Photons Are Perfect, Protocols Are Not: Cracking QKD BBM92 via The Internet

Kashyap Thimmaraju[1 [0009−0006−1507−3896]], Max Henri Julian Hiort[1 [0009−0002−4322−9940]], Darshit Suratwala[1 [0000−0002−7602−9702]], Elham Amini[1 [0000−0002−4580−4243]], and Jean-Pierre Seifert[1 [0000−0002−5372−4825]]

Technische Universität Berlin,
Berlin, Germany
{kashyap.thimmaraju, jean-pierre.seifert}@tu-berlin.de

Abstract. Quantum Key Distribution (QKD) promises information-theoretic secrecy by leveraging quantum physics, yet practical deployments depend heavily on a classical, packet-switched control channel for timing, synchronization, and key management. Surprisingly, this critical classical component has been largely overlooked from a network- and system-security perspective. In this paper, we present a comprehensive security evaluation of the classical channel in a commercial-grade entangled-photon BBM92 QKD deployment. We introduce Pickle-TCP, a transparent, QKD-protocol-aware fault-injection proxy that manipulates control-plane messages. Using only intercepted network traffic, we reverse-engineer the proprietary QKD protocol and uncover zero-day exploitable vulnerabilities in authentication, synchronization, and control-message handling. Leveraging this understanding, we demonstrate three concrete and repeatable attacks that disrupt authentication, silently desynchronize key pools, and halt key generation in our deployment without triggering operator-visible alarms. As a practical countermeasure, we evaluated a post-quantum-secure tunnel for the classical channel using WireGuard and Rosenpass and measured an approximately 6.6% key generation throughput overhead. While our evaluation examines a single commercial BBM92 implementation, it shows that under-specified classical-channel design choices can create exploitable security failures in production QKD systems. More broadly, our findings highlight the gap between quantum-security theory and the engineering realities of networked control planes, and establish a network-security case study for entanglement-based QKD systems.

Keywords: Symmetric-key Cryptography · Quantum Key Distribution · Network Security · BBM92 · QKD Key Pool Attacks · ETSI · Remote Code Execution

1 Introduction

As quantum computing advances threaten to break conventional public-key cryptography within the next decade, governments and industries worldwide are rac-

ing to secure critical communications infrastructure [5]. Quantum Key Distribution (QKD) has emerged as a promising solution: unlike classical cryptography that relies on computational hardness assumptions, QKD leverages fundamental quantum mechanics principles, the no-cloning theorem and observer effect [3,38], to provide information-theoretic secrecy. Any eavesdropping attempt inherently disturbs quantum states, guaranteeing detection of adversaries. Charles H. Bennett and Gilles Brassard were awarded the 2025 ACM Turing Award for their foundational contributions that include QKD [1].

From 2020 to 2025, the QKD ecosystem has expanded dramatically: at least 34 distinct testbeds, commercial networks, and projects are now active globally, 21 in Europe, 7 in the United States, and 6 across Asia (China, India, Japan, Singapore, and South Korea) [45]. At least 16 vendors now offer QKD systems targeting defense, financial services, telecommunications, critical infrastructure, healthcare, and data centers, with approximately five specializing in entanglement-based BBM92 implementations [45]. This growth is fueled by substantial public funding initiatives: the European Quantum Communication Infrastructure (EuroQCI) project aims to build quantum communication infrastructure with 90 million Euros in investment [17], while the US National Science Foundation has allocated $231.15 million for quantum information science research in 2026 [34]. Recognizing this maturation, standardization bodies have entered the field: ISO published ISO/IEC 23837:2023 for QKD component and security requirements [21], and ETSI has developed comprehensive QKD profiles [16, 19], signaling that integration and interoperability are now central concerns for mainstream cryptographic systems.

In entanglement-based protocols like BBM92 (see Figure 1), pairs of entangled photons are generated, and their quantum correlations: established through polarization and basis measurements; allow two parties (Alice and Bob) to create shared cryptographic keys. Any attempt at eavesdropping disturbs these correlations, providing guaranteed detection of adversaries [3, 42]. The quantum channel exclusively carries photons used to establish correlated measurement outcomes, while the classical channel transmits necessary control information for clock synchronization, basis selection announcements, error correction, and key distillation processes [9]. Hence, the security guarantees of QKD rely critically not only on quantum principles but also on the classical control plane's robustness.

While extensive research has probed the quantum-optical vulnerabilities in QKD systems, considerably less attention has been directed toward securing the classical communication channel, despite its familiarity to network adversaries. A comprehensive security review by the German Federal Office for Information Security (BSI) [20] highlights 49 distinct attack paths across diverse categories, including Calibration [31], Detector-blinding [28,30], Photon-number splitting [26, 29], and Trojan-horse attacks [22], among others. Yet, nearly all documented vulnerabilities target quantum hardware directly. The remaining attacks involve (cache) side-channels on QKD systems [2, 6, 24, 36] requiring the attacker to have physical access to the systems.

In contrast, the classical channel, though crucial for protocol operation, remains underexplored in QKD security research. Existing work has investigated aspects of the classical-quantum interplay [32,33], authentication weaknesses [23, 35], reconciliation side-channels [25], and denial-of-service conditions [7]. However, no comprehensive network-centric security evaluation targeting the classical control plane has been conducted to date.

This gap is particularly concerning given that established protocols such as BB84, BBM92, and ETSI standards explicitly indicate that the classical channel may remain unencrypted and sometimes unauthenticated [16]. Furthermore, QKD systems face a fundamental bootstrapping challenge: they require authenticated classical channels to maintain information-theoretic security, yet generating the initial authentication keys presents a circular dependency. While post-quantum cryptography (PQC) offers a practical solution to this cold-start problem, the integration of PQC-based authentication with QKD systems remains largely unexplored in practice.

This paper presents the first rigorous *network-security evaluation focused solely on the TCP communication of the classical channel* of a commercial entangled-photon QKD system. By reverse-engineering the TCP communication, we identified that the TCP payload used Python Pickle for binary (de)serialization. To systematically analyze the TCP communication of our QKD system, we decided to develop Pickle-TCP, a transparent, application-aware proxy for intercepting and manipulating Python-Pickle-serialized TCP control streams. Pickle-TCP enables precise manipulation of classical-channel messages without affecting the quantum channel. Leveraging this capability, we identified and empirically demonstrated three concrete attacks: (a) **Authentication compromise:** exploiting a flaw in the HMAC-based authentication protocol to obtain oracle-signed messages, and in conjunction with Pickle to obtain remote code execution on both QKD servers that hosted the QKD generated key and the key used to sign the classical channel (Fig. 2); (b) **Silent desynchronization:** suppressing key-identifier frames to disrupt synchronization and create mismatched key pools, with Bob's attack-window key-generation rate dropping by a mean 99.3% across ten runs (Table. 2); and (c) **Stalling key generation:** delaying classical exchanges of measurement-related data, where 250 ms was the smallest sustained delay that collapsed key generation on both sides in our setup (Fig. 5 and Table 2).

Our empirical evidence comes from one commercial BBM92 entanglement-photon based QKD vendor, out of five worldwide commercial vendors. Accordingly, our claim is not that all QKD implementations are vulnerable in the same way, but that security-critical classical-channel choices remain under-specified in current practice and can create exploitable failures. In particular, ETSI GS QKD 014 standardizes the northbound key-delivery API between KMEs and applications, ETSI GS QKD 08 specifies the QKD Module Security, while ETSI GS QKD 016 defines a protection profile for QKD modules from the physical implementation to the output of final secret keys; neither document prescribes a concrete peer-to-peer control-plane protocol, authentication handshake, or key-

identifier synchronization mechanism for the proprietary classical channel we analyze [13, 16].

As a practical countermeasure, we integrated WireGuard [8] with Rosenpass [39] to provide post-quantum-secure tunneling for the classical channel. Across ten runs, the tunneled setup achieved a mean run-average key-generation rate of 0.808 keys/s while preserving the 2 keys/s peak rate, and it eliminates content-aware manipulation and injection by an on-path adversary, but does not by itself prevent pure delay/drop availability attacks. To the best of our knowledge, this is the first empirical validation of PQC+QKD integration for securing the classical control plane.

This paper makes the following contributions.

1. Reverse-engineering and traffic analysis of a commercial BBM92 classical control protocol
2. Development of Pickle-TCP, an open-source transparent fault-injection proxy available at: `https://github.com/MaxHTu/tcp_proxy`
3. Demonstration of three attack vectors compromising authentication, synchronization, and availability
4. First empirical evaluation of PQC+QKD integration (using Rosenpass), measuring approximately 6.6% overhead in key generation throughput
5. Identification of critical gaps in ETSI QKD standards and broader lessons for classical-channel security in QKD deployments

Ethical Disclosure. Experiments were performed on hardware we own in an isolated laboratory environment. No production networks, public optical fibres, or third-party services were probed or disrupted. We informed our QKD vendor about our findings confidentially and they immediately acknowledged and patched their codebase.

Paper Structure. The paper proceeds as follows: Section 2 provides necessary context, Section 3 details our threat model, Section 4 explains classical channel interception methods, and Section 5 introduces Pickle-TCP. Sections 6 and 7 discuss our attacks, Section 8 proposes a secure classical channel solution, and Sections 9, 10, and 11 provide discussions, related work, and concluding remarks, respectively.

2 What Is QKD? And How Does It Work?

The BBM92 protocol, proposed by Bennett, Brassard, and Mermin in 1992, is an entanglement-based quantum key distribution scheme that leverages the non-local correlations of entangled photon pairs to establish symmetric cryptographic keys between two nodes, Alice and Bob [3]. Unlike prepare-and-measure schemes such as BB84, BBM92 eliminates the need for trusted state preparation by generating entangled photon pairs at a central (laser-based) source and distributes one photon from each pair to Alice and the other to Bob via an optical channel that is typically termed the quantum channel. Both parties independently

and randomly choose measurement bases (e.g., rectilinear or diagonal) and perform polarization measurements. When their basis choices are compatible, their outcomes are perfectly (or anti-) correlated due to quantum entanglement [42].

