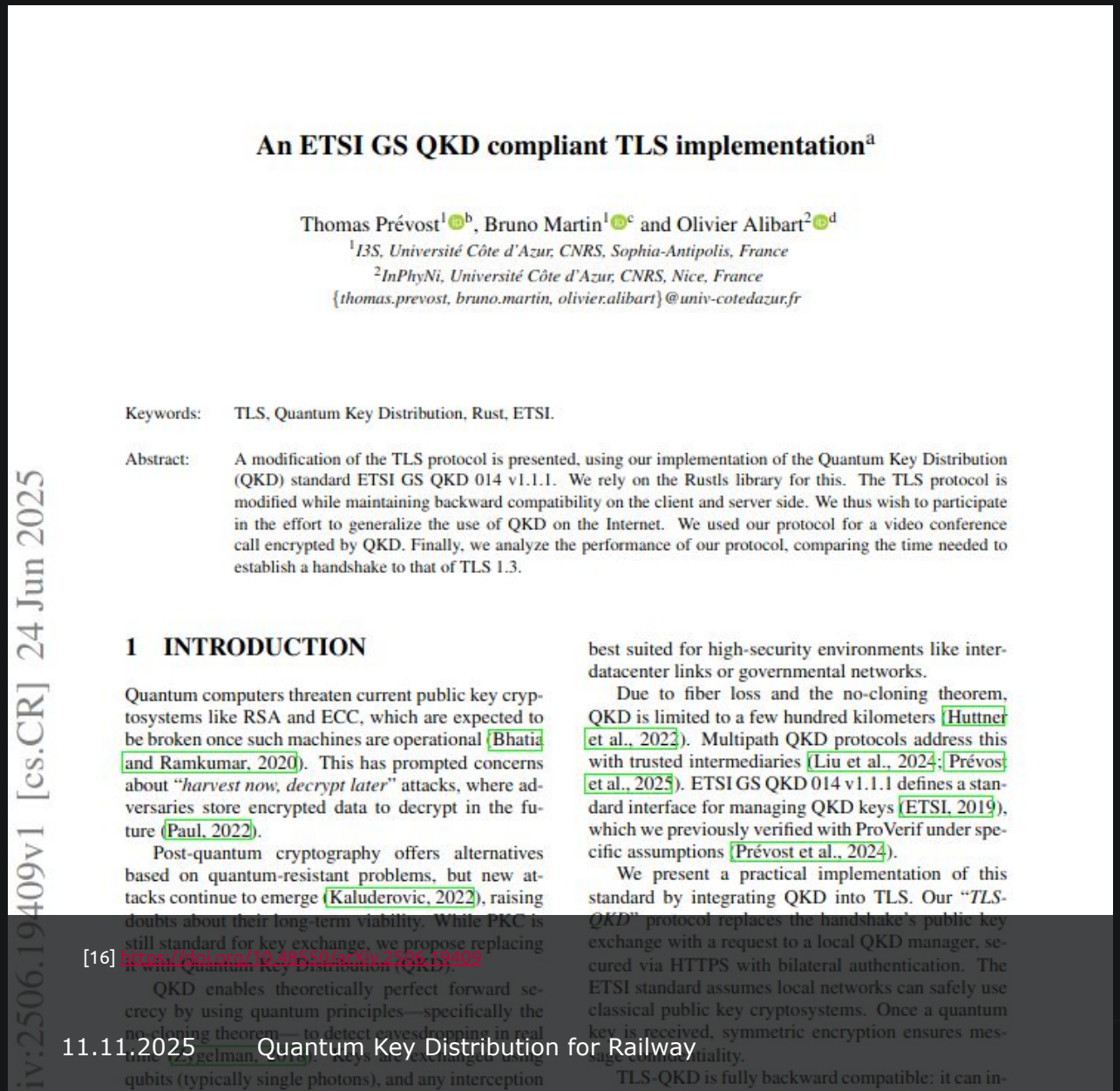
Reference	Role in this thesis
TLS 1.3 / RFC 8446	baseline protected channel and PSK reference
ETSI GS QKD 014	SAE-to-KME key delivery API inspiration
ITU-T QKDN vocabulary	role terminology for architecture discussion
KFS mapping	secure site supplies key material, train consumes capsules, RBC looks up records

