Article

Post-Quantum Migration of the Tor Application

Denis Berger, Mouad Lemoudden and William J. Buchanan *

Blockpass ID Lab, Edinburgh Napier University, Edinburgh EH10 5DT, UK; denis@denisb.net (D.B.);
m.lemoudden@napier.ac.uk (M.L.)
* Correspondence: b.buchanan@napier.ac.uk

Abstract: The efficiency of Shor's and Grover's algorithms and the advancement of quantum computers implies that the cryptography used until now to protect one's privacy is potentially vulnerable to retrospective decryption, also known as the harvest now, decrypt later attack in the near future. This dissertation proposes an overview of the cryptographic schemes used by Tor, highlighting the non-quantum-resistant ones and introducing theoretical performance assessment methods of a local Tor network. The measurement is divided into three phases. We start with benchmarking a local Tor network simulation on constrained devices to isolate the time taken by classical cryptography processes. Secondly, the analysis incorporates existing benchmarks of quantum-secure algorithms and compares these performances on the devices. Lastly, the estimation of overhead is calculated by replacing the measured times of traditional cryptography with the times recorded for Post-Quantum Cryptography (PQC) execution within the specified Tor environment. By focusing on the replaceable cryptographic components, using theoretical estimations, and leveraging existing benchmarks, valuable insights into the potential impact of PQC can be obtained without needing to implement it fully.

Keywords: Tor; post-quantum cryptography; NIST; onion routing

1. Introduction

1.1. Background

Shor's algorithm [1] has shown that our existing asymmetric (public key) methods are at risk, and Grover's algorithm [2] outlines a risk to symmetric key methods. As the Tor network uses cryptography as its base for security, we thus need to understand its possible migration path. Released in 2002, Tor currently has about two million estimated users [3]. It aims to fight censorship and surveillance, allowing users to browse anonymously. The definition of Tor's anonymity is described by Dingledine and Syverson [4] as the unlinkability between the user's identity and their actions. Their paper highlights the "not perfect" side of the definition, explaining that it aims to prevent traffic analysis to a certain degree. Tor's mode of operation mainly relies on three types of "proxies", through which the user connection goes. The client selects an entry node, middle node, and exit node, forming a path to the targeted resource. For each node, a key exchange happens between the client and the node, starting with the entry relay and then extending to the other nodes. Once the key exchange has occurred, data are re-encrypted at each step of the circuit. They undergo three layers of encryption until they reach the exit node, where the traffic is decapsulated to reach its final destination.

The Tor network is often associated with facilitating illegal activities, primarily through *onion services*—formerly known as hidden services. However, metrics collected by the Tor

paper reveal a discrepancy between the proportion of onion service traffic and that of the total network bandwidth. While unlawful content is hosted on onion services, this constitutes only a small fraction of the overall activity enabled by the Tor network (see Appendix A). OSs are a way to present content within the network itself without having to leave Tor. They conceal the physical location of the server and are accessible exclusively through the network using addresses ending in ".onion". Since the implementation of Version 3, these 56-character alphanumeric strings are created by encoding the service's long-term public key (32 bytes Ed25519), checksum, and version ("x03" by default) in Base32 [5] #224. Dingledine and Syverson [4] introduced a new mechanism called rendezvous-point relying on a DH key exchange where the user and OS establish a shared secret through Diffie–Hellman, with a "rendezvous" relay connecting the circuits from both parties without learning their identities or reading the data transmitted. Yet, since version 0.3.2.1—alpha—onion service cryptography has been improved by replacing the schemes SHA1, DH, and 1024-RSA with SHA3, Ed25519, and curve25519. Today, the main usage of cryptography in the network, as shown in Tables A1 and A2 relies on RSA, ECC, and AES. The latter, as a symmetric encryption algorithm, is less vulnerable to QC. An attacker equipped with a sufficiently advanced quantum computer could, in theory, compromise these cryptographic schemes to impersonate relay nodes, decrypt confidential content, and forge digital signatures, thereby creating documents that appear authentic to recipients. Gidney and Ekerå [6] estimate 20 million required physical qubits to break RSA-2048 encryption and highlight a considerable decrease in this estimation from 2015 to 2021, thus encouraging a faster adoption of PQC.

1.2. Purpose

On the 13th of August 2024, NIST [7] released three long-awaited PQC standards, including the CRYSTALS-Kyber KEM and CRYSTALS-Dilithium signature schemes (renamed "ML-KEM" and "ML-DSA"), both based on lattice problems. Another lattice-based standard publication, with FALCON to follow. On the other hand, NIST more recently announced a new timeline, banning RSA and ECDSA starting in 2035 [8]. Amid the wave of new publications in the field, the following question persists:

How can one evaluate its relay performance now as a reference point for later?

The four core aims of this paper can be summarised in the following points to assist in answering the latter research question.

- First, based on the literature analysis, this paper intends to identify a set of appropriate schemes for Tor post-quantum cryptography migration.
- To set up a functioning Tor network comprising multiple physical nodes in order to measure it and evaluate its performance at scale.
- Thirdly, identifying and measuring the cryptographic processes within Tor software.
- Finally, the scope of this research is to estimate a theoretical overhead value by incorporating the results of the previously defined objectives with post-quantum benchmark performances made on the devices used.

2. Literature Review

Tor specifications are supplied by proposals, allowing its implementation to grow and adapt to attacks over time. This literature review focuses on the cryptography features of the Tor paper. Firstly, the network's cryptography design must be explored to understand the technical environment and constraints. The Section 2 of this review will go through the main schemes used, reviewing the relevant cryptosystems and their place within onion routing in order to identify the ones that could be broken by a cryptographically relevant QC and their potential replacement candidates.

2.1. Tor Design and Architecture

In 2004, Dingledine and Syverson [4] addressed the disadvantages of the initial design by introducing features that have remained fundamental to this day. One of them is directory authorities: they are a small group of "trusted nodes" reaching a common agreement on the network state. In fact, Tor is not a fully decentralised, peer-to-peer network. This consensus allows Tor clients to have a global view of the network and aims to prevent malicious relays from manipulating its topology. Dingledine and Syverson [4] explain how these servers provide this information by sharing a directory signed with their long-term keys. Yet, the paper's technical design has been improved many times since. For instance, shortly after, the TAP was introduced [9] and strengthened the cryptographic rigour by formalising a circuit-based approach.

2.1.1. Circuits and Handshake

Circuit creation relies on the directory authorities in the very first step of the process when a given client (OP) chooses the first relay. As it needs a list of reliable relays, it will fetch the consensus document from the directory authorities, providing necessary data on the relays, such as their public keys, roles, supported protocols, and bandwidth [10]. Figure 1 below shows a high-level view of the circuit creation process. A circuit is typically composed of four channels: OP to R1, R1 to R2, R2 to exit node (RN), and RN to destination. After a circuit is established, the OP securely shares a unique symmetric key with each relay in the circuit. Additionally, each R exchanges a distinct circuit identifier with the nodes adjacent to it within the circuit, facilitating secure and efficient communication between nodes without revealing the entire path. A channel is defined as an encrypted link established directly between two relays or between a client and a relay, implemented as TLS sessions over TCP.
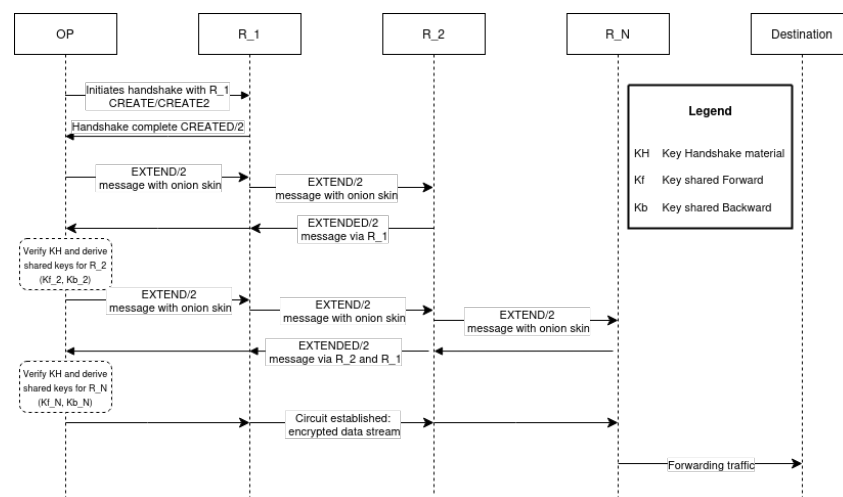

Figure 1. Tor circuit creation. The /2 represents the version, as their older specifications have become obsolete [10].