In the post-processing phase, the correlated outcomes are then sifted through a general purpose computer on Alice and Bob. They then compare their basis choices over a classical channel (e.g., public network such as the Internet) and retain only those bits where their bases matched. Note that the actual measurement information is not exchanged between them. This process forms the raw symmetric key. However, due to noise and potential eavesdropping, discrepancies may exist between Alice's and Bob's keys. Therefore, they engage in error correction (reconciliation) to align their bit strings, followed by privacy amplification to reduce Eve's potential knowledge [27]. Throughout these post-processing steps, authentication protocols (typically using pre-shared short keys and message authentication codes, MACs) are used to ensure message integrity and prevent man-in-the-middle attacks over the classical channel [38].

Concretely, the classical channel in a BBM92 deployment is not a mere side channel for logging or coordination. It carries the protocol-critical exchanges that let Alice and Bob sift detections, maintain timing alignment, reconcile measurement outcomes, trigger privacy amplification, and agree on which distilled keys are ready for downstream use. If these exchanges are forged, suppressed, or delayed, the quantum layer may still deliver photon correlations while the end-to-end key-establishment process silently fails, diverges, or becomes attacker-controlled.

The final output is a symmetric key of reduced length, depending on the initial raw key rate, quantum bit error rate (QBER), and efficiency of post-processing. For example, starting from a 1,000 bit raw key with a QBER of 3 percent, the final secure key might be 256–700 bits long after error correction and privacy amplification [27, 37]. The link between the quantum and classical channels is thus critical: while entanglement enables secure key generation, the classical channel is essential for coordination, sifting, reconciliation, authentication, and final key distillation [37, 42]. This observation is not specific to BBM92 alone; any QKD deployment that relies on classical coordination for synchronization, reconciliation, and key management can inherit similar control-plane risks.

ETSI QKD Standards

The European Telecommunications Standards Institute (ETSI) Industry Specification Group on QKD specifies in ETSI GS QKD 014 [13] that a deployment comprises distinct entities: Quantum Key Distribution Modules (QKDMs), Key Management Entities (KMEs), and Secure Application Entities (SAEs). QKDMs interface with quantum-optical hardware to generate raw keys; after classical post-processing (error correction, privacy amplification), keys are transferred to KMEs over an internal interface. KMEs maintain key stores/buffers and enforce key-lifetime policies. The classical control channel, often implemented over Ethernet/IP, handles timing, reconciliation of measurements, orchestration of key

transfer, and authenticated message exchange. ETSI defines an Application Interface in GS QKD 004 [14] and a REST key-delivery API in GS QKD 014 [13] between KMEs and applications (SAEs).

Once raw keys have been distilled, they are moved from QKDMs to KMEs over an internal interface. KMEs store them and apply policy. Applications obtain keys via standardized interfaces: GS QKD 004 specifies a technology-neutral set of service operations and parameters for requesting and acknowledging keys from KMEs, while GS QKD 014 defines HTTP/JSON endpoints for key delivery; deployments are expected to protect this interface (e.g., via TLS/HTTPS). These interfaces enable interoperable delivery of keys to SAEs such as VPN endpoints, TLS libraries, or databases.

ETSI also provides supplementary specifications for deployment and assurance in GS QKD 012 [12] and GS QKD 016 [16] offers a Common-Criteria protection profile for a pair of prepare-and-measure QKDMs connected by a point-to-point link. Security-proofs are described in GS QKD 005 [10] and module/component specs in GS QKD 008 [9] and 011 [11].

3 Attacker Model

We assume two honest QKD nodes, **Alice** and **Bob** who are connected via the quantum (optical) and classical channels respectively as shown in Figure 1 and described in Section 2. Eve the adversary **cannot** tamper with or measure individual photons without perturbing the entanglement resulting in no-keys being agreed upon. Instead, we assume Eve, is present on the Internet path between Alice and Bob, e.g., a malicious employee at an Internet Service Provider (ISP), making Eve an *on-path* attacker.

Eve's ultimate objective is to get access to the keys being generated by the QKD system, however, degrading or subverting final key correctness, confidentiality and availability while remaining undetected by built-in monitoring are other objectives too. Eve has no physical access to Alice or Bob's hardware, does not have access to the keys used to authenticate the classical channel communication, and cannot break modern (quantum or classical) cryptography. However, Eve *does* understand the public elements of the control protocol by reversing the state machine from TCP/UDP streams and can deploy commodity middleboxes at any point between the peers (typical for a malicious ISP who can redirect specific network flows for further analysis). For example, Eve can: (a) read all packets (passive eavesdropping); (b) delay, drop, duplicate or reorder packets; (c) inject or modify payload bytes in-flight; and (d) open new transport socket connections to its peers.

4 Decoding the Workings of the Network Traffic

We obtained a commercial entangled-photon BB92-based QKD system as part of a government research project to evaluate potential side-channels in such systems. After we had the technicians install the system, we had several questions

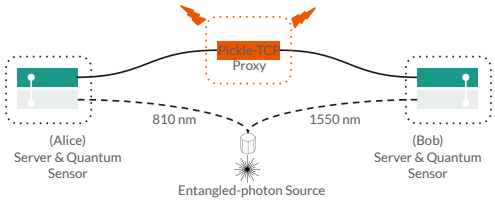

Fig. 1: Topology of our QKD security testing network.

on how the entire system works. Given, our background in network systems security, we were interested in the classical channel communication. Therefore, we analyzed and dissected the network traffic several times. In the following we provide a distilled analysis from a ten minute packet capture of the network traffic from a fresh start of the QKD nodes to stable key generation. We examine the classical channel's message types, sequencing, authentication mechanisms, and timing dependencies, insights we subsequently exploit in our attacks (Sections 6 and 7).

4.1 TCP Streams & Authentication

On the classical channel, using Wireshark we first observed that Alice and Bob initiate two separate TCP connections: one where Alice connects to Bob on port 5000 and another where Bob connects to Alice on port 5000. So each node uses a dedicated *one-way* TCP connection to send data, note that the other end of each TCP connection merely sends ACKs back, no data is sent. Immediately after the three-way TCP handshake we observed the authentication protocol which is shown in Figure 2.

After Alice and Bob complete their 3-way TCP handshake (with Bob being the server in this case as he is listening on port 5000), Bob sends Alice a `CHALLENGE` and `RANDOM_BYTES`. Alice uses her pre-shared key (K) to sign `RANDOM_BYTES` and sends BOB `HMAC(RANDOM_BYTES)`. If `HMAC(RANDOM_BYTES)` matches Bob's `HMAC(RANDOM_BYTES)`, then Bob responds back with `WELCOME`. Next, Alice repeats the steps that Bob took starting with the `CHALLENGE` and `RANDOM_BYTES` and ending with the `WELCOME`. This completes the authentication.

In multiple observations of the authentication protocol, we noticed that a successful authentication ends with Alice sending Bob a fixed 32-byte field, after which Bob sends the Welcome message. Observing a constant 32-byte suffix, typical for SHA-256's length and confirming it on sample messages, we concluded the message is $\mathrm{HMAC}_{\text{SHA-256}}(K, \mathrm{RANDOM\_BYTES})$.

A Hash-based Message Authentication Code (HMAC) is a keyed integrity check:

$$\mathrm{HMAC}_K(m) = H\big((K \oplus \mathrm{opad}) \,\|\, H((K \oplus \mathrm{ipad}) \,\|\, m)\big)$$

where H is a cryptographic hash, in this case SHA-256. It lets the receiver check that the tag was produced by someone who knows K and that the message m was not modified in transit.

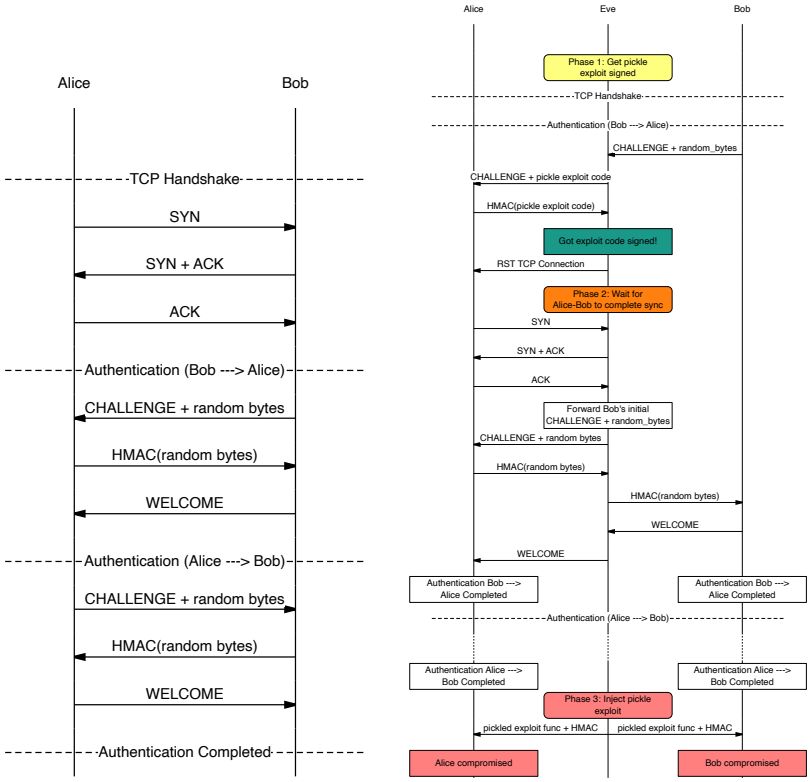

Fig. 2: Left: Message sequence pattern illustrating the QKD authentication protocol. Each node independently challenges its peer with random bytes, expecting a valid HMAC signature in response. Right: Step-by-step exploitation of the authentication vulnerability, where the attacker (Eve) tricks Alice into signing a malicious payload.