Backup: Validation and Scope Boundaries

- **QKD-related paths are simulated or fixture-backed in the inspected repositories.**
- Test certificates, PSK files and key capsules are prototype fixtures.
- External mechanisms are reused implementations, not new cryptographic designs.
- Timing results describe local implementation paths, not universal protocol overhead.
- Android PoC validates core TLS 1.3 PSK key consumption, not full lifecycle policy.
- **No live optical QKD hardware measurement is claimed in the current snapshot.**

TLS Post-Quantum Approaches (2)

[16]



An ETSI GS QKD compliant TLS implementation[a]

Thomas Prévost[1][b], Bruno Martin[1][c] and Olivier Alibart[2][d]
[1]I3S, Université Côte d'Azur, CNRS, Sophia-Antipolis, France
[2]InPhyNi, Université Côte d'Azur, CNRS, Nice, France
{thomas.prevost, bruno.martin, olivier.alibart}@univ-cotedazur.fr

Keywords: TLS, Quantum Key Distribution, Rust, ETSI.

Abstract: A modification of the TLS protocol is presented, using our implementation of the Quantum Key Distribution (QKD) standard ETSI GS QKD 014 v1.1.1. We rely on the RustIs library for this. The TLS protocol is modified while maintaining backward compatibility on the client and server side. We thus wish to participate in the effort to generalize the use of QKD on the Internet. We used our protocol for a video conference call encrypted by QKD. Finally, we analyze the performance of our protocol, comparing the time needed to establish a handshake to that of TLS 1.3.

1 INTRODUCTION

Quantum computers threaten current public key cryptosystems like RSA and ECC, which are expected to be broken once such machines are operational (Bhatia and Ramkumar, 2020). This has prompted concerns about "*harvest now, decrypt later*" attacks, where adversaries store encrypted data to decrypt in the future (Paul, 2022).

Post-quantum cryptography offers alternatives based on quantum-resistant problems, but new attacks continue to emerge (Kaluderovic, 2022), raising doubts about their long-term viability. While PKC is still standard for key exchange, we propose replacing it with Quantum Key Distribution (QKD).

QKD enables theoretically perfect forward secrecy by using quantum principles—specifically the no-cloning theorem—to detect eavesdropping in real time (Zygelman, 2018). Keys are exchanged using qubits (typically single photons), and any interception

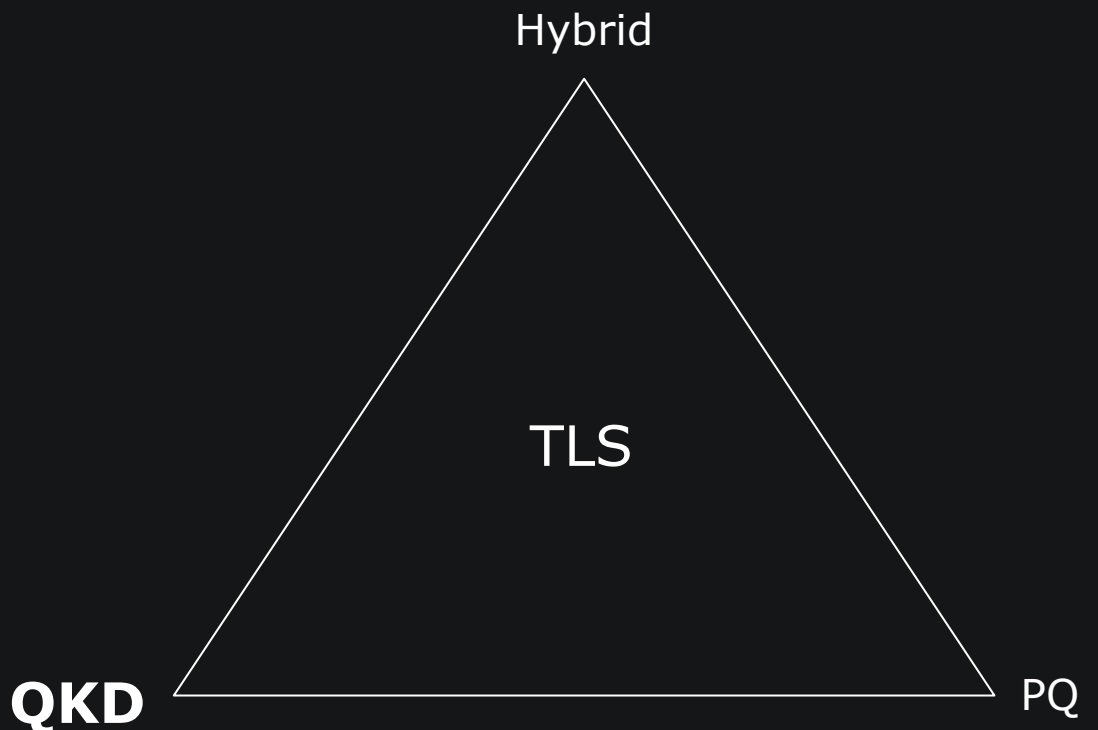
best suited for high-security environments like inter-datacenter links or governmental networks.