Tor relies on several cell types: cells, shown in Figure 1, are responsible for circuit creation. During channel negotiation, CERTS cells are used to describe the keys that a Tor instance is claiming to have. It also provides certificates to authenticate that those keys belong to long-term key(s) that uniquely identify a relay.

In standard TLS, the authentication is usually only necessary on the server side as the client identity is not important; this also applies to the first connection of a Tor circuit. Yet, the three other channel establishments need mutual authentication as the nodes must prove the genuineness of their affiliation in the circuit [11].

NTOR, the key exchange protocol currently used by Tor, introduced significant improvements over the TAP protocol [12]. It reduces the computational overhead by employing ECC, offering faster key exchange and smaller data transmissions. Also, NTOR strengthens FS, preventing decryption, even if long-term keys are compromised. The paper reflects on the challenge of One-Way Authenticated Key Exchange (1W-AKE) and how it applies to anonymity. In 1W-AKE, one party (the client) authenticates the other (relay) and remains unauthenticated. Goldberg et al.'s design [12] offers to authenticate the relay to the client by using the relay's public key and performing a key exchange via Curve25519 Diffie–Hellman.

Gosh and Kate [13] address the limitations of NTOR in light of quantum algorithms while maintaining forward secrecy, calling their 1W-AKE implementation HybridOR. It combines lattice-based cryptography, Learning With Error (ring-LWE), with the current DH assumption, maintaining compatibility with the current infrastructure.

Schanck et al. [11] proposed a hybrid design of the ntor circuit-extension handshake demonstrating a practical implementation using NTRU Encrypt. Schanck et al. [11] permits the incorporation of any number of KEMs, allowing the other quantum-resistant algorithms to be integrated more easily without rebuilding the entire protocol. They compare the performance of this hybrid handshake with the main Tor's handshakes (tap and ntor) and the one from Gosh and Kate. Computation time (in microseconds, μs) and the communication overhead, the number of bytes transmitted between the client and server, are shown for an instantiation with ntruees443ep1. NTOR remains the most efficient in terms of both overhead and time taken. They also display the proportion of the total handshake time that is spent on the client-side operations relative to the overall handshake time. ntor equally shares the handshake time, the hybrid reaches 74%, and Gosh–Kate 67%.

Gosh–Kate protocol's implementation by Schanck et al. [11] showed, on average, a smaller computation time (900 μs) but a larger number of bytes (1344). On the other hand, their hybrid design heavily relies on the client role (661 μs for client init and 74%), which could increase latency if the client's device has limited resources. However, their hybrid design implementation reduces the number of computational steps and amount of data exchanged. The evaluation is limited to NTRUEncrypt only; comparing it with other quantum-robust schemes would shed light on the efficiency of the overall hybrid implementation. They highlight a major obstacle: the size of the circuit-extension handshake "CREATE" cells is limited to 505 bytes (in NTOR), while ntruees443ep1 requires 693 bytes [11]. The work resulted in two distinct specification proposals: one aiming to widen the cell size [5] #249, and the other to enable the hybridisation of the ntor protocol and a KEM. The first one has been superseded by proposal #340, introducing the sub-protocol "RelayCell", which focuses on cell packing and fragmentation. The second, Ref. [5] #263, was made obsolete by proposal #269; created by the authors of the papers and Tor's developers, it takes the incorporation of different post-quantum KEMs further and emphasises compatibility with Tor's handshake.

Currently, the Tor code includes four different circuit-extension handshakes [10]:

- "CreateFast" deprecated (unauthenticated, non-forward-secure) handshake, which was previously used for the first hop of each circuit.
- The ntor handshake [12].
- The onion service ntor handshake variant allows each party to encrypt data (without forward secrecy) after the first message. Clients have used it since version 3 of the OS protocol to encrypt data in the introduction and rendezvous cells [5] #224.
- The ntor v3 also permits each party to encrypt data at the cost of FS, enables the client to send an authenticated encrypted message within its onion skin and allows the relay to send an encrypted and authenticated reply as part of its response [5] #332.

In 2019, Lauer et al. [14] proposed a 0-RTT handshake relying on puncturable KEMs to achieve a lower latency design than ntor; yet, the paper shows that it can result in higher computational overhead on certain low-power devices. The authors claim to achieve "immediate" FS, which appears to meet Tor's need for perfect FS, as the immediate variant mandates the deletion of ephemeral keys instantaneously. This approach could facilitate key management and allow the integration of post-quantum KEMs within the Tor handshake.

2.1.2. Relay and Link Layer

Also called onion skinning, the Relay "layer" is one of the most important of Tor's protocols. At the origin OP, a symmetric key is shared with each relay in the circuit using a telescoping key exchange protocol. During the circuit creation, as shown in Figure 1, instead of encrypting an entire circuit in one go, the circuit is built incrementally, with the client negotiating session keys with one node at a time. The OP encrypts the message multiple times, starting with the key for the final relay and moving backwards. When a message is transmitted, each intermediate relay decrypts one layer using its symmetric key, revealing the next hop or, for R_N, the destination. The message is always padded to a fixed size, preventing traffic analysis based on message length. This approach ensures that no single relay has a full view of the communication path, enhancing anonymity. However, according to Degabriele and Stam [15], due to its use of AES-128 in counter mode for each layer of encryption, Tor's relay protocol is susceptible to tagging attacks, where a malicious entry and exit node can tamper with and detect changes in the data to de-anonymise the user. This is exacerbated by Tor's low-latency design, which prioritises performance over stronger cryptographic guarantees.

Rogaway and Zhang [16] introduce the concept of Onion Authenticated Encryption, including indistinguishability from random bits and end-to-end authenticity verification as critical security measures.

While Degabriele and Stam [15] assess the relay protocol under CCA, identifying potential metadata leakage or improper handling of intermediary nodes, Rogaway and Zhang's findings [16] highlight that Tor's current protocol fails to meet these security benchmarks, specifically showing vulnerability to tagging attacks due to its reliance on counter-mode AES with an absence of a mechanism for authenticity checking throughout the relay path. They suggest improving integrity validation by applying robust authenticated encryption with associated data (AEAD) schemes, which would ensure the detection of tampering at an intermediary (middle node) layer. Moreover, they propose to add layer-specific nonce usage and key diversification across encryption layers against tagging attacks. Globally, their work calls for a protocol restructuring, which would not be solved by the implementation of PQC. Since Grover's algorithm [2] demonstrates limited parallelisation capabilities, the quantum threat is not estimated to be significant to asymmetric cryptosystems; thus, increasing the key size is expected to provide sufficient security [17].

Concerning the link layer, TLS ensures authentication, integrity, and encryption of data in transit between nodes. Its speed is a crucial metric in the Tor protocol for the user experience as it directly impacts circuit creation latency (every new Tor circuit requires a fresh TLS handshake between nodes) and the network scalability, as a higher TLS connection capacity allows it to handle more users efficiently. At present, Tor's TLS avoids session resumption for additional security and relies on stateless connections to avoid potential state-carrying [10]. Hence, treating each connection independently makes it harder for one to track connections or identify patterns. The newer version omits client certificates and uses a single-element, non-distinctive certificate chain to avoid detection by deep packet inspection systems, mimicking HTTPS traffic. TLS renegotiation was added but

later moved to encrypted data records in version 3 to improve anti-blocking features and limit observable TLS signatures.

Tor aims to cease using TLS 1.2 [5] #294, as 1.3 incorporates significant improvements such as the handshake design, particularly with its 1-RTT, meaning only one round-trip time is required until the first application message is sent, decreasing latency [18]. In a "classical" key exchange context, the assumption relies on Diffie–Hellman, while with PQC integration, it typically includes KEM, implicating a revision of the 1-RTT mode. In 2020, Schwabe et al. [19] introduced "KEMTLS", a fully post-quantum modification of TLS 1.3, replacing signature-based authentication with KEM. Their design drastically cuts the computational costs associated with PQC signatures while achieving IND-CCA but not strict full-forward secrecy, for which they declared achieving levels 1, 3, and 5 according to the Noise protocol framework [20]. The same authors released a variant of their previous KEMTLS named KEMTLS-PDK [21], yet in the context of Tor, this revised version relying on pre-distributed keys and partially cached information might not be a straightforward fit if connections could be linked back to cached or pre-shared keys.