4.2 Decoding Pickled QKD messages

Despite the ability to transmit messages in plain-text (recall Section 2), we observed the TCP payload after the authentication to be encoded, e.g., there were no clear-text JSON objects. After researching best possibilities to decode the payload stream, we identified the payload to be encoded as length-prefixed Python *Pickle* objects over persistent TCP sockets. Each object encoded a high-level action from the source service to the target service. From the ten minute capture, we observed three identical services to be running on Alice and Bob namely: synchronization, remote-connection management and error-correction each of which accounted for 60%, 22% and 1% for Alice and 72%, 27% and 1% for Bob. The ETSI QKD GS 0014 orchestration and key management interface [13] was present only on Alice and accounted for the remaining 17% for Alice. The synchronization service included three actions to synchronize

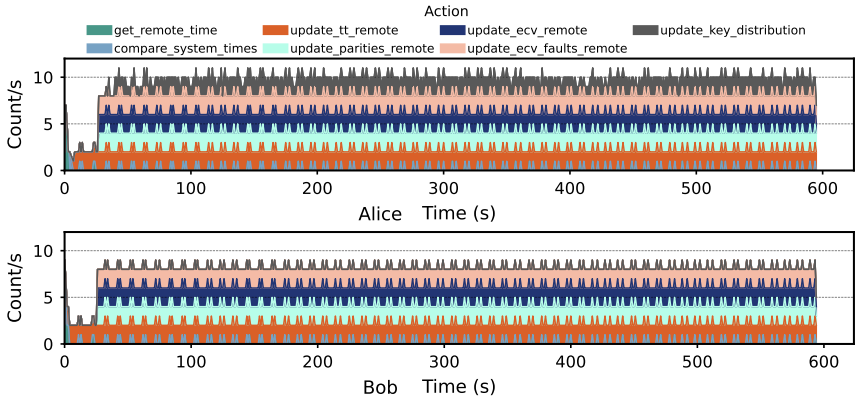

Fig. 3: Distribution of messages starting from synchronization (first 25 s) process up to nearly ten minutes of key generation.

system clocks and the detector clocks. However, one of the actions was also used to piggy-back the detector basis measurements throughout the communication. The error-correction service included three actions and the orchestration merely one action.

4.3 Traffic Analysis

Once we could interpret the message types and message content, we created a time-series of the messages exchanged to observe the phases of the actions exchanged during the synchronization, error-correction and orchestration plotted in Figure 3. From the time-series plot we can discern that the actions performed are highly periodic which indicates that the protocol exhibits highly regular, cyclic message patterns with predictable sequencing, making it vulnerable to targeted manipulation. Alice in this case, has one more action than Bob which involves sending the `OKMS` message that is used between Alice and Bob to agree upon a `key_id` that links an identifier with the generated key after post-processing. For traffic flowing from Alice → Bob, we observed a peak bandwidth of 35.83 Mbps and an average bandwidth of 0.56 Mbps. The mean packets per second (pps) was 9.47 and the peak pps was 11. A total of 41.8 MB was sent with a total of 5646 application layer messages. For traffic flowing from Bob → Alice, we observed a peak bandwidth of 5.37 Mbps and an average bandwidth of 5.35 Mbps . The average pps was 7.95 and the peak pps was 9. A total of 39.8 MB was sent with a total of 4733 application layer messages. From these network statistics, we note that Alice sends 85% of its data at the very start of the synchronization, and sends the remaining 15% over the ten minutes. Bob on the other hand has its 39.8 MB roughly evenly distributed over the ten minutes.

Table 1: Specifications of the tested QKD system and Pickle-TCP fault-injection proxy.

System (Pickle-TCP)		QKD Stack	
Param	**Value**	**Param**	**Value**
CPU	AMD EPYC 8024P (8-Core, 3.0 GHz)	CPU	Intel Xeon D-1718T (8-Core, 2.6 GHz)
RAM	64 GB	RAM	8 GB
NIC	2×Solarflare SFC9220 10/40G	NIC	1 GbE Intel I219 (Alice & Bob)
OS	Ubuntu 24.04 LTS	OS	Debian GNU/Linux 11
Kernel	6.8.0-58-generic	Kernel	5.10.0-26-amd64
Python	v3.12.3	Wavelength	810 nm (Alice), 1550 nm (Bob)
Tproxy	Linux kernel module	PAM	Auto-Polarization Control
		Channels	Standard SMF (wavelength dep.)
		Length	30 km (100 km max)
		Link Loss	Sum of channel losses

Experimental Setup

Figure 1 depicts our security testing infrastructure. We used a research-grade BBM92 ELVIS 1550 QKD system from Quantum Optics Jena that comprise of an entangled-photon pair source that splits the entangled-photon pair into 810 *nm* and 1550 *nm* wavelengths that are sent to Alice and Bob respectively. Eve the eavesdropper is shown by the server that connects the classical network between Alice and Bob. Eve uses the tcp-based fault-injection proxy we developed Pickle-TCP to intercept and tamper with the TCP-based traffic that is exchanged between Alice and Bob. The QKD system specification and Pickle-TCP aka Eve are tabulated in Table 1.

5 Pickle-TCP Fault Injection

We now leverage our traffic analysis insights of how the QKD systems work from the previous section, to think like an attacker and attempt to compromise the QKD systems as outlined in our attacker model (Section 3). Assuming to be an on-path adversary, our first challenge is to be able to introduce faults into the TCP streams. By faults, we mean, a message could be blocked, delayed or re-ordered; it also means a new message could be injected into the stream. However, since every TCP message is enticated by a HMAC, tampering with the packets will merely be ignored by the receiver. Second, even if we were able to tamper with the TCP payload and the signature, and the size of the message changes, we'd have to redo the checksum and sequence numbers which complicates our attack. To overcome this challenge, we leveraged the Linux kernel's Transparent proxy support (`TPROXY`) to develop our custom Pickle-TCP fault-injection proxy with the following design goals.

5.1 Design Goals

We had the following design goals for PICKLE-TCP. **Transparent in-band operation:** The fault-injector must see every byte that the classical channel sees while remaining invisible to both QKD peers. **Protocol-aware:** Rather than working with raw TCP segments, the proxy decodes each length-prefixed Pickle object, allowing us to delay, drop, or inject at the level of protocol actions, e.g., drop `key_id` messages. **Modular design:** As new attacks are developed, they can be easily integrated into our system

5.2 Implementation

As shown in Figure 1, we set-up a server between Alice and Bob that can either operate as a software switch or as Pickle-TCP fault-injection proxy. When fault-injection is enabled, Linux TPROXY transparently diverts the traffic through the proxy. TPROXY is a Linux Netfilter that diverts selected TCP/UDP traffic to a local user-space socket while preserving the packets' original destination and source addresses and ports as observed by applications. Unlike `DNAT/REDIRECT`, no L3/L4 header rewriting is performed on the receive path: the kernel delivers the packet to the proxy as if it were addressed to the original 5-tuple. When combined with the `IP_TRANSPARENT` socket option and appropriate policy routing, the proxy can also originate connections using the client's source address, making the server perceive the original client IP. For TCP, the proxy terminates the client connection and initiates a separate server-facing connection; therefore, sequence numbers, window scaling, selective acknowledgments, and negotiated TCP options are established independently on each socket rather than preserved end-to-end. This arrangement enables user-space software to intercept and control flows transparently.

The user first provides a yaml configuration file that describes the properties and composition of actions of the attack. When the decoder receives the TCP segment, it decodes the TCP payload and then sends it to the payload handler. Depending on the type of message and the attack configuration, the payload handler then dispatches the payload to the respective attack. If no attack is configured, the messages is encoded-back and forwarded to TPROXY which then takes care of sending out the appropriate socket. This works bi-directionally. Attacks can be composed to target multiple actions, e.g., drop one type of action and then delay another type of action. Attacks can be started or stopped by updating the configuration file at run-time. In the current version which is open-source, we have implemented the following attacks:

- **Block:** In this attack, actions specified in the configuration file will be blocked.
- **Delay:** In this attack, actions specified in the configuration file will be delayed by the specified delay-time.
- **Inject:** In this attack, the attacker can inject messages into the stream. If at the Pickle-TCP layer, then this would also require the message authentication code to be appended to the message.

5.3 Fault-Injection Methodology

Armed with our protocol analysis revealing predictable message sequences (Section 4) and our Pickle-TCP fault-injection proxy, we scrutinized the authentication protocol, service interactions, and timing dependencies to identify the corresponding messages or sequence of messages that could be potentially exploited by an attacker. We identified the following services and actions for the following reasons:

- `Authentication protocol` is a sequence of messages that is used by Alice and Bob to authenticate each other as shown in Figure 2. Could Eve obtain legitimate signatures of her messages that she could inject into the channel?
- `Synchronization messages` are needed to keep the system in sync. What happens if TCP segments are delayed or dropped? Would the systems continue to generate keys? What is the maximum tolerable delay?
- `Orchestration and key management` are defined by the ETSI GS QKD 014 standard and involves sending a message with a key_id. What happens if this message is blocked or delayed? Would it impact the keys generated?
- `Error-correction` are messages needed for post-processing based on the BB92 protocol. What happens if this message is blocked or delayed? Would it impact the keys generated?

Based on the potential services to target, we implemented configuration files for the aforementioned attacks using Pickle-TCP. We captured the TCP streams from 0-100 synchronization and then stable key generation (QBER of less than 11%), injected the fault, and continued to capture the traffic. Each experiment was run for four minutes from the first moment that the nodes were 100% % synchronized unless otherwise stated. We capture network traffic at the respective egress interfaces of our proxy for Alice→Bob and Bob→Alice traffic.