Due to fiber loss and the no-cloning theorem, QKD is limited to a few hundred kilometers (Huttner et al., 2022). Multipath QKD protocols address this with trusted intermediaries (Liu et al., 2024; Prévost et al., 2025). ETSI GS QKD 014 v1.1.1 defines a standard interface for managing QKD keys (ETSI, 2019), which we previously verified with ProVerif under specific assumptions (Prévost et al., 2024).

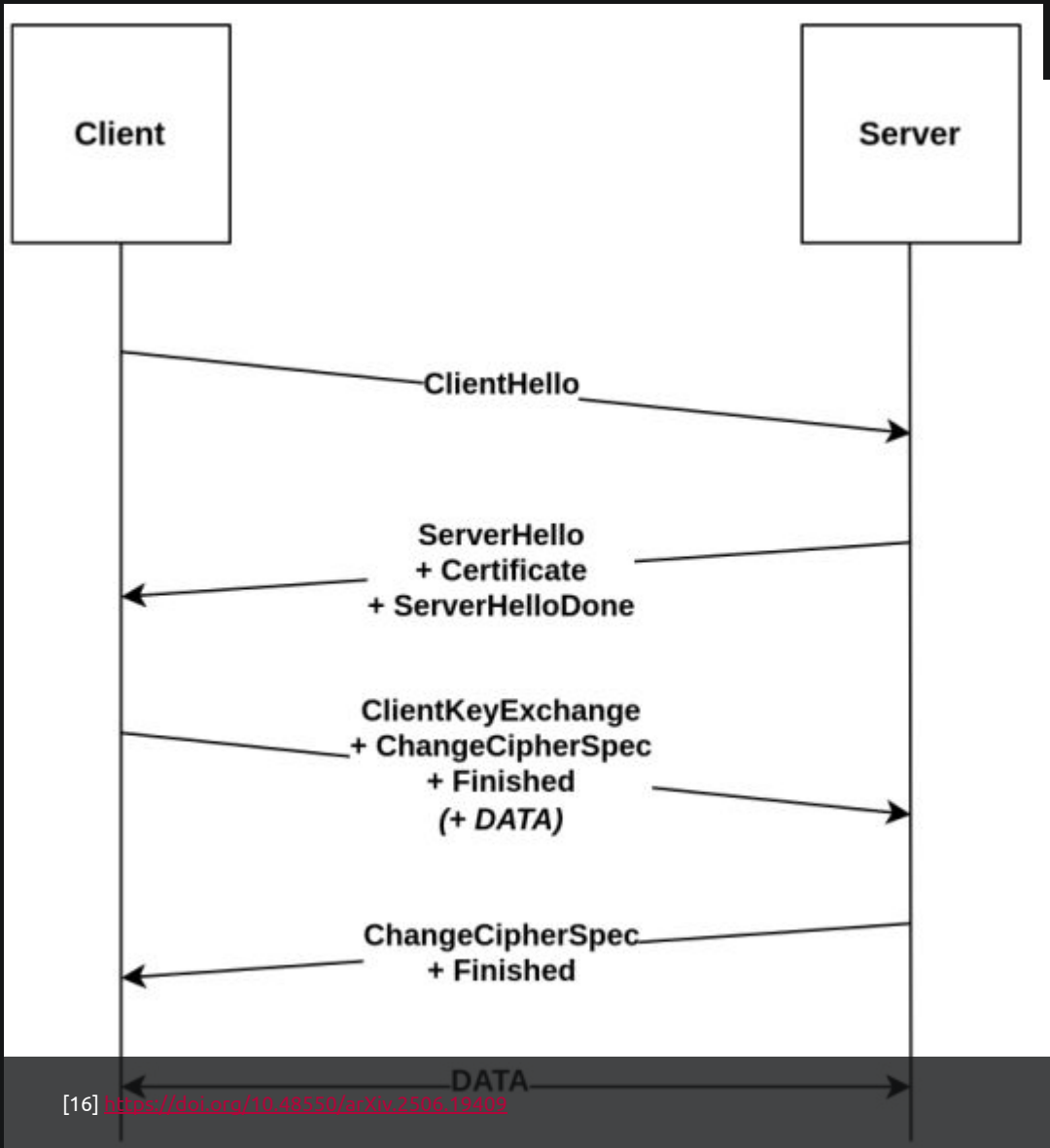
We present a practical implementation of this standard by integrating QKD into TLS. Our "*TLS-QKD*" protocol replaces the handshake's public key exchange with a request to a local QKD manager, secured via HTTPS with bilateral authentication. The ETSI standard assumes local networks can safely use classical public key cryptosystems. Once a quantum key is received, symmetric encryption ensures message confidentiality.