As for hybrid instantiation, the main method involves concatenating the classical and PQC key materials (public keys/ciphertexts) and treating them as a single element. Stebila et al. [22] draw a transitional capable design with backwards compatibility, allowing one device (client or server) to still use traditional schemes if it is not "hybrid-aware", therefore resulting in three possible scenarios: hybrid handshake, client downgrade, or server downgrade to classical only. Their design aims to keep TLS 1.3 features such as high performance or 1-RTT.

The IETF draft declares the main security property of KEMs as IND-CCA2, correlating with the first motivation of CECPQ2 [23]—the second version of the TLS 1.3 key exchange protocol developed by Google and Cloudflare. In fact, Langley [23] highlights that, in TLS, managing confidentiality is more straightforward than authenticity, as encryption keys are independently negotiated for each session, while post-quantum authenticity presents a greater challenge, as it needs to integrate with the existing certificate authority and certificate ecosystem, making it considerably more complex to establish and maintain. Tor's TLS connection layer for relays and bridges, like the standard TLS, uses X.509 certificates to authenticate themselves during handshake exchanges [10]. Concerning the OS, the ".onion" address, being a hash of the service's public key, confirms the server's identity itself without the need for a CA. Firefox, on which Tor's browser is based, has already released the option to use X25519Kyber768 for TLS. Tor's TLS implementations have been slightly modified from the standard to meet its needs for compatibility and against traffic analysis. The ongoing research and deployments give a strong base to migrate the network's link layer; if the circuit-layer protocol ensures the main security and anonymity guarantees, in Tor, TLS is a necessary complement and has mitigated critical bugs in the past [5] #294. The relay and circuit-extend protocols, as Tor-specific protocols, seem to need a deeper analysis to build a migration design. However, the relay layer does not appear highly endangered [24]. The stream protocol has not been reviewed due to its significant reliance on the other layers and its minimal impact on onion routing compared to them.

2.2. Schemes

Based on research papers and Tor's official source code, this section outlines the used schemes and their potential replacement candidates, each addressing specific security requirements. NIST defined the security level of PQC from 1 to 5. The first level is equivalent to the 128-bit traditional security level (strength of AES-128).

TAP [9] implemented a standard of at most 80-bit security [25], through the use of RSA with a 1024-bit modulus and DH 1024 in its key exchange. Since the NTOR [12] paper, Tor

cryptography publications [11,13] have agreed on the necessity of at least a 128-bit security standard. Tor has started to adopt Curve25519 (also written as x25519) in most key-enabled cryptography processes; nonetheless, RSA is still used as an identity key in the relay layer (see Table A1) and in TLS 1.2.

Baseri et al. [24] compiled a comparative inventory of the vulnerable protocols and evaluated the corresponding mitigation strategies. The paper presents a risk assessment framework that includes multiple known attack strategies based on the STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege) model. They differentiate the risk into two categories: the algorithmic level, where a mathematical assumption is not strong enough, and the protocol level, where the implementation presents a vulnerability, such as side-channel or fault-injection attacks. Hybrid implementation, aiming to protect the new cryptography migration from these risks, is highlighted in the context of a crypto-agility approach. However, due to Tor's infrastructure, the computational load, packet size, and bandwidth requirements need to be considered.

Table 1 breaks up Tor's layers by schemes (currently used), their security level, key size in bytes, and role. When looking at the replacements, the benchmarking of PQC schemes tends to highlight the overall performance of lattice-based algorithms: Saber and Kyber as KEMs, and Dilithium as a signature scheme, in constrained environments, such as running on a Raspberry Pi [26]. Nevertheless, Saber was judged to be less efficient than Kyber in a broader range of environments [27].

Table 1. Cryptography schemes of Tor's main layers. For clarity, the "directory authorities" section is presented within the relay layer; however, it may be more appropriately classified as a distinct category. For further specifications, refer to Figure A4.

Layer	Protocol / Version	Scheme	Security Category	Role
Circuit-extend	ntor	HMAC-SHA256	2	Message integrity in handshake, authenticity
		HKDF	2	Key derivation from shared secret
		Ed25519	unsafe	Relay identity verification
	ntor v3 and OS-ntor	SHA3-256	2	Integrity, authenticity of messages
		SHAKE-256	4	Keying material generation with randomness
	All handshakes	AES-128-CTR	1	Message content encryption during handshake
		X25519	unsafe	Key agreement
Relay	Standard circuits	SHA-1	unsafe	Cell header integrity check (being phased out)
		SHA-256	2	Cell message integrity check
		AES-128-CTR	1	Relay cell encryption in layers
		Ed25519	unsafe	Relay identity verification and key signing
	OS v3	AES-256-CTR	5	Encryption for data integrity and identity validation
		SHA-256	2	Key derivation, message integrity, handshake validation
		Ed25519	unsafe	Signature for consensus data
	Directory authorities	SHA-256	2	Integrity through hashes of consensus documents and descriptors
		RSA2048 or 3072	unsafe	Authority identity key, Directory server public signing key
Link	TLS 1.2	RSA2048	unsafe	Identity authentication
		DHE	unsafe	Ephemeral key exchange with forward secrecy
	TLS 1.3	AEAD only	1	Channel encryption; e.g., AES-GCM, ChaCha20-Poly1305, having built-in integrity checks
	All	SHA-256 and above	>2	Message integrity, authentication, key derivation
		ECDHE	unsafe	Ephemeral key exchange with forward secrecy
	Internal	X.509	unsafe	Certificate-based identity and link verification

The following first covers quantum-safe key exchange schemes, followed by a review of PQC signature algorithms.

2.2.1. Key Exchange

For key exchange, Tor almost always uses X25519 (Tables 1 and A1). It is known for its efficiency and is used in multiple PQC hybrid instantiations [22,28]. The work of Schanck et al. [11] is also a hybridisation of X25519 with ntruees443ep1 and HKDF-SHA256. In proposal #269 [5], a hybrid handshake alternative, it mainly differs from ntor in the computation of the authentication tag and key derivation.

Two lattice-based schemes have been instantiated as examples in the proposal:

- NTRUE KEM with the EESS443EP2-specific parameter set, estimated at 128-bit security for both traditional and quantum-resistant settings. The design declares the maximum

message size as $m = 49$ bytes, the KEM public key length as $|PK| = 615$ bytes, and a KEM ciphertext size of $|CT| = 610$ bytes.

- NewHope KEM is declared with $|PK| = 1824$ and $|CT| = 2048$.

The chosen NTRU parameter EESS443EP2 is interesting as it proposes relatively small keys; Cheng et al. [29] showed the performance of the EES parameters in constrained devices. Yet, according to Bernstein et al. [30], this classic NTRU parameter falls under a category vulnerable to automorphism-based exploits and other attacks such as lattice reduction. In contrast, NTRU Prime is still of interest from a security point of view, achieving IND-CCA2; it is a third-round NIST candidate with its "Streamlined" and "LPRime" variants. The preferred parameter is Streamlined, sntrup761, as a balance of performance and security; it approximately achieves NIST category two post-quantum security with an estimation of 2^{153} based on the Core-SVP.

Another KEM scheme that was not selected for standardisation by NIST in 2022 is FrodoKEM. For randomness generation, its main implementations rely either on AES-CTR or SHAKE XOF, with the latter offering better performance across a broader range of hardware types, as demonstrated by Bos et al. [31]. FrodoKEMs claimed categories 1, 3, and 5 of NIST PQC security according to the given parameters; FrodoKEM-640 targets the first level and FrodoKEM-976 the third. In Table 2, their key and ciphertext sizes are significantly larger compared to the sntrup761 setting except for the shared secret key. Furthermore, ML-KEM presents multiple advantages. It is indistinguishable from a chosen ciphertext attack, assuming that D-MLWE is intractable and that G, H, and J are random functions. Additionally, it maintains IND-CCA2 by a quantum adversary able to make both classical and quantum queries (in superposition) to G, H, and J [32,33]. Above all, it shows encouraging performances in comparison with other schemes [26,34]. Table 3 shows the three distinct standardised formats of Kyber and their corresponding key sizes.