In the following, we report on three exploitable vulnerabilities from our security tests. We validated all three attacks in 10 independent runs each, where an independent run denotes a fresh reboot of the QKD nodes followed by a new session establishment. For the authentication attack, the success criterion was: *we obtain a remote shell on the compromised system.* For the silent-desynchronization attack, the success criterion was observing a drop in the `update_key_distribution` in the Alice→Bob traffic and a divergence in the number of keys stored by Alice and Bob during the attack interval, which we verified directly from the keypool traces. For the delay attack, the success criterion was observing changes in the `update_key_distribution` and synchronization state of Alice and Bob and that no new keys were generated during the attack interval, again verified from the keypool traces. Unless otherwise stated, a representative trace is shown in Figure 5 and 2. The interested reader can review the Appendix 11 for visualizations from all the runs. For the two synchronization attacks, we injected faults only after stable operation was reached and sustained the attack for 60 s in the plotted runs; the silent-desynchronization attack dropped Alice-to-Bob OKMS/`key_id` also referred to as `update_key_distribution` messages, while the delay attack delayed Alice-to-Bob `update_tt_remote` messages.

The 250 ms setting was selected because the delay sweep showed no sustained disruption for 50–200 ms, whereas 250 ms was the smallest delay that consistently stalled the system in our setup (see Figure 8 in the Appendix 11 for the data).

6 Authentication Protocol Vulnerability: Signing Oracle Exploitation

A critical result from our security testing was the discovery of a vulnerability in the authentication mechanism. While the use of Python pickle for serialization exacerbated the impact (allowing remote code execution), the root cause is a non-trivial logical flaw in the authentication protocol design itself. This vulnerability allows an on-path attacker to treat the legitimate QKD node as a "signing oracle," injecting attacker-controlled data into the classical channel that the system validates as authentic. This is particularly dangerous because QKD systems effectively trust the classical channel implicitly once authentication is deemed successful.

Root Cause: Protocol-Level Signing Oracle

The authentication vulnerability we discovered stems from a fundamental flaw in the challenge-response protocol design, independent of any serialization mechanism. While Python Pickle exacerbated the impact (enabling remote code execution), the core security failure lies in treating the legitimate peer as an unwitting *signing oracle*.

The Protocol Weakness. HMAC is a symmetric message authentication code: both parties share the same secret key K and can generate valid tags on arbitrary messages. The protocol's vulnerability arises from a subtle but critical flaw: it allows an adversary to choose the input that gets authenticated without binding that input to a legitimate protocol context.

As shown in Figure 2, Bob sends Alice a `CHALLENGE` containing `RANDOM_BYTES`, and Alice responds with $\text{HMAC}_K(\text{RANDOM\_BYTES})$. However, an on-path attacker Eve can intercept Bob's challenge and replace `RANDOM_BYTES` with attacker-controlled data $m_{\text{malicious}}$ before forwarding to Alice. Alice, believing she is authenticating to Bob, computes $\text{HMAC}_K(m_{\text{malicious}})$ and returns it. Eve now possesses a valid authentication tag for arbitrary attacker-chosen data, issued by a legitimate node. The key generation of Alice and Bob are not impacted by this attack as shown in Table 2.

Impact Independent of Serialization. Even without Pickle's remote code execution capabilities, this signing oracle enables several attacks:

- **Message injection:** Eve can inject $(m_{\text{malicious}}, \text{HMAC}_K(m_{\text{malicious}}))$ into the classical channel, bypassing authentication.
- **Replay attacks:** Authenticated attacker-chosen messages can be replayed at strategic protocol phases.

– **Protocol manipulation:** Forged control messages (e.g., repeated key identifiers, modified synchronization parameters) can desynchronize nodes or corrupt key pools.

Why Pickle Made It Worse. The use of Python Pickle for serialization transformed a message-injection vulnerability into remote code execution. Pickle deserializes arbitrary Python objects, including those that execute code during unpickling (via `__reduce__` methods). By obtaining a valid HMAC for a malicious Pickle payload, Eve could inject executable code that passes authentication checks, achieving complete system compromise.

Proper Protocol Design. Secure challenge-response protocols prevent signing oracle attacks through:

– **Context binding:** HMACs must include protocol-specific context (session identifiers, role markers, sequence numbers).
– **Mutual authentication structure:** The responder should prove knowledge of K without producing MACs over challenger-chosen data.
– **Freshness guarantees:** Nonces or counters prevent replay attacks.
– **Transcript authentication:** MACs should cover the entire handshake history, not isolated challenges.

The authentication vulnerability underscores the danger of bespoke security protocols, and the unspoken truth about QKD: the absence of standardized, peer-reviewed authentication schemes. ETSI GS QKD 008 on the QKD Modulel Security Specification [9] provides no concrete guidance on authentication protocol selection, forcing each vendor to navigate these challenges independently.

Exploiting The Authentication Handshake.

Figure 2 depicts the message-sequence pattern of Eve exploiting the authentication vulnerability over three phases. In phase 1, Eve wants to get her payload signed. She waits for the TCP connections to be established. In the figure, Alice has initiated the three-way TCP handshake with Bob which has been intercepted by Eve with our Pickle-TCP Fault Injection to replace `RANDOM_BYTES` with her own data and forwards it to Alice. Untrusted deserialization of pickled data could spawn a reverse bash shell, which allows an attacker to access both servers remotely. Alice then uses her pre-shared key (with Bob) to sign the payload and returns `HMAC(RANDOM_BYTES)`, which Eve gets access to along the path. This successfully completes phase one. Eve then resets the TCP connection with Alice using a TCP-RST flag. In phase two, Eve re-establishes a TCP connection with Alice and forwards Bob's initial `CHALLENGE` and `RANDOM_BYTES` to Alice. Eve then lets the handshake complete by only forwarding the TCP segments. After Alice and Bob have authenticated each other in both directions, phase three begins. In phase three Eve, injects the malicious Pickle payload to Alice and Bob which is executed by both of them and gives the attacker a shell from which she can read out the QKD-generated keys. We validated this exploit in

ten independent runs and obtained a remote shell on both QKD servers in every run.

Identifying this flaw required deep protocol awareness: to a standard firewall or intrusion detection system, these messages appear as valid, authenticated traffic. This highlights the immense difficulty of designing bespoke authentication protocols for QKD, where the assumption of a "pre-authenticated" classical channel often leads vendors to implement ad-hoc, insecure solutions.

7 State-Synchrony Failure

In QKD, state synchrony ensures that nodes remain coordinated through classical channel exchanges used for basis state announcements, clock synchronization, error correction, and key distillation. Proper synchrony ensures nodes maintain coordinated protocol execution through predictable message sequencing. Here, we evaluate vulnerabilities where an on-path attacker manipulates specific classical channel messages, either to desynchronize the nodes or stall the key generation process, impacting availability and potentially causing data loss.

Silent Desynchronization

The classical channel exchanges key-synchronization messages known as OKMS messages. In our deployment, these messages carry the `key_id` values used by Alice and Bob to keep their key pools aligned. Using Pickle-TCP, we silently dropped OKMS messages from Alice to Bob for 60 s. This attack resulted in Bob's key-generation rate dropping to zero while Alice continued generating keys normally, as depicted in Figure 4.

In the representative run of Figure 4, Alice generated 54 keys during the 62 s attack window, corresponding to 0.870968 keys/s and thus remaining essentially at her 0.874172 keys/s baseline. Bob, in contrast, generated no keys at all during the same interval. Over the full 265 s trace, Alice accumulated 209 keys whereas Bob accumulated only 154, leaving a final 55-key mismatch between the two key pools.

The keypool data in Table 2 verifies that this attack succeeded for one representative run. Across all ten runs, Bob is the victim in 10/10 cases, with a mean 99.3% attack-window key-rate drop and zero-key streaks lasting 61–63 s. Alice remained largely unaffected, with a mean drop of 4.5% and longest zero-key streaks of only 1 s. Thus, the attack silently creates persistent key-pool divergence while the non-targeted side continues to operate near baseline.

In Section 4 of ETSI GS QKD 014 [13], paragraph three states the following: "Keys are generated and shared securely with QKD technology by KMEs. Key management methods used by KMEs and how KMEs relay keys securely in a QKD network is outside the scope of the present document." It does not standardize how peer QKD nodes internally synchronize `key_id` values over their proprietary control channel. In our testbed, these identifiers were exchanged over the same channel as synchronization and error-correction messages. Thus,

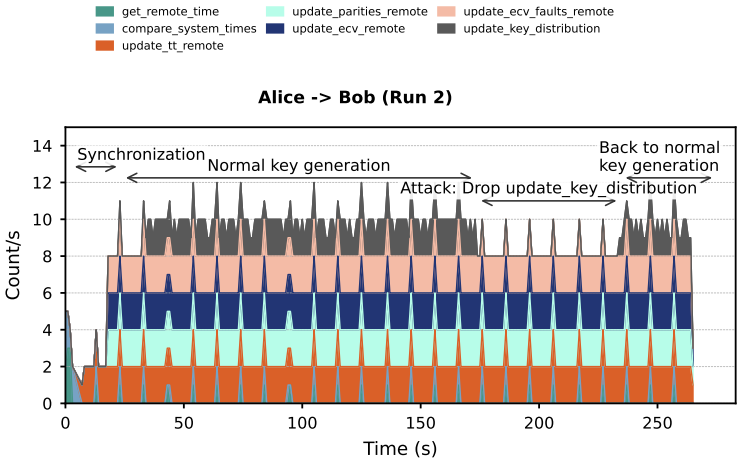

Fig. 4: Effect of selectively blocking `OKMS` messages containing key identifiers (`key_id`). The annotated time-series show the precise moment the packets are dropped by our proxy for Alice→Bob traffic.

silently dropping OKMS messages led directly to out-of-sync key pools and created a reliability failure for subsequent encryption/decryption operations.