TLS-QKD is fully backward compatible: it can in-

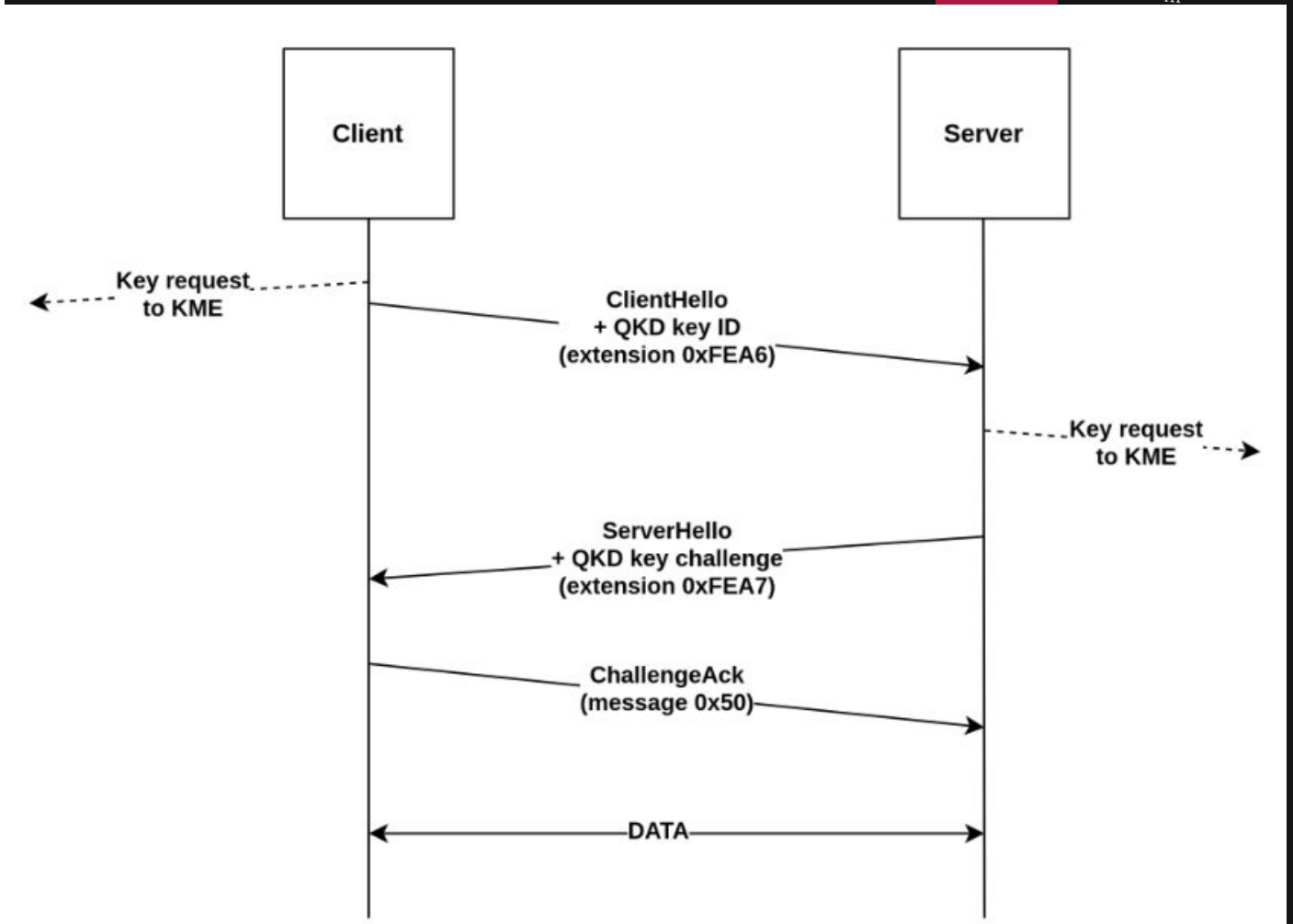
[16] https://doi.org/10.48550/arXiv.2506.19409



“Classical” TLS vs. TLS-QKD



(b) Handshake analysis TLS 1.3.



(a) Handshake on TLS-QKD.

[16] https://doi.org/10.48550/arXiv.2506.19409

TLS Post-Quantum Approaches (3)

[17]

Post-Quantum TLS Without Handshake Signatures

Full version, March 15, 2022

Peter Schwabe
Max Planck Institute for Security and Privacy & Radboud University
peter@cryptojedi.org

Douglas Stebila
University of Waterloo
dstebila@uwaterloo.ca

Thom Wiggers
Radboud University
thom@thomwiggers.nl

ABSTRACT

We present KEMTLS, an alternative to the TLS 1.3 handshake that uses key-encapsulation mechanisms (KEMs) instead of signatures for server authentication. Among existing post-quantum candidates, signature schemes generally have larger public key/signature sizes compared to the public key/ciphertext sizes of KEMs: by using an IND-CCA-secure KEM for server authentication in post-quantum TLS, we obtain multiple benefits. A size-optimized post-quantum instantiation of KEMTLS requires less than half the bandwidth of a size-optimized post-quantum instantiation of TLS 1.3. In a speed-optimized instantiation, KEMTLS reduces the amount of server CPU cycles by almost 90% compared to TLS 1.3, while at the same time reducing communication size, reducing the time until the client can start sending encrypted application data, and eliminating code for signatures from the server's trusted code base.

Update: in Appendix F we present updated measurements based on the NIST standardization project's round-3 candidate schemes.

CCS CONCEPTS

• **Security and privacy** → **Security protocols**; **Web protocol security**; **Public key encryption**.