Table 2. Sizes (in bytes) of FrodoKEMs and sntrup761's keys and ciphertexts.

	Encapsulation Key	Decapsulation Key	Ciphertext	Shared Secret Key
sntrup761	1158	1763	1039	32
FrodoKEM-640-AES	9616	19,888	9720	16
FrodoKEM-640-SHAKE	9616	19,888	9720	16
FrodoKEM-976-AES	15,632	31,296	15,744	24

Table 3. Sizes (in bytes) of keys and ciphertexts of ML-KEM [33].

	Encapsulation Key	Decapsulation Key	Ciphertext	Shared Secret Key
ML-KEM-512	800	1632	768	32
ML-KEM-768	1184	2400	1088	32
ML-KEM-1024	1568	3168	1568	32

In hybrid implementations, the increase in handshake size tends to lead to higher bandwidth usage and latency during the handshake process. However, the overall performance heavily depends on the underlying mathematical assumption and its implementation. The Open Quantum Safe [35] benchmarking displays the performance of the schemes on x86_64 architecture according to the operations (key generation, encapsulation, decapsulation) per second and per CPU cycle. It ranks HQC-128 and Kyber512, both achieving NIST level 1 security, with higher key generation per second. Table 4 shows a summary of OQS measures for the schemes of interest. Overall, taking into consideration size and performance

attributes, Kyber/ML-KEM appears to be the highest performer. It is followed by HQC, which reaches faster key generation per second and maintains good key encapsulation and decapsulation speeds despite larger key and ciphertext sizes (see Table 5). The streamlined NTRU Prime, sntrup761, comes next with smaller keys and ciphertext sizes than HQC-128, ML-KEM-768, or FrodoKEM-640. Claiming at least the second NIST security category, it represents a trade-off between performance, security, and bandwidth. FrodoKEM tends to fall behind due to reduced operational efficiency, considering its large key sizes and security level.

Table 4. KEM performance comparison. Operations per second per algorithm, reference code type (unoptimised), 2024-04-02, OQS [35].

	Keygen/s	Keygen (Cycles)	Encaps/s	Decaps/s	Security Category
HQC-128	6719.33	371,892	3571.00	2349.00	1
HQC-192	2855.00	875,523	1465.00	1000.00	3
Kyber512	23,348.00	106,974	18,970.00	16,118.00	1
Kyber768	13,660.67	182,899	11,630.00	10,096.33	3
sntrup761	115.55	21,636,049	2308.00	818.00	2
FrodoKEM-640-SHAKE	345.99	7,224,951	282.67	280.33	1

While Kyber is a derivation of the Kyber PKE algorithm using FO transform, HQC uses a variant of the FO transform called HHK, allowing HQC to achieve IND-CCA2 security [36]. Among a few other minor modifications, the FIPS 203 standard for Kyber and ML-KEM specifies a variant of the FO transform for the encapsulation and decapsulation mechanism; in this work, the Kyber benchmarks are considered for evaluating ML-KEM performance.

The code-based HQC scheme displays excellent performances, yet its large keys and ciphertexts (Table 5) may require further adaptation to integrate it within Tor protocols.

Table 5. Sizes (in bytes) of keys and ciphertexts of HQC [36].

	Encapsulation Key	Ciphertext	Secret Key
HQC-128	2249	4487	56
HQC-192	4522	9042	64
HQC-256	7245	14,485	72

The 512-byte limit affects the amount of data that can be included in the handshake process. The currently open proposal #340 [5] aims to implement a cell packing, optimising cell usage, and a fragmentation mechanism allowing for larger cryptographic keys. Based on the design of proposal #269, the work on PQC migration toward a hybrid handshake [11,13], and the attributes of the observed schemes, ML-KEM-768 and sntrup761 appear to fit the circuit-extension requirements. HQC may be an alternative if larger keys can be implemented easily. This work ignores strong cryptosystems such as McEliece, which imposes a substantial key size and, therefore, communication overhead.

The current cell size limit remains an obstacle to the hybrid or full PQC migration for all layers. Concerning the relay layer, no key exchange has been processed, but the challenge also concerns signature schemes. Regarding the link layer, it produces key exchange within its TLS implementation. RFC [37] standard defines the handshake with three core stages: server parameters, key exchange, and authentication. In a PQC environment, the second phase would rely on KEMs. The third is reviewed in the next section. Currently, Tor's TLS

uses P256 ECDHE (and still has legacy support for secp224r1) for handshakes and Ed25519 for server identity, known as link keys [10]. Indeed, ML-KEM-768 appears to be the selected parameter for the newly adopted designs [22]. Færøy [38], in a fork of C Tor, experimented with the integration of hybrid key exchange TLS with `X25519Kyber768Draft00`.

In 2020, Paquin et al. [39] presented the performance trade-offs of several schemes, including hybrid ECDH-P256-Kyber512. Compared to the classical p-256 curve, the hybrid scheme shows an increase in packet loss and completion time. Their work highlights the effect of higher RTTs, where the hybrid scheme demonstrates greater degradation; in their conclusion, the authors discussed how the enlargement of the MTU might improve TLS establishment performance. In 2024, the post-quantum TLS survey by Alnahawi et al. [40] points out the significant impact of large keys, using FrodoKEM as an analogy. FrodoKEM takes 2.7× the time of a Kyber handshake and 2.53× the time of a typical classical handshake with ×25519. They state that the combination of Kyber512 (NIST security level 1) and ×25519 took 1.25× the time of a traditional handshake, while NIST level 3 of this hybridisation has a very close performance (1.28×). Finally, the paper provides an overview of how the work of [19] differs from other designs in its approach of achieving AKE as the pure PQC solution KEMTLS. Indeed, KEMTLS is achieving a form of 1W-AKE, as the server is authenticated to the client without DSA by using a long-term KEM public key for encapsulation, with the server responding via an encapsulation based on the client's ephemeral KEM public key.

Therefore, achieving a 1W-AKE means the client does not need to authenticate itself. A pure KEM 1W-AKE implementation is interesting in the context of the future circuit-extension handshake. Recently, Pan and al. [41] introduced a one-way Verifiable Weak FS notion and presented the first lattice-based tightly FS AKE via key confirmation in the classical random oracle model (using the lattice-based protocol from Pan et al. [42]), showing that the OW-VwFS can be transformed *tightly* to FS using key confirmation in the random oracle model (ROM). The circuit-extension handshake requires FS [11], and in the near future, a fully KEM-reliant 1-WAKE handshake could potentially replace ntor or hybrid-motor. Based on [19]'s benchmarks (NIST Round 3 estimation results), it could be as fast as or faster than the current handshake encryption.

2.2.2. Signature

This subsection takes a closer look at the potential signature mechanisms that could be integrated within Tor's different layers. The current 1W-AKE relies on public key cryptography. The hybrid circuit-extension design presented in proposal #269, inspired by Schanck et al.'s work [11], does not integrate post-quantum DSA. It maintains the usage of ECDH primitives (signing with the server's Curve25519 public key) for authentication. This is based on the assumption that the session negotiation itself is not vulnerable to quantum attacks. The hybrid protocol derives session keys and authentication tags from the shared secrets of both the ECDH shares and KEM-derived secrets, allowing authenticity verification without an explicit digital signature during session establishment. This approach avoids the additional computational cost and larger communication footprints of the standardised DSAs and the significant modifications to the Tor protocol that their incorporation would lead to. Although the recently introduced ML-DSA and SLH-DSA (Table A4) have larger signature sizes compared to the stateful hash-based schemes XMSS and LMS (Table A5), they offer the advantage of reducing state management complexity and lowering operational overhead; thus, they are better candidates if used in the handshake. The Falcon scheme, like ML-DSA, is based on lattice; it proposes even smaller keys (Table A6) but considerably slower runtimes; Dilithium, with security level 2 parameters, generates key pairs approximately 178 times faster per second than Falcon512, which corresponds to security level 1 (Table 6). SLH-DSA is the only stateless hash-based standard

so far, offering robust security without relying on new mathematical problems. However, it generates large signatures (from nearly 8 KB to 50 KB) and performs slower signing than its lattice-based competitors; SPHINCS+ at security level 3 achieves about 14 signing operations per second, as shown in Table 6.