Stalled Key Generation

The second attack demonstrates classical-channel sensitivity to network-induced latency. The `update_tt_remote` message in our QKD system serves dual purposes: it synchronizes clocks and carries measurement-related information needed for subsequent post-processing. We evaluated multiple induced delays and observed a threshold effect in our setup:

For 50, 100, 150, and 200 ms, the largest observed `update_key_distribution` gaps were 1.51 s, 1.03 s, 10.01 s, and 1.02 s respectively (see the Appendix 11 for all the runs). We assumed a 20 s sustained-disruption threshold to account for more than a "network glitch", so these settings did not trigger the same failure mode. Sustained disruption first appears at 250 ms. In the representative 250 ms run, both Alice and Bob fall from a 0.933 keys/s baseline to 0.016 keys/s during the 61 s attack window, each producing only one key and exhibiting a 60 s zero-key streak. The same collapse persists at 300 and 400 ms, while at 500 ms the zero-key streak extends to 78–79 s and Bob's attack-window drop reaches 100%. We therefore report 250 ms as the smallest delay that consistently stalled our system in these experiments. By delaying these messages using Pickle-TCP, the attacker forces the nodes into a continuous synchronization state without producing keys. This denial-of-service (DoS) attack persisted throughout our attack interval, as illustrated in Figure 5. From the QKD vendor's web-based GUI monitoring interface, synchronization status misleadingly remained at 100%, with no

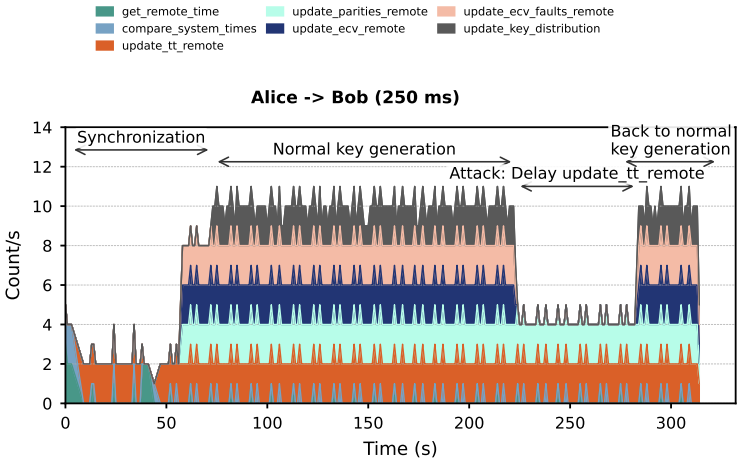

Fig. 5: Impact of a 250 ms induced delay in synchronization messages (`update_tt_remote`). We can see that key generation halts as the `update_key_distribution` is not sent despite Alice being synchronized with Bob.

indication of issues, despite the total absence of key generation in the key pool files.

This attack highlights the ease and stealthiness with which subtle manipulation of classical-channel traffic can disrupt critical QKD processes. Without robust detection mechanisms for minor network delays, such attacks are difficult for defenders to diagnose and can easily be misattributed to benign network fluctuations.

8 QKD Authentication Bootstrapping Problem and PQC-Enabled WireGuard Tunnels As A Practical Solution

Our attacks demonstrate that practical QKD security depends critically on encrypting and authenticating the classical control channel, yet this creates a fundamental bootstrapping paradox. QKD's information-theoretic security guarantees assume an authenticated classical channel exists, but generating the initial authentication keys presents a circular dependency: we cannot establish authenticated communication without pre-shared keys, yet we cannot generate QKD keys without authenticated communication which we term as the "authentication bootstrapping problem". This is where post quantum cryptography plays a key role.

Post-quantum cryptography (PQC) and QKD address complementary aspects of long-term security, and neither alone suffices for robust QKD deploy-

Table 2: Representative keypool generation metrics for the authentication, block, and delay attacks, and for a representative Rosenpass run.

Case	Side	Window	Duration (s)	Keys Generated	Avg. Rate (keys/s)	SD (keys/s)	Peak Rate (keys/s)	Zero Seconds	Longest Zero Streak (s)
Authentication attack (Run 3)	Alice	Full	253	234	0.925	0.364	2	27	4
Authentication attack (Run 3)	Bob	Full	253	234	0.925	0.364	2	27	4
Block `update_key_distribution` (Run 2)	Alice	Full	265	209	0.789	0.492	2	66	23
Block `update_key_distribution` (Run 2)	Alice	Baseline	151	132	0.874	0.479	2	28	10
Block `update_key_distribution` (Run 2)	Alice	Attack	62	54	0.871	0.380	2	9	1
Block `update_key_distribution` (Run 2)	Alice	Recovery	31	23	0.742	0.438	1	8	1
Block `update_key_distribution` (Run 2)	Bob	Full	265	154	0.581	0.558	2	120	61
Block `update_key_distribution` (Run 2)	Bob	Baseline	151	132	0.874	0.479	2	28	10
Block `update_key_distribution` (Run 2)	**Bob**	***Attack***	**62**	**0**	**0.000**	**0.000**	**0**	**62**	**62**
Block `update_key_distribution` (Run 2)	Bob	Recovery	31	22	0.710	0.454	1	9	2
Delay `update_tt_remote` (250 ms)	Alice	Full	314	170	0.541	0.529	2	149	73
Delay `update_tt_remote` (250 ms)	Alice	Baseline	151	141	0.934	0.339	2	14	1
Delay `update_tt_remote` (250 ms)	**Alice**	***Attack***	**61**	**1**	**0.016**	**0.127**	**1**	**60**	**60**
Delay `update_tt_remote` (250 ms)	Alice	Recovery	31	28	0.903	0.390	2	4	1
Delay `update_tt_remote` (250 ms)	Bob	Full	314	169	0.538	0.530	2	150	73
Delay `update_tt_remote` (250 ms)	Bob	Baseline	151	141	0.934	0.339	2	14	1
Delay `update_tt_remote` (250 ms)	**Bob**	***Attack***	**61**	**1**	**0.016**	**0.127**	**1**	**60**	**60**
Delay `update_tt_remote` (250 ms)	Bob	Recovery	31	27	0.871	0.421	2	5	1
Rosenpass (Run 5)	Alice	Full	225	180	0.800	0.499	2	55	9
Rosenpass (Run 5)	Bob	Full	225	179	0.796	0.493	2	55	9

ment. PQC provides computationally-hard authentication and key exchange resistant to quantum attacks, solving the bootstrapping problem by establishing initial authenticated channels. However, PQC cannot prove an adversary has not learned secret keys, security relies on mathematical assumptions (e.g., lattice problem hardness) that, while believed quantum-resistant, remain vulnerable to "Store Now Decrypt Later" attacks through future algorithmic advances or implementation flaws. Conversely, QKD generates symmetric keys with information-theoretic security through the no-cloning theorem in quantum mechanics, provably preventing adversaries from learning keys through channel eavesdropping without detection. Yet QKD fundamentally requires pre-existing authenticated channels to function securely; without authentication, man-in-the-middle attacks trivially compromise QKD. The combination leverages PQC for bootstrapping initial authentication credentials while QKD-generated keys refresh and sustain long-term security with information-theoretic guarantees. This strategy combines computational security for cold-start authentication with information-theoretic security for sustained operation, ensuring ongoing key generation remains secure even if PQC assumptions are eventually broken.

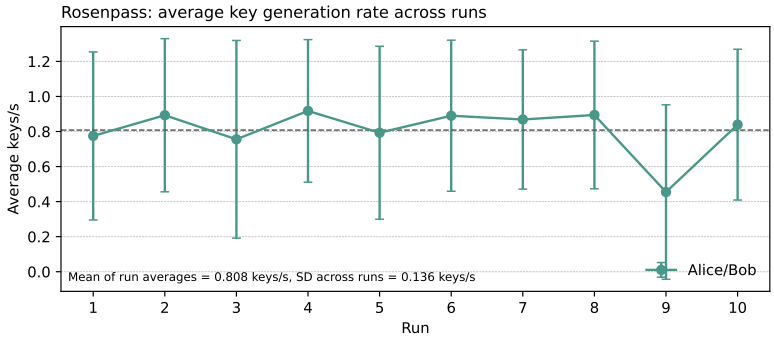

Fig. 6: Rosenpass/WireGuard tunneling overhead across ten runs. The mean run-average key-generation rate is 0.808 keys/s.

PQC+QKD Integration for Classical Channel Security

While the theoretical benefits of combining PQC and QKD have been discussed [7], no prior work has empirically validated this integration for protecting QKD classical control channels. Our contribution is demonstrating that PQC-tunneled QKD is practically viable using WireGuard [8], a modern VPN protocol, that supports third-party modules like Rosenpass [39], enabling the establishment of PQC-secured tunnels. To the best of our knowledge, this is the first application of such a scheme to protect QKD control-plane traffic. We implemented a drop-in Rosenpass WireGuard tunnel to encapsulate the entire classical communication path between QKD sites by replacing our Pickle-TCP fault-injection proxy with WireGuard. This approach ensures confidentiality and integrity against both passive eavesdropping and active manipulation, closing a critical systems-security gap.

Across ten Rosenpass runs, the mean run-average key-generation rate was 0.808 keys/s with a standard deviation of 0.136 keys/s, and individual runs ranged from 0.455 to 0.918 keys/s, as shown in Figure 6. In the representative run in Table 2, Alice and Bob generated 180 and 179 keys over 225 s, corresponding to 0.800 and 0.795556 keys/s respectively. The peak key rate remained unchanged at 2 keys/s on both sides. This overhead of approximately 6.6% is worthwhile because the tunnel prevents the content-aware injection and tampering exploited by Pickle-TCP. However, as with any network tunnel, it does not by itself eliminate pure delay/drop availability attacks by an on-path adversary.