KEYWORDS

Post-quantum cryptography; key-encapsulation mechanisms; Transport Layer Security; NIST PQC

ACM Reference Format:
Peter Schwabe, Douglas Stebila, and Thom Wiggers. 2020. Post-Quantum TLS Without Handshake Signatures: Full version, March 15, 2022. In *2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA,

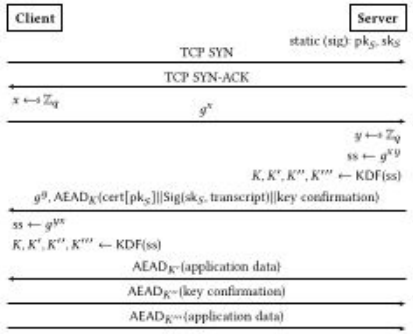


Figure 1: High-level overview of TLS 1.3, using signatures for server authentication.

RSA or elliptic-curve signatures. Public keys for the signatures are embedded in certificates and transmitted during the handshake. Figure 1 gives a high-level overview of the TLS 1.3 protocol, focusing on the signed-Diffie–Hellman aspect of the handshake.

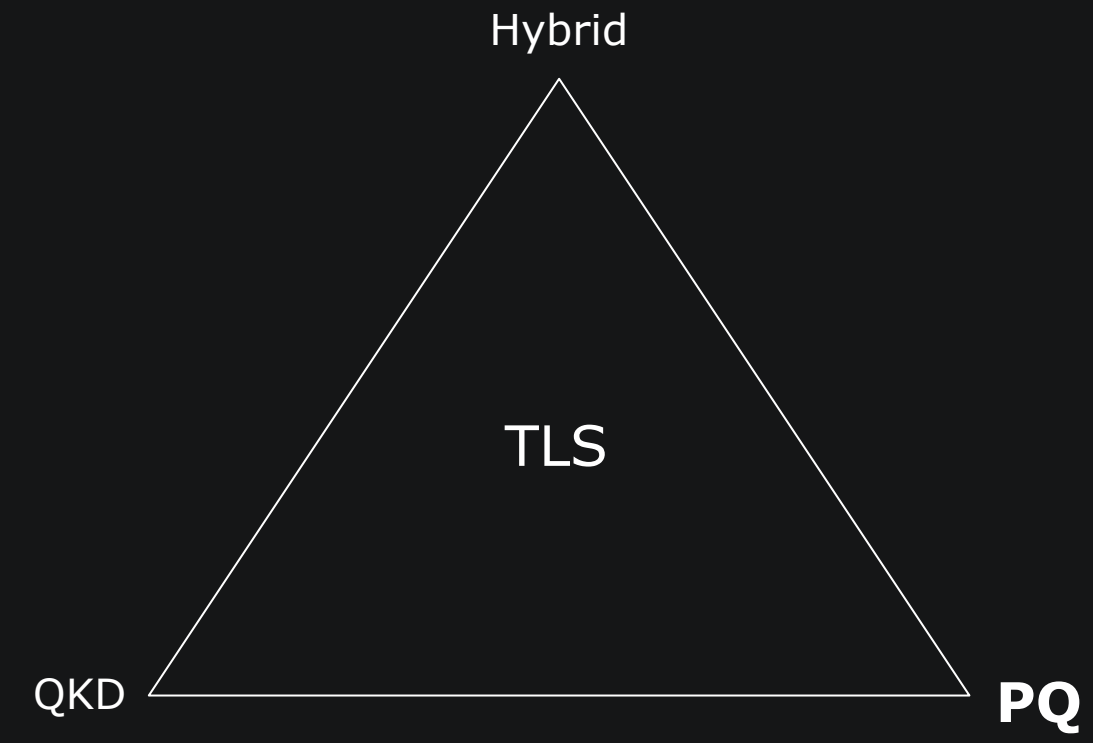
Preparing for post-quantum TLS. There have been many experiments and much research in the past five years on moving the TLS ecosystem to post-quantum cryptography. Most of the work has focused on adding post-quantum key exchange to TLS, usually in the context of so-called "hybrid" key exchange that uses both a post-quantum algorithm and a traditional (usually elliptic curve) algorithm, beginning with an experimental demonstration in 2015 of ring-LWE-based key exchange in TLS 1.2 [21].