Table 6. PQC performance comparison. Operations per second per algorithm, reference code type (unoptimised), 2 April 2024, OQS [35].

	Keygen/s	Keypair (Cycles)	Sign/s	Verify/s	Security Category
Falcon-512	53.26	46,938,518	176.55	17,246.00	1
Falcon-1024	17.91	139,565,458	80.56	8341.00	5
Dilithium2	9486.00	263,444	2099.33	8752.33	2
Dilithium3	5184.33	482,134	1320.00	5519.00	3
SPHINCS+-SHA2-128f-s	558.81	4,473,508	23.86	404.73	1
SPHINCS+-SHA2-192f-s	382.08	6,542,247	14.52	270.91	3

The relay layer is heavily reliant on the circuit-extension handshake in terms of authentication, yet it also depends on Tor's own certificate mechanism. Tor does not rely on standard CA-issued certificates for its core functioning, avoiding reliance on centralised CAs. Signing is performed with Ed25519 keys; their format differs for certificates used by authorities to sign their identity key [10]. *CERTS* cells are at least 104 bytes when containing a single certificate of 96 bytes (32 of certified key and 64 of signature), with its expiration date of 4 bytes plus 4 bytes representing the number, type, and length of certificates in the cell. This number is approximate, as an extension can be added to bundle the signing key along with the certificate, which adds up to at least 36 bytes. Therefore, 140 bytes would represent one certificate and its signing key, which fits into the current absolute maximum fixed size of 512 bytes. ML-DSA, the smallest recently standardised post-quantum equivalent, has a total of 3732 bytes for the public key and signature sizes (Table A4).

In regard to onion services, which also rely on a public key for authentication (Table A2) to prevent the linking of descriptors, they use a blinded version of the identity key that changes at regular intervals instead of using the identity key directly [10] #224. This allows us to hide the original identity key while still authenticating without linking the real key directly. The feature is vulnerable to quantum attacks where a capable adversary could forge the OS signature and potentially redirect queries addressed to the service. Regarding its migration to PQC primitives, Eaton et al. [43] assessed four schemes, including Dilithium, which outperforms the others both in signing and verification but also reaches a close result of its unblinded counterpart.

TLS signatures can be divided into two categories: "online" signatures of messages in the handshake protocol and static ones of certificates in the certificate chain; the "statics" allow a greater signing time as the computation takes place in advance. Typically, the root certificate signs the intermediate CA certificate, which itself signs the leaf certificate. The latter is used to sign the transcript during the handshake; the other signatures are static. The certificate chain size gives practical insight into assessing the overhead. Kampanakis and Childs-Klein [44] estimate the authentication data sizes of ML-DSA-44 and ML-DSA-6 to be 14 KB and 19 KB, respectively (including the intermediate CA certificate). Their work on the increased latency caused by the certificates is based on the estimated sizes of the chain; the observed chains are built with RSA, not actual ML-DSA certificates. The paper states a 32% handshake time increase compared to the 2.5 KB chain. In Tor TLS, the certified key type varies from Ed25519 to a hash (SHA256) of an X.509 certificate. Since

the hash itself is not directly reversible, this part of TLS authentication is, therefore, less exposed to quantum attacks. Yet, the underlying X.509 certificate remains dependent on the public key primitive used. While standards drafts are proposing to include the recently standardised SLH-DSA [45], its performance stays far behind Dilithium and Falcon [46]. Kampanakis and Kallitsis [47] proposed a backwards-compatible mechanism to omit the intermediate CA certificate, allowing lighter and faster PQC TLS handshakes.

Currently, an IETF draft is in progress to extend ACME challenges to validate ".onion" domains through a Tor-compatible mechanism; the draft by Misell [48] presents a new challenge type, "onion-csr-01", while still incorporating the "http-01", "dns-01", and "tls-alpn-01" challenges. Relying on the broader web PKI, aside from the potential privacy concerns for the onion service, remains sensitive to quantum attacks.

If proposal #340 is implemented as specified, the digest field, responsible for checking the cell integrity, should reach a size of 14 bytes, allowing the integration of larger hashes (currently SHA-1). Although hash functions are less directly threatened by quantum algorithms than PKE schemes, it has been shown that Grover's algorithm can be used to reduce the time required for preimage attacks on hash functions like SHA-1, SHA-2, and SHA-3, as well as minimise the time needed for hash inversion in $O(\sqrt{n})$, halving the security of the function [49].

Ultimately, NIST [50] is still looking for a signature scheme and has declared that some second-round candidates have undergone minimal or no formal cryptanalysis in published research. Thus, potential signature schemes may still reveal themselves in future research and deployment as a match for Tor's requirements. For instance, FAEST, which achieves faster key generation than ML-DSA, displays smaller key sizes and relies on the mature security of SHA3 and AES, yet its signatures are more than twice the size of those of the latter lattice-based scheme at their respective minimum security levels [51].

2.3. Reflection

The attraction towards hybrid models is justified by the potential fallback to classical encryption in case a given quantum-robust scheme fails. Even if they have been cautiously examined, new PQC schemes could potentially be broken by an adversary. As an analogy, SIKE reached the fourth round of NIST competition before being proven insecure [52]. To avoid solely relying on relatively young solutions, hybrids appear to outweigh the faster, fully PQC solutions (KEMTLS). As for the ease of instantiation, the concatenation method of algorithms allows for the simpler implementation of these hybrids.

The circuit layer comes up as the most urgent layer to migrate to PQC. Implementing a transitional hybrid scheme would require a KEM and a DSA mechanism. The second migration would concern the link protocol. In comparison, TLS has benefitted from a greater number of deployments and more extensive research. While OS reuses these fundamental blocks, it incorporates additional mechanisms such as introduction, rendezvous points, and hidden service descriptor encryption, which have separate cryptographic processes. Public key cryptography is used in both cases: for key exchange during the introduction and rendezvous points; and authentication for both the points and descriptors.

Regarding Tor's components that should be tested with quantum-robust KEMs, four stand out:

- The circuit-extension handshake;
- Onion skin;
- Link layer TLS;
- OS Introduction, rendezvous points.

As for the ones that can be evaluated using DSA, the following stand out:

- Consensus document;

- Relay identity authentication;
- Link TLS handshake authentication;
- Encrypted descriptor.

Section 2.1.2 highlighted the fact that Tor already faces cryptography challenges that will not be solved by the integration of quantum-resistant schemes. The latter implies a potential redesign of certain underlying protocols beforehand, as proposed by Degabriele and Stam [15]. Overall, the review emphasised the various cryptographic layers and examined their interactions. It appears technically feasible to transition to PQC standards incrementally, layer by layer. Regarding the necessity of migrating the encapsulated TLS layer to PQC when its outer layer is already quantum-resistant, the recommendation is affirmative, as it is preferable to ensure comprehensive quantum security.

On the other hand, Rahman et al. [53] proposed an integration of QKD for symmetric key exchange without relying on trusted nodes. The design presents a quantum relay to facilitate the key exchange between client and nodes, yet this approach would require a specific quantum communication infrastructure, such as quantum repeaters, which are still in development and not widely available. It can be observed that the diversity of KEM standards is currently limited compared to that of DSA (ML-DSA, SLH-DSA, XMSS, and LMS).

3. Methodology and Design

This section seeks to develop the implementation design based on insights gained from the literature review. This research takes an inductive approach, collecting data to build a theory regarding the impact of PQC migration in Tor.

3.1. Technical Requirements

On a security level, the implemented protocols will need to achieve IND-CCA2 and at least the first NIST PQC security category. From a size perspective, none of the schemes observed fit in the current cell size. At the data-link layer, to reduce fragmentation, it is preferable for public key sizes to fit in the Ethernet MTU of 1500 bytes. Only ML-KEM-512,768 or sntrup761 for KEM and Falcon-512 or ML-DSA-44 for signatures meet this requirement. Yet, as previously noted, Tor cell body lengths are currently limited to 509 bytes and will be adjusted to 493 bytes, except for "DATAGRAM" messages, which aim to support the UDP-over-Tor proposal [5] #339. Fragmentation of PQC keys across several cells appears inevitable (Tables 2, 3, 5, and A4–A6). Reassembly, verification, and potential retransmissions will be necessary. ML-KEM and Falcon encapsulation and public keys, respectively, fit in two Tor cells. In terms of actual runtime, ML-DSA significantly outperforms Falcon.