Our implementation is compatible with existing QKD software stacks and requires no invasive code changes, supporting practical deployment. A robust production implementation should ideally terminate the tunnel directly at the KME servers, aligning with zero-trust principles. We recommend configuring WireGuard for perfect forward secrecy (PFS), potentially by seeding the initial handshake with PQ-resistant key exchange (via Rosenpass) and periodically refreshing with QKD-derived keys, thus segmenting key pools for distinct secu-

rity purposes PQC for initial authentication, QKD for sustained information-theoretic security.

9 From Cryptographic Optimism to Systems Realism

From a Single-Vendor Study to Broader Lessons. Our empirical evaluation examined one commercial BBM92 deployment, so we do not claim that every QKD implementation exhibits the same flaws. Rather, our results show how security-relevant choices on the classical control plane can create exploitable failures in practice. In our deployment, the authentication flaw (Section 6) arose from a bespoke challenge-response design combined with unsafe serialization; the synchronization attacks (Section 7) exploited assumptions about reliable and timely network delivery. These specific failures are implementation-dependent, but they also illustrate a broader lesson: when authentication, peer synchronization, and resilience behavior are left to proprietary control-plane designs, the resulting attack surface can be substantial. Given the high cost of BBM92 systems (exceeding $300,000 for hardware alone) and the small vendor ecosystem, we view this paper as a baseline empirical case study rather than a universal claim about all QKD products.

Call for Standardization. ETSI documents cover important parts of the QKD stack, but their scope should be read carefully. ETSI GS QKD 014 standardizes the northbound key-delivery interface between KMEs and applications, while ETSI GS QKD 016 defines a protection profile for QKD modules from physical implementation to the output of final secret keys [13,16]. In contrast, our evaluation targets the proprietary peer-to-peer classical control channel between the QKD nodes. For this channel, critical design choices remain open in practice: concrete authentication handshakes, bootstrap-key establishment, peer `key_id` synchronization, and resilience to delay/drop faults. We therefore advocate: (1) standardized and peer-reviewed authentication mechanisms for internal classical control traffic; (2) explicit control-plane specifications with defined message flows, state transitions, and corner-case handling; and (3) reference architectures for synchronization, observability, and availability resilience in QKD-SDN environments [15].

Inherited Internet Threats. Our evaluation shows that QKD deployments inherit familiar Internet threats because their control planes run over general-purpose packet networks. The observed key-pool divergence and synchronization attacks stem from protocol designs that implicitly trust the network or fail to provide robust synchronization and visibility. Moving to out-of-band, time-based one-time passwords for key synchronization would mitigate these risks by eliminating the need for sensitive state to cross the classical channel.

Fragility and the Need for Robust Systems Design. Our analysis exposes a gap between QKD's cryptographic promises and its real-world software and network implementation. Fragility arises from weak state machines, unauthenticated control traffic, insecure serialization (e.g., pickle), and protocol designs that cannot handle packet loss, reordering, or adversarial interference. Integrat-

ing network metrics into quantum models and segmenting key management could harden QKD against such systemic threats, but only if paired with rigorous systems engineering.

10 Related Work

Prior research on QKD security has predominantly focused on quantum-optical vulnerabilities, formal security analyses, and performance aspects of deployed systems. In contrast, fewer studies have systematically examined the classical control channel from a network-security perspective.

Quantum-optical and physical-layer attacks. Sajeed et al. [41] introduced a hierarchical framework primarily targeting quantum-optical vulnerabilities in quantum communication systems, but did not address network-layer manipulations or protocol-level attacks on classical channels. Sushchev et al. [43], Jain et al. [22], and Biswas et al. [4] experimentally studied optical or device-level side channels in practical QKD systems. These works target the quantum or physical layer, whereas we exclusively analyze the classical control plane.

Classical-channel performance, key management, and DoS. Mehic et al. conducted multiple analyses [32, 33] on classical/quantum channel interactions. Their 2017 study [32] showed that increased quantum bit error rates do not necessarily increase public-channel traffic, underscoring the importance of classical-channel performance. Their 2022 study [33] addressed denial-of-service (DoS) vulnerabilities in QKD key management for software-defined networks using centralized SDN-based detection. Tankovic et al. [44] analyzed the performance of ETSI GS QKD 014 in multi-user scenarios. Our work complements these studies by actively exploiting weaknesses in the classical message protocols themselves rather than focusing on throughput, monitoring, or resource management.

Authentication and reconciliation security. Pacher et al. [35] discussed man-in-the-middle attacks against authentication schemes that are not information-theoretically secure, primarily in a theoretical model with stronger cryptanalytic capabilities than ours. Kiktenko et al. [23] proposed lightweight authentication mechanisms for resource-constrained QKD systems. Lamas-Linares et al. [25] and Kim et al. [24] studied leakage during reconciliation and parity computation. In contrast, we show how an on-path adversary can exploit authentication and synchronization behavior over the deployed control channel itself.

Protecting the classical channel with PQC/TLS and standardization. Das et al. [7] secured ETSI-compliant key APIs with post-quantum cryptographic proxies at the application interface. Garcia et al. [40] proposed quantum-resistant TLS mechanisms for classical channels. The ETSI ISG QKD Activity Report 2023 [18] summarized ongoing standardization work. Our paper differs in two ways: we first empirically characterize exploitable vulnerabilities in a proprietary QKD control plane, and then evaluate a practical PQC tunnel (WireGuard + Rosenpass) as a deployment-oriented mitigation for that control plane.

11 Conclusion

This paper presents a system-security evaluation of the classical control channel in a commercial BBM92 QKD deployment. By developing Pickle-TCP, a transparent fault-injection proxy, we reverse-engineered the undocumented control protocol and demonstrated that practical QKD systems can face severe network-layer vulnerabilities despite the security properties of the quantum channel.

Our analysis shows that while the quantum layer may resist eavesdropping, the classical layer can remain fragile. In the evaluated deployment, we exploited an authentication flaw that, together with unsafe Pickle deserialization, yielded remote code execution. We also demonstrated that the state machine is susceptible to timing and message-suppression attacks that silently desynchronize key pools or halt key generation without triggering alarms.

These findings highlight a gap between cryptographic theory and engineering reality. More precisely, they show that under-specified classical-control-plane decisions can create exploitable failures in deployed QKD systems. Proxying the TCP traffic and exploiting the authentication protocol worked 10/10 times without disrupting the key generation rate on Alice and Bob. Keypool analysis confirms that blocking `update_key_distribution` makes Bob the victim in 10/10 runs with a mean 99.3% attack-window rate drop, while delaying `update_tt_remote` reveals a sharp threshold at 250 ms, where both sides collapse by more than 98%. To mitigate content-aware tampering and injection on this channel, we implemented and evaluated a PQC-secured tunnel using Rosenpass (which integrates PQC with WireGuard), which achieved a mean run-average of 0.808 keys/s across ten runs translating to around a 6.6% overhead. This protection is practical and has been adopted by the vendor, but it should be paired with stronger availability resilience against pure delay/drop attacks.

Our broader contribution is to argue for a system-security perspective on QKD deployment: robust authentication mechanisms, well-specified synchronization behavior, safer serialization choices, standardized software interfaces, and zero-trust network assumptions for the classical channel. Looking ahead, the quantum-networking community should treat the classical control plane as a first-class security boundary if quantum advantage is to remain meaningful in realistic adversarial environments.

12 Acknowledgements

The authors thank the anonymous reviewers and shepherd for their feedback and improvements to this paper. We acknowledge financial support by the German Federal Ministry of Research, Technology and Space (BMFTR) in the framework of the Q-Fiber project number 16KISQ124 and the Q-net-Q Project which has received funding from the European Union's Digital Europe Programme under grant agreement No 101091732, and is co-funded by the BMFTR. We thank Kevin Füchsel, Oliver de Vries, Pablo Vazquez, René Heilmann and Arno Dorl for acknowledging our findings and supporting our work throughout the project. We express our thanks to Thomas Hühn and

Paul Spooren for the discussion on Rosenpass. First author K. T. acknowledges Marvin Saas and Niclas Kühnapfel for sharing their valuable insights during the initial stage of this research.

References

1. ACM: Charles H. Bennett and Gilles Brassard are the recipients of the 2025 ACM A.M. Turing Award for their essential role in establishing the foundations of quantum information science and transforming secure communication and computing, https://awards.acm.org/about/2025-turing
2. Barrett, J., Colbeck, R., Kent, A.: Memory Attacks on Device-Independent Quantum Cryptography. Physical Review Letters **110**(1), 010503 (Jan 2013). https://doi.org/10.1103/PhysRevLett.110.010503, https://link.aps.org/doi/10.1103/PhysRevLett.110.010503
3. Bennett, C.H., Brassard, G., Mermin, N.D.: Quantum cryptography without bell's theorem. Physical Review Letters **68**(5), 557–559 (1992). https://doi.org/10.1103/PhysRevLett.68.557
4. Biswas, A., Banerji, A., Chandravanshi, P., Kumar, R., Singh, R.P.: Experimental Side Channel Analysis of BB84 QKD Source. IEEE Journal of Quantum Electronics **57**(6), 1–7 (Dec 2021). https://doi.org/10.1109/jqe.2021.3111332, https://ieeexplore.ieee.org/document/9531968/, publisher: Institute of Electrical and Electronics Engineers (IEEE)
5. Core, I.: Harvest Now, Decrypt Later: The Quantum Security Threat (Apr 2025)
6. Curty, M., Lo, H.K.: Foiling covert channels and malicious classical post-processing units in quantum key distribution. npj Quantum Information **5**(1), 14 (Feb 2019). https://doi.org/10.1038/s41534-019-0131-5, https://www.nature.com/articles/s41534-019-0131-5
7. Das, S., Varshney, A., Ravi, P., Chattopadhyay, A.: Mind the Gap: Securing QKD Interfaces with Post-Quantum Proxies (2025), https://eprint.iacr.org/2025/1177, publication info: Preprint.
8. Donenfeld, J.A.: WireGuard: Next Generation Kernel Network Tunnel. In: Proceedings 2017 Network and Distributed System Security Symposium. Internet Society, San Diego, CA (2017). https://doi.org/10.14722/ndss.2017.23160, https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/wireguard-next-generation-kernel-network-tunnel/
9. ETSI: Quantum Key Distribution (QKD); QKD Module Security Specification. Group Specification (GS) ETSI GS QKD 008 V1.1.1, European Telecommunications Standards Institute (ETSI), Sophia Antipolis, France (Dec 2010), https://www.etsi.org/deliver/etsi_gs/qkd/001_099/008/01.01.01_60/gs_qkd008v010101p.pdf, version 1.1.1, published December 2010
10. ETSI: Quantum Key Distribution (QKD); Security Proofs. Group Specification (GS) ETSI GS QKD 005 V1.1.1, European Telecommunications Standards Institute (ETSI), Sophia Antipolis, France (Dec 2010), https://www.etsi.org/deliver/etsi_gs/qkd/001_099/005/01.01.01_60/gs_qkd005v010101p.pdf, version 1.1.1, published December 2010
11. ETSI: Quantum Key Distribution (QKD); Component Characterization: Characterizing Optical Components for QKD Systems. Group Specification (GS) ETSI GS QKD 011 V1.1.1, European Telecommunications Standards Institute (ETSI), Sophia Antipolis, France (May 2016), https://www.etsi.org/deliver/etsi_gs/