Public experiments by industry started in 2016 with the CECPQ1 experiment by Google [76], combining X25519 ECDH [9] with NewHope lattice-based key exchange [2] in the TLS 1.2 handshake. A CECPQ2 followup experiment with TLS 1.3 was announced in late [illegible] is currently being run by Google using a combination of X25519 and the lattice-based scheme NTRU-HRSS [54, 55], and by Cloudflare using X25519/NTRU-HRSS and X25519 together

[17] https://doi.org/10.1145/3372297.3423350

1 INTRODUCTION

The Transport Layer Security (TLS) protocol is possibly one of the [illegible] way to transfer web pages [92], but is also to secure communications



TLS Post-Quantum Approaches (4)

[18]

QKD-KEM: Hybrid QKD Integration into TLS with OpenSSL Providers

Javier Blanco-Romero
Department of Telematic Engineering
Universidad Carlos III de Madrid
Leganés, Madrid, Spain
frblanco@pa.uc3m.es

Pedro Otero García
atlanTTic Research Center (IC Lab)
University of Vigo
Spain
pedro.otero@det.uvigo.es

Daniel Sobral-Blanco
Department of Telematic Engineering
Universidad Carlos III de Madrid
Leganés, Madrid, Spain
dsobral@pa.uc3m.es

Florina Almenares Mendoza
Department of Telematic Engineering
Universidad Carlos III de Madrid
Leganés, Madrid, Spain
florina@it.uc3m.es

Ana Fernández Vilas
atlanTTic Research Center (IC Lab)
University of Vigo (Spain)
avilas@det.uvigo.es

Rebeca P. Díaz-Redondo
atlanTTic Research Center (IC Lab)
University of Vigo (Spain)
rebeca@det.uvigo.es

***Abstract*—Quantum Key Distribution (QKD) promises information-theoretic security, yet integrating QKD into existing protocols like TLS remains challenging due to its fundamentally different operational model. In this paper, we propose a hybrid QKD-KEM protocol with two distinct integration approaches: a client-initiated flow compatible with both ETSI 004 and 014 specifications, and a server-initiated flow similar to existing work but limited to stateless ETSI 014 APIs. Unlike previous implementations, our work specifically addresses the integration of stateful QKD key exchange protocols (ETSI 004) which is essential for production QKD networks but has remained largely unexplored. By adapting OpenSSL's provider infrastructure to accommodate QKD's pre-distributed key model, we maintain compatibility with current TLS implementations while offering dual layers of security. Performance evaluations demonstrate the feasibility of our hybrid scheme with acceptable overhead, showing that robust security against quantum threats is achievable while addressing the unique requirements of different QKD API specifications.**

***Index Terms*—Post–Quantum Cryptography, PQC, QKD, TLS, OpenSSL**

[18] https://doi.org/10.48550/arXiv.2503.07196

I. Introduction

The Transport Layer Security (TLS) protocol is the cor-

introducing significant security and performance improve-

Recent standardization efforts related to TLS 1.3 have established a framework for hybrid key exchange that combines traditional and post-quantum cryptography [5]. Their approach demonstrates how to achieve security through the concatenation of shared secrets from different key exchange methods, maintaining protection as long as at least one component remains unbroken. This design principle is relevant for QKD deployments where extra security is needed: while QKD provides information-theoretic security, supplementing it with post-quantum cryptography can provide additional protection against implementation vulnerabilities or operational compromises in the QKD system.

Our work explores QKD integration into TLS using OpenSSL's provider infrastructure, encapsulating both PQC shared secrets and QKD key identifiers into a single KEM operation. Unlike previous implementations, we specifically address the integration of stateful QKD protocols through ETSI 004 API. Our implementation maintains compatibility with existing TLS while providing dual security layers, demonstrating a viable pathway for quantum-safe TLS with acceptable performance overhead.

The remainder of this paper is organized as follows. Sec- [illegible] an overview of the integration of QKD with secure communication protocols, discussing the related