Regarding relays, memory ($\geq$1 to 1.5 GB of RAM per node), disc storage ($\geq$200 MB), and bandwidth minimal requirements ($\geq$16 Mbps), as defined by the Tor paper, will need to be met in the hardware equipment used.

Arti offers a safer and faster development than C Tor due to Rust's design. Generally, the Rust language does not outperform C, yet it shows close performance [54]. Initiated in 2020, the Rust implementation takes a simpler approach compared to the older C version. At present, a performance comparison of the two implementations is difficult to make as Arti is not fully finished. For instance, relays cannot be run just yet [10]. Thus, this work focuses on C language Tor implementation and cryptographic libraries.

3.2. Network Architecture

A star topology will be formed around a central switch device. For the sake of mimicking the key roles, the network needs to provide enough diversity; therefore, nine nodes will operate together. Two machines will act as directory authorities. Four nodes

will serve as guard relays, two as exit relays, and one will be dedicated to functioning as a middle hop and OS. Given the size and configuration of the network, there will be no dedicated directory caches, and nodes will endorse multiple flags, such as the onion service descriptor "HSDir" flag.

3.3. Local Benchmark

The implementation will start by looking at the tools relevant to measuring the Tor network and running them within our scenario. A packet capture will permit the observation of the time of transmitted frames with the corresponding traffic type.

Tor uses `sbws` (Simple Bandwidth Scanner) and OnionPerf to measure its performance [3]. Sbws usually relies on the download and upload of files through the circuits to measure the benchmark. This method can be mimicked manually using tools such as `wget` or `curl`. The implementation will look for a method to efficiently measure the circuit round-trip latencies and circuit build times.

3.4. Code Performance

With regard to the three layers, the list of C files below represents the core Tor protocol operations; they will be assessed on the performance of their most computationally expensive functions.

- The `circuit_establish_circuit` function in `circuitbuild.c` handles the process of selecting paths and managing the handshake protocols between nodes to establish circuits.
- `onion.c` is responsible for the creation, encoding, and parsing of cells (CREATE, CREATED, EXTEND, and EXTENDED). It invokes handshakes.
- `channel.c` implements the transmission, reception, and processing of cells across different connections.
- `relay.c` manages RELAY cells, including their encryption, decryption, forwarding, and processing for routing data.

The following code files will be evaluated to weigh the cryptography operations they represent:

- `onion_ntor.c` implements the ntor handshake.
- `onion_ntor_v3.c`, as specified in proposal 340 [5], requires the enhanced handshake, ntor v3, for cell fragmentation.
- `relay_crypto.c` handles relay cell payload encryption, decryption, integrity verification via cryptographic digests, and the initialisation of symmetric keys.
- `hs_ntor.c` is closer to ntor v3 than ntor; it is the implementation of the handshake for onion services.
- `tortls.c` is the main TLS implementation file in Tor.
- `channeltls.c` is the only instantiation of channel abstraction. It handles the v3+ link handshake, certificate verification, and cell processing over OR connections.

In Tor's "crypto" code directory, the deprecated fast handshake file `onion_fast.c` is ignored. A frequency analysis will be conducted to associate the roles of nodes with the functions they execute the most.

3.5. PQC

The following algorithms will be considered viable candidates: ML-KEM-512 and Falcon-512 at security level 1; sntrup761 and ML-DSA-44 at security level 2; and ML-KEM-768 at security level 3. The Open Quantum Safe C library [35] provides implementations of the considered schemes. The speed tests for signature and KEMs will be run on different

devices with their code. Regarding the performance evaluation of the classical scheme, the OpenSSL `speed` tool will be used. Combinations of the selected schemes will be made to evaluate their timing with their security level. All of them will be run on at least two different devices. The advertised results summarised in Sections 2.2.1 and 2.2.2 will be taken into account.

3.6. Reflection

The implementation of this outline should be helpful in shedding light on the impact of post-quantum cryptography in Tor, according to the assumption that if a given cryptographic process represents a proportion of Tor's total runtime, then an increase in latency will lead to a rise in at least the total timing.

4. Implementation

Firstly, the benchmark techniques will focus on the local network before targeting cryptographic timings (circuit-extend, onion service, and TLS). Secondly, the measurements obtained will be compared with their potential PQC substitutes.

4.1. System Setup

The hardware used for this implementation consists of nine Raspberry Pis used as onion relays: two model 4b with 8 GB of RAM, and seven model 5 (two with 8 GB and five with 4 GB of RAM). A Thinkpad laptop X1 with i7-10850H CPU and 32GB of RAM is used as a client. They are connected using two-metre-long straight-through Ethernet cables and a Cisco Catalyst 2900 switch (Figure A5). It should be noted that a Raspberry 4b with 4 GB of RAM is added for stability when needed.

Each Raspberry runs Ubuntu server 24.04, and the Lenovo personal computer runs Kali Linux 2024.3. The switch runs DHCP to avoid accessing the Raspberries through monitors, but the IPs are configured statically; the switch's minimal configuration can be found in Listing A1. Each node runs Tor version 0.4.8.10.

The Nginx web server is used for the hidden service. It publishes a simple HTTP page, as shown in Figure A6. It is resolved and reached by launching the Tor browser using a separate Tor daemon (see Figure A7). Tor software is designed to run within a large network with a large number of nodes; the implementation of a local testing network with a small number of nodes meets multiple constraints. To ensure anonymity, Tor code discards guards on certain conditions.

4.2. Local Network Benchmark

The average bandwidth connectivity between devices was measured to be 93.8 Mb/s using the iPerf network performance measurement tool [55]. The network was configured as described in Section 3.2; Chutney [10], a Python tool for testing the Tor network locally, was used to generate torrc configuration files for each node (Listings A2 and A3).

The torrc files were slightly adapted, but most of the settings remained identical to Chutney's templates. Listings A4 and A5 display the used parameters for directory authorities and guard relay. For instance, due to the size of the network setting, `PathsNeededToBuildCircuits` was decreased to allow more flexible circuit building. The client configuration also pointed to the local directory authorities and contained the setting for testing networks, which adjusted default values, as described in "man tor" (Listing A6). The conflux setting was disabled, and the nodes were specified in the configuration to facilitate the onion.

If accessing the internet through the local Tor nodes, from the browser's point of view, the timing displayed is heavily influenced by external variables such as broadband and thus is not relevant to evaluating the Tor implementation itself. However, considering the

timing difference between the same test request made from Firefox to the internet directly and the Tor browser going through the local nodes, it can be inferred that passing through the established Tor channel only adds a few hundred milliseconds (Figure A7). Tor's traffic is mostly web [3]; measuring the time taken to visit an internet resource adds a relevant data point of the overhead introduced by the class. The laptop was configured to perform NAT between its ethernet and WLAN interfaces, and its IP was set as the default gateway on all nodes. DNS points to the Google 8.8.8.8 address.

4.2.1. Pcap Analysis

This configuration allows sniffing of the first packet sent to the guard relay and the first packet leaving the exit node (Figure A8) (to the duckduckgo.com domain without search content). For this circuit exiting by node ".23", the client sends its Tor encrypted packet to the entry node at 15:29:50.360220332. The exit node sends a DNS request at 15:29:50.414655443 and its decapsulated `ClientHello` message at 15:29:50.462253502 (see Figure A9). Therefore, the timing to exit the circuit is 102 milliseconds, including the DNS requests (which are influenced by broadband), and 54 milliseconds without.

When trying a similar request, the measured time, with "DNS overhead", is 110 milliseconds and approximately 60 milliseconds without (see Figure A11). The browser process is killed between each measure, and its cache stays empty. In this capture, the entry and exit nodes are the same, but the circuit is different, as shown in Figure A10. For a third circuit and another web request, we obtain 4 ms from the first TCP sent to the DNS request made by the exit node and 57 ms if waiting for the `ClientHello`. This timing further stresses the need to automate the measurement process to derive metrics from a larger sample size.