qkd/001_099/011/01.01.01_60/gs_qkd011v010101p.pdf, version 1.1.1, published May 2016
12. ETSI: Quantum Key Distribution (QKD); Device and Communication Channel Parameters for QKD Deployment. Group Specification (GS) ETSI GS QKD 012 V1.1.1, European Telecommunications Standards Institute (ETSI), Sophia Antipolis, France (Feb 2019), https://www.etsi.org/deliver/etsi_gs/QKD/001_099/012/01.01.01_60/gs_qkd012v010101p.pdf, version 1.1.1, published February 2019
13. ETSI: Quantum Key Distribution (QKD); Protocol and Data Format of REST-based Key Delivery API. Group Specification (GS) ETSI GS QKD 014 V1.1.1, European Telecommunications Standards Institute (ETSI), Sophia Antipolis, France (Feb 2019), https://www.etsi.org/deliver/etsi_gs/QKD/001_099/014/01.01.01_60/gs_qkd014v010101p.pdf, version 1.1.1, published February 2019
14. ETSI: Quantum Key Distribution (QKD); Application Interface. Group Specification (GS) ETSI GS QKD 004 V2.1.1, European Telecommunications Standards Institute (ETSI), Sophia Antipolis, France (Aug 2020), https://www.etsi.org/deliver/etsi_gs/QKD/001_099/004/02.01.01_60/gs_qkd004v020101p.pdf, version 2.1.1, published August 2020
15. ETSI: Quantum Key Distribution (QKD); Orchestration Interface for Software Defined Networks. Group Specification (GS) ETSI GS QKD 018 V1.1.1, European Telecommunications Standards Institute (ETSI), Sophia Antipolis, France (Apr 2022), https://www.etsi.org/deliver/etsi_gs/QKD/001_099/018/01.01.01_60/gs_qkd018v010101p.pdf, version 1.1.1, published April 2022
16. ETSI: Quantum Key Distribution (QKD); Common Criteria Protection Profile — Pair of Prepare and Measure Quantum Key Distribution Modules. Group Specification (GS) ETSI GS QKD 016 V2.1.1, European Telecommunications Standards Institute (ETSI), Sophia Antipolis, France (Jan 2024), https://www.etsi.org/deliver/etsi_gs/QKD/001_099/016/02.01.01_60/gs_QKD016v020101p.pdf, version 2.1.1, published January 2024
17. EU: Quantum communication infrastructure (EuroQCI): Call completed with 24 proposals - European Health and Digital Executive Agency (HaDEA). https://hadea.ec.europa.eu/news/quantum-communication-infrastructure-euroqci-call-completed-24-proposals-2025-03-31_en
18. European Telecommunications Standards Institute (ETSI): ISG QKD Activity Report 2023. https://www.etsi.org/committee-activity/activity-report-qkd (2023), web page, accessed 2 August 2025
19. for Information Security, G.F.O.: Position Paper on Quantum Key Distribution. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Crypto/Quantum_Positionspapier.pdf
20. for Information Security, G.F.O.: A Study on Implementation Attacks against QKD Systems. https://www.bsi.bund.de/EN/Service-Navi/Publikationen/Studien/QKD-Systems/Implementation_Attacks_QKD_Systems.html?nn=1100930
21. ISO/IEC: ISO/IEC 23837-2:2023. https://www.iso.org/standard/77309.html
22. Jain, N., Anisimova, E., Khan, I., Makarov, V., Marquardt, C., Leuchs, G.: Trojan-horse attacks threaten the security of practical quantum cryptography. New Journal of Physics **16**(12), 123030 (2014). https://doi.org/10.1088/1367-2630/16/12/123030
23. Kiktenko, E.O., Malyshev, A.O., Gavreev, M.A., Bozhedarov, A.A., Pozhar, N.O., Anufriev, M.N., Fedorov, A.K.: Lightweight Authentication for Quantum Key Distribution. IEEE Transactions on Information Theory **66**(10), 6354–6368 (Oct 2020). https://doi.org/10.1109/TIT.2020.2989459, https://ieeexplore.ieee.org/abstract/document/9076167

24. Kim, G., Park, D., Kim, H., Hong, S.: Side Channel Vulnerability in Parity Computation of Generic Key Reconciliation Process on QKD. In: 2021 International Conference on Information and Communication Technology Convergence (ICTC). pp. 257–261. IEEE, Jeju Island, Korea, Republic of (Oct 2021). https://doi.org/10.1109/ictc52510.2021.9620820, https://ieeexplore.ieee.org/document/9620820/
25. Lamas-Linares, A., Kurtsiefer, C.: Breaking a quantum key distribution system through a timing side channel. Optics Express **15**(15), 9388–9393 (Jul 2007). https://doi.org/10.1364/OE.15.009388, https://opg.optica.org/oe/abstract.cfm?uri=oe-15-15-9388, publisher: Optica Publishing Group
26. Liu, W.T., Sun, S.H., Liang, L.M., Yuan, J.M.: Proof-of-principle experiment of a modified photon-number-splitting attack against quantum key distribution. Physical Review A **83**(4), 042326 (Apr 2011). https://doi.org/10.1103/PhysRevA.83.042326, https://link.aps.org/doi/10.1103/PhysRevA.83.042326
27. Lütkenhaus, N.: Security against individual attacks for realistic quantum key distribution. Physical Review A **61**(5), 052304 (2000). https://doi.org/10.1103/PhysRevA.61.052304
28. Lydersen, L., Wiechers, C., Wittmann, C., Elser, D., Skaar, J., Makarov, V.: Hacking commercial quantum cryptography systems by tailored bright illumination (Mar 2011), http://arxiv.org/abs/1008.4593, arXiv:1008.4593
29. Mailloux, L.O., Hodson, D.D., Grimaila, M.R., Engle, R.D., Mclaughlin, C.V., Baumgartner, G.B.: Using Modeling and Simulation to Study Photon Number Splitting Attacks. IEEE Access **4**, 2188–2197 (2016). https://doi.org/10.1109/ACCESS.2016.2555759, https://ieeexplore.ieee.org/document/7456202/
30. Makarov, V.: Controlling passively quenched single photon detectors by bright light. New Journal of Physics **11**(6), 065003 (2009), https://iopscience.iop.org/article/10.1088/1367-2630/11/6/065003/meta, publisher: IOP Publishing
31. Mao, Y., Wang, Y., Huang, W., Qin, H., Huang, D., Guo, Y.: Hidden-Markov-model-based calibration-attack recognition for continuous-variable quantum key distribution. Physical Review A **101**(6), 062320 (Jun 2020). https://doi.org/10.1103/PhysRevA.101.062320, https://link.aps.org/doi/10.1103/PhysRevA.101.062320
32. Mehic, M., Maurhart, O., Rass, S., Komosny, D., Rezac, F., Voznak, M.: Analysis of the Public Channel of Quantum Key Distribution Link. IEEE Journal of Quantum Electronics **53**(5), 1–8 (Oct 2017). https://doi.org/10.1109/JQE.2017.2740426, https://ieeexplore.ieee.org/document/8014429/
33. Mehic, M., Rass, S., Dervisevic, E., Voznak, M.: Tackling Denial of Service Attacks on Key Management in Software-Defined Quantum Key Distribution Networks. IEEE Access **10**, 110512–110520 (2022). https://doi.org/10.1109/access.2022.3214511, https://ieeexplore.ieee.org/document/9919155/, publisher: Institute of Electrical and Electronics Engineers (IEEE)
34. NSF: FY 2026 Budget Request to Congress - NSF Budget Requests to Congress and Annual Appropriations | NSF - U.S. National Science Foundation. https://www.nsf.gov/about/budget/fy2026
35. Pacher, C., Abidin, A., Lorünser, T., Peev, M., Ursin, R., Zeilinger, A., Larsson, J.Å.: Attacks on quantum key distribution protocols that employ non-ITS authentication. Quantum Information Processing **15**(1), 327–362