4.2.2. Simple Bandwidth Scanner

As for the `sbws` tool, it is currently developed for targeting the real Tor or a locally hosted (single-device) Tor simulation. Here, it was built locally, and its source code file, "generals.py", was slightly modified to force the generated Tor daemon to point at the local directory authorities. This method avoids including the processing of the browser. To use this tool, a new node was added to act as a web server. It published the 1 GiB file accessible on port 443 with HTTPS (TLS is required on the target in this scenario to run `sbws`). The laptop was configured as a DNS server, allowing the nodes to resolve the address "local.tor" to IP 192.168.1.24. The self-signed SSL certificate was updated as trusted on the nodes, and the option `ServerDNSDetectHijacking 0` was added to their torrc configuration. The tool stops after one loop when detecting that it is assessing a testing network and creates a raw text file, which is used to generate "v3bw" files. As for OS, the "Attachstream" functionality is not currently supported for .onion addresses.

Gauging the circuit's timings against the direct and non-encapsulated traffic would bring a relevant comparison. Thus, the following subsection aims to achieve this.

4.2.3. Curl Measurements

First, we compare the latency of passing through the network with sending curl requests directly to the external server. Listings 1 and 2 display the curl requests run from the client passing through its Tor daemon with SOCKS5.

Listing 1. Curl GET request to duckduckgo.

```
curl --socks5-hostname 127.0.0.1:9050 -w "%{time_total} " -o /dev/null -s https://
    duckduckgo.com
```

The Tor paper defines round-trip latencies as the time between sending the HTTP request and receiving the HTTP response header [3]. Here, this is replicated by the `time_total` variable in the curl command. This technique also permits us to reach the OS and to directly benchmark it.

Listing 2. Curl GET request to onion service.

```
curl --socks5-hostname 127.0.0.1:9050 -w "%{time_total} " -o /dev/null http://
    adg4jkv2xpciraegkj3rpcfrdnzlaeqnlt7a3wzylhzcvshzkerhacyd.onion
```

Furthermore, POST requests are also sent to the target address, allowing us to download and upload performances with a 1 GiB large file (just as sbws can achieve) and the average of a typical POST request. This is performed by adding curl parameters, as shown in Listing A8, for the evaluation of access to an external resource without relying on broadband internet. Figure 2 represents the average speeds measured with the 1 GiB file.
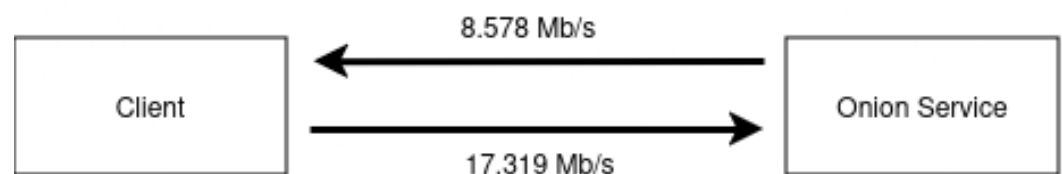

Figure 2. Average download and upload speeds of onion services.

Finally, to evaluate the overhead of TLS in the onion service, it is implemented and compared with HTTP.

4.2.4. Circuit Build Time

Using the `stem` library, Listing 3 allows obtaining timing metrics for a given number of built circuits. Two ways were implemented: the first was by waiting for the controller to mark the status of the circuit as built, and the second one was by checking its status at small intervals. The latter demonstrated greater consistency in its results.

Listing 3. Python script using the `stem` controller library.

```python
import time
import numpy as np
from statistics import mean, median, stdev
from stem import CircStatus
from stem.control import Controller

def benchmark_circuits(controller, numCircuits=1000):
    circuitBuildTimes = []
    for i in range(numCircuits):
        startTime = time.time()
        # Creating a new circuit and waiting until circuit is fully built
        circId = controller.create_circuit(await_build=False)
        while True:
            circ = controller.get_circuit(circId)
            if circ.status == CircStatus.BUILT:
                break
            time.sleep(0.001) # Checking the status as frequently as possible
        endTime = time.time()
        buildTime = endTime - startTime
        circuitBuildTimes.append(buildTime)
        controller.close_circuit(circId)

    if circuitBuildTimes:
        avgTime = mean(circuitBuildTimes)
        mdTime = median(circuitBuildTimes)
        mnTime = min(circuitBuildTimes)
        mxTime = max(circuitBuildTimes)
        sdTime = stdev(circuitBuildTimes) if len(circuitBuildTimes) > 1 else 0
        q1Time = np.percentile(circuitBuildTimes, 25)
        q3Time = np.percentile(circuitBuildTimes, 75)
        print(f''Metrics for {numCircuits} circuits:'')
        print(f''  Average build time: {avgTime:.4f} seconds'')
        print(f''  Median build time: {mdTime:.4f} seconds'')
        print(f''  Min build time: {mnTime:.4f} seconds'')
        print(f''  Max build time: {mxTime:.4f} seconds'')
        print(f''  Standard dev: {sdTime:.4f} seconds'')
        print(f''  First quartile (Q1): {q1Time:.4f} seconds'')
        print(f''  Third quartile (Q3): {q3Time:.4f} seconds'')
    else:
        print(''No circuits'')

if __name__ == ''__main__'':
    with Controller.from_port(port=9051) as controller:
        benchmark_circuits(controller, numCircuits=1000)
```

4.3. Cryptography Benchmark

To measure the cryptographic operations performance in terms of execution time, the relevant functions of the files described in Section 3.4 were executed with the `clock_gettime()` function, the results were returned as standard output, and the `RunAsDaemon` option needed to be disabled. The executable file was run on all devices. The functions were used at different frequencies according to the node's role.

Focusing on the traditional cryptography used, Table 7 presents the performance achieved by Raspberry Pi 4 and 5 (4GB) using OpenSSL.

Table 7. Benchmark of Tor's asymmetric schemes on Raspberry Pi. Average operation iteration per second.

	Sign/s	Verify/s	Model
Ed25519	2939.1	1327.0	4b
RSA2048	199.2	7492.9	4b
RSA3072	21.3	1209.6	4b
Ed25519	16,301.1	6846.1	5
RSA2048	281.0	10,289.0	5
RSA3072	90.1	4652.2	5
	Key Exchange/s		Model
X25519	1044.6		4b
X25519	5973.5		5

The first Tor protocol to invoke these cryptosystems is the circuit-extend. Lauer et al. [14] measured the time of the ntor handshake execution by the user at 0.30 ms and 0.10 ms by the OR. Yet, Schanck et al. [11] benchmarked a 0.000527 ms total performance for ntor computation. Here, a low-level approach was taken to estimate the ntor execution time: `onion_skin_ntor_create` computed the first client-side step of ntor in 0.13 ms, while the function `onion_skin_ntor_client_handshake`, which performs the ntor final client-side step, was executed in 0.77 ms on average by all nodes. The server-side steps of the handshake achieved an average of 2.1 ms altogether.

In the PQC migration context, the ntor v3 handshake is expected to be the standard, as previously mentioned in Section 3.4. It performed significantly better than the classical ntor with 0.67 ms on the client side and 0.63 ms on the server side.

Figure 3 shows the mean and median values of most measured functions; very small values have been excluded for the graph's clarity.
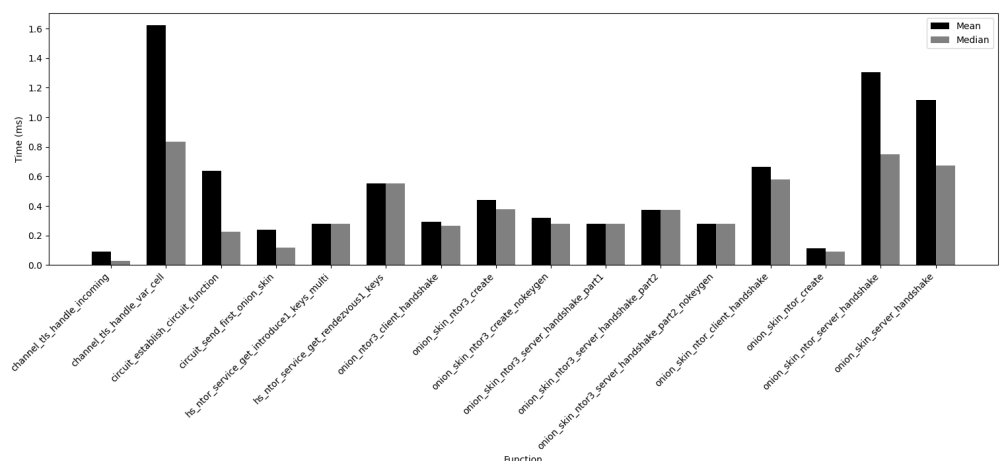
Figure 3. Execution speed per function.