(Jan 2016). https://doi.org/10.1007/s11128-015-1160-4, http://arxiv.org/abs/1209.0365, arXiv:1209.0365 [quant-ph]
36. Park, D., Kim, G., Heo, D., Kim, S., Kim, H., Hong, S.: Single trace side-channel attack on key reconciliation in quantum key distribution system and its efficient countermeasures. ICT Express **7**(1), 36–40 (Mar 2021). https://doi.org/10.1016/j.icte.2021.01.013, https://www.sciencedirect.com/science/article/pii/S2405959521000138
37. Peev, M., Pacher, C., Alléaume, R., et al.: The secoqc quantum key distribution network in vienna. New Journal of Physics **11**, 075001 (2009). https://doi.org/10.1088/1367-2630/11/7/075001
38. Renner, R.: Security of quantum key distribution. Ph.D. thesis, ETH Zurich (2005). https://doi.org/10.3929/ethz-a-005115027
39. Rosenpass: Rosenpass, https://rosenpass.eu/
40. Rubio García, C., Rommel, S., Takarabt, S., Vegas Olmos, J.J., Guilley, S., Nguyen, P., Tafur Monroy, I.: Quantum-resistant Transport Layer Security. Computer Communications **213**, 345–358 (Jan 2024). https://doi.org/10.1016/j.comcom.2023.11.010, https://www.sciencedirect.com/science/article/pii/S0140366423004012
41. Sajeed, S., Chaiwongkhot, P., Huang, A., Qin, H., Egorov, V., Kozubov, A., Gaidash, A., Chistiakov, V., Vasiliev, A., Gleim, A., Makarov, V.: An approach for security evaluation and certification of a complete quantum communication system. Scientific Reports **11**(1), 5110 (Mar 2021). https://doi.org/10.1038/s41598-021-84139-3, https://www.nature.com/articles/s41598-021-84139-3
42. Scarani, V., Bechmann-Pasquinucci, H., Cerf, N.J., Dušek, M., Lütkenhaus, N., Peev, M.: The security of practical quantum key distribution. Reviews of Modern Physics **81**(3), 1301–1350 (2009). https://doi.org/10.1103/RevModPhys.81.1301
43. Sushchev, I.S., Bulavkin, D.S., Bugai, K.E., Sidelnikova, A.S., Dvoretskiy, D.A.: Trojan-horse attack on a real-world quantum key distribution system: Theoretical and experimental security analysis. Physical Review Applied **22**(3), 034032 (Sep 2024). https://doi.org/10.1103/PhysRevApplied.22.034032, https://link.aps.org/doi/10.1103/PhysRevApplied.22.034032, publisher: American Physical Society
44. Tankovic, A., Dervisevic, E., Voznak, M., Mehic, M., Kaljic, E.: Performance Analysis of ETSI GS QKD 014 Protocol in Multi-User Environment. In: 2024 23rd International Symposium INFOTEH-JAHORINA (INFOTEH). pp. 1–5 (Mar 2024). https://doi.org/10.1109/INFOTEH60418.2024.10495984, https://ieeexplore.ieee.org/document/10495984/, iSSN: 2767-9470
45. Trunk, F., Neumann, J., Naegele-Jackson, S.: Quantum Key Distribution Testbeds. WiN-Labor Recherche (2024), https://www.win-labor.dfn.de/files/2024/07/WiN-Labor-Recherche_QKD-Testbeds.pdf

Appendix

The data from ten runs of dropping the `update_key_distribution` message is shown in Figure 7. The attack outcome is consistent, but the pre-attack timing is not: the attack window begins only after the nodes have reached stable operation, and this occurs anywhere between 169.6 s and 345.4 s after trace start. Thus some runs settle quickly, whereas others spend substantially longer synchronizing after a fresh reboot. Run 2, which we use as the representative trace in the main

paper, reaches the attack only after 173.6 s and then shows the cleanest victim-side collapse: Alice remains at 0.870968 keys/s during the 62 s attack window while Bob drops to 0 keys/s, yielding a 55-key final mismatch over the full trace (209 keys on Alice versus 154 on Bob). Across the sweep, Runs 6 and 7 exhibit brief spikes on Alice during the attack window, and Runs 1, 7, and 10 even show slightly higher Alice attack-window rates than their baselines. This variance indicates that the non-targeted side can continue producing keys with small burstiness differences, but the attack effect on Bob is stable in every run. The keypool traces confirm this: Bob generated either zero or one key during the attack window in all 10 runs, corresponding to a mean 99.3% drop and 61–63 s zero-key streaks.

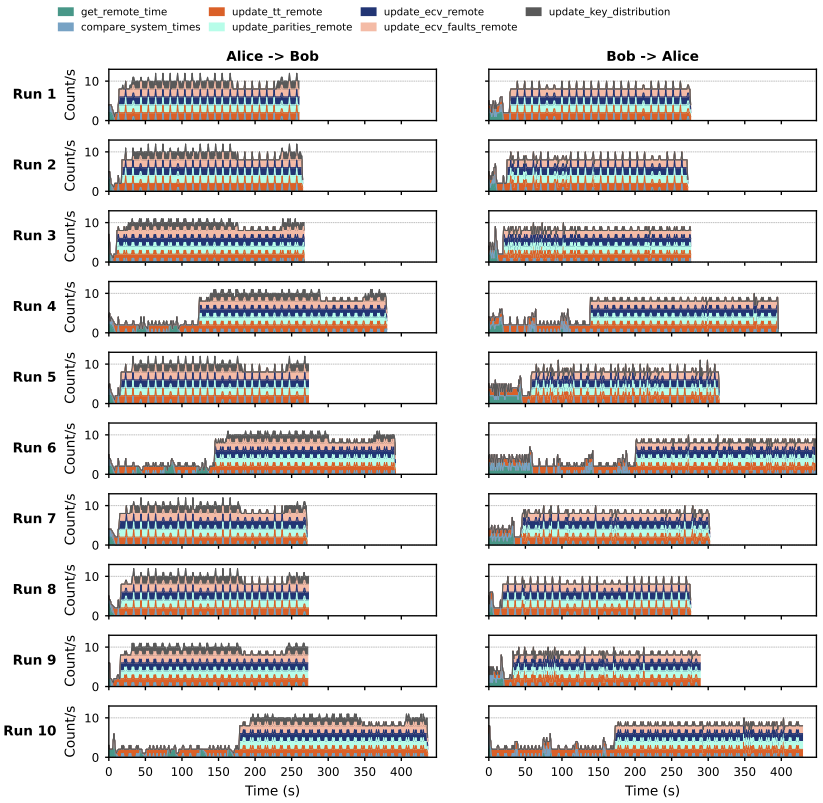

Fig. 7: Ten independent runs of selectively blocking `OKMS` messages containing key identifiers (`key_id`). The pre-attack synchronization period varies substantially across runs, but once the drop phase starts, Bob's key generation consistently collapses for about one minute while Alice continues near baseline, sometimes with brief spikes.

The data from sweeping delays of the `update_tt_remote` message from 50 ms up to 500 ms in 50 ms steps is shown in Figure 8. The sweep reveals a clear threshold effect. For 50, 100, and 200 ms, the largest observed `update_key_distribution` gaps are only 1.51 s, 1.03 s, and 1.02 s, respectively, so the attack does not work in a sustained way. The 150 ms case is the closest near-miss: it produces a visible 10.01 s interruption and 10 s zero-key streaks on both sides, but still remains below the 20 s sustained-disruption threshold. At 250 ms the behavior changes sharply. From that point onward the attack works reliably: at 250, 300, and 400 ms both nodes drop from roughly 0.91–0.93 keys/s baseline to about 0.016 keys/s during the attack window, with 60–61 s zero-key streaks and only one residual key generated per side. The 500 ms case is more severe still: the disruption expands to 78 s, Bob drops to 0 keys/s, and both sides take about 10 s after the attack ends to produce the first new key again. Consistent with the timelines, once the threshold is crossed the error-correction related messages (`update_ecv_remote` and `update_ecv_faults_remote`) and `update_key_distribution` disappear; at 500 ms, `update_parities_remote` also stops and the nodes fall back into synchronization.

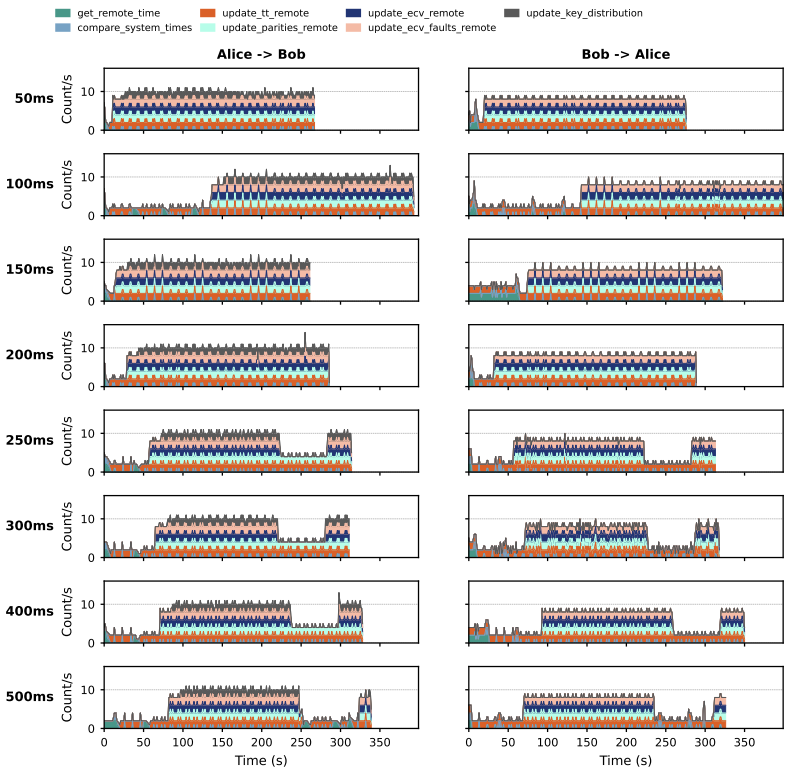

Fig. 8: Delay sweep for `update_tt_remote`. Delays of 50–200 ms cause only transient disturbances, with 150 ms producing the largest near-miss. Sustained failure first appears at 250 ms and remains stable through 400 ms, while 500 ms further suppresses protocol activity and pushes the nodes back into synchronization. The key-generation collapse was verified from the Alice and Bob keypool traces.