4.4. PQC Runtimes

The schemes of interest were benchmarked using the OQS library to grasp the performance of the nodes against PQC schemes. Table 8 shows the average number of operations per second on Raspberry Pi (4GB of RAM) 4b and 5 for both KEMs and signatures. Here, Raspberry Pi 4 significantly under-performed compared to the results published by OQS. For example, ML-KEM-512 ran 3807 key generation operations per second, whereas

Kyber-512 achieved 23,348 keygen/s (Table 4). On the other hand, model 5 accomplished closer outputs.

Table 8. Benchmark of PQC schemes on Raspberry Pi. Average time (µs) per operation, code retrieved from liboqs [35].

	Keygen/s	Encaps/s	Decaps/s	Model
ML-KEM-512	3807.3	3462.0	2783.0	4b
ML-KEM-768	2539.0	3905.0	2571.5	4b
sntrup761	25.2	841.7	410.6	4b
ML-KEM-512	23,168.3	19,178.0	15,136.3	5
ML-KEM-768	13,976.6	11,900.6	9614.3	5
sntrup761	158.1	6067.0	3210.3	5
	Keypair/s	**Sign/s**	**Verify/s**	
Falcon-512	23.6	615.3	7866.3	4b
ML-DSA-44	2642.7	286.1	3213.9	4b
Falcon-512	95.4	3360.6	19,831.3	5
ML-DSA-44	8986.3	1885.0	8139.0	5

Based on the fact that a key exchange for a KEM requires key generation, encapsulation, and decapsulation operations,

$$T_{\text{KE}} = T_{\text{kengen}} + T_{\text{encaps}} + T_{\text{decaps}}.$$

Here, the fastest post-quantum KEM scheme performed slightly faster than X25519. The 4b model achieved 0.95 ms per classical KE exchange and 0.91 ms per ML-KEM-512 KE; nonetheless, model 5 achieved closer results with 0.167 ms per X25519 against 0.161 ms. The operations per second are displayed in Tables 7 and 8. On the client (x86_64 architecture), X25519 was benchmarked as KEM and it resulted in 22,839.0 keygen/s, 11,950.9 encaps/s, and 26,040.8 decaps/s, thus corresponding to a speed of 0.1659 ms per key exchange. Meanwhile, ML-KEM-512 produced 38,732.0 keygen/s, 33,241.3 encaps/s, and 36,170 decaps/s, showing a speed difference of 66% (between 0.1659 and 0.0835). From a runtime point of view, adopting ML-KEM-44 to replace Curve25519 key exchange on these devices would result in an acceleration of the overall process. Yet, this is the only scheme where X25519 is slower. On Raspberry model 4b, ML-KEM-768 showed a 14% execution time increase, while sntrup761 performed the KE operations in 43.3 ms, nearly 50 times slower than X25519. Nevertheless, on the client, the NTRU scheme was only six times slower than the classical key exchange with a timing of 1.005 ms/KE. Regarding DSA schemes, Edwards-curve 25519 signing remains the fastest of the standards, followed by Falcon-512, which is still, on average, 4.8 times slower. Moreover, Ed25519 key and signature sizes (32 and 64 bytes, respectively) stay significantly smaller than both post-quantum DSAs (see Tables A4 and A6), which introduces storage and transmission overhead. It should be noted that the need for flexible key management increases as the public and private keys of Falcon and ML-DSA differ, unlike the currently used Ed25519.

In terms of verifying operation speed, Falcon slightly surpasses RSA2048 on Raspberry Pi 4b and runs almost twice as fast in comparison to model 5. Without taking size and key pair generation into consideration, the fastest combination of quantum-robust signature cryptosystems, as a hybrid, would be Falcon-512 with Ed25519 at security level one. RSA is not considered, as Tor is trying to move away slowly from it. However, if one considers key sizes and key pair operations per second, ML-DSA with Ed25519 is a better choice, achieving the second NIST security category.

4.5. Reflection

The size of the network has consequently influenced the nodes' configurations, ensuring enough guards, which has been a core challenge in achieving correct circuit building. Logs have been highly valuable alongside Nyx, a Tor command-line monitor (Figure A12). The settings have mostly been manually deployed, and most could be easily automated with a script.

5. Evaluation

This section will evaluate the implementation and its outcomes in relation to the aims and objectives outlined in the initial research proposal.

5.1. Results

5.1.1. Bandwidth and Circuit Performance

For an average bandwidth of 93.8 Mbits/s between nodes, without passing through the circuits in the network, the mean time taken for a circuit to be built varies between approximately 44 and 60 milliseconds. The tools used did not allow accurate measurement of the circuit build time per hop but rather the entire circuit building. The `stem` scripts, however, may be modified to measure a single-hop latency. This was not performed here, yet the overall latency gave an approximation of the single-hop circuit build time. Their results are compared in Figure 4. The graph displays metrics from twelve different script executions over time. The upper row displays timings more spread out, yet faster than the lower one. They have been differentiated, as each script execution would either average a circuit time close to 44 ms or the upper 57 ms due to hardware differences in the chosen nodes.
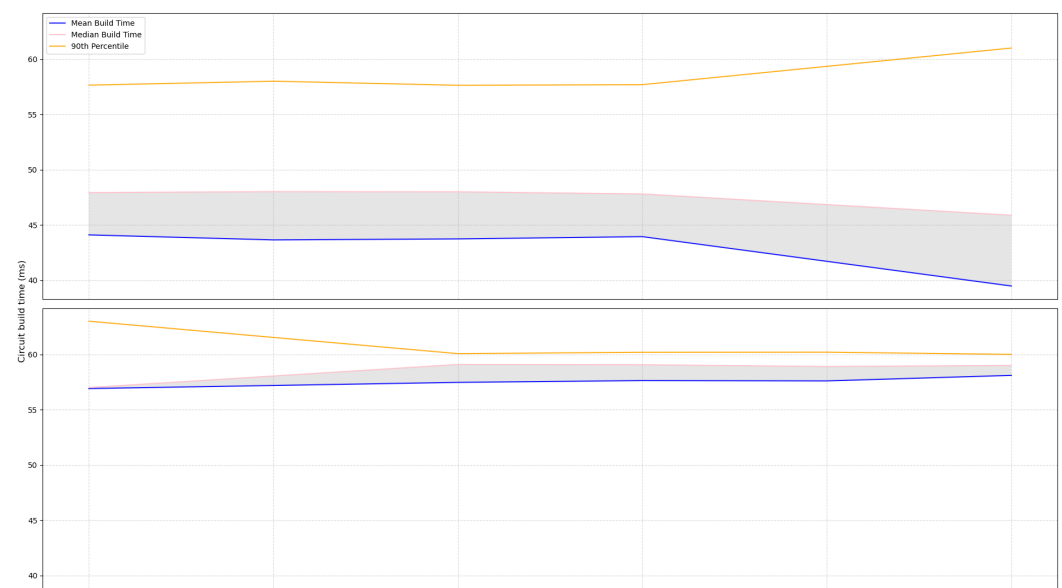

Figure 4. Circuit build time. The Q1 and Q3 measurements are omitted from the graph.

When using Tor circuits to fetch the "local.tor" HTML page (HTTPS), the upload speed was 3808.52 kB/s, and the download speed reached 9648.06 kB/s. Without passing through circuits, much closer values were obtained: an upload speed of 11,511.83 kB/s and a download speed of 11,410.24 kB/s. This implies asymmetrical bandwidth capabilities between paths; this is supported by the results obtained in the file transmission, as shown in Figure 2, with higher upload speeds on the exit path.

The Pcap analysis results show limitations in terms of accuracy; on the other hand, using a greater sample of curl timings data, relevant metrics were obtained. Table 9 displays

the median and average values of a single request according to the targeted source. The three targets were running Nginx web servers, all equipped with SSL certificates.

Table 9. Performance of web traffic through local Tor network. Total time (ms) per curl.

	Duckduckgo	**Local Webserver**	**Onion Service**	**Request Type**
Mean	262.9	136.7	174.6	GET
Median	260.8	142.4	166.0	GET
Mean	–	141.8	172.6	POST
Median	–	148.0	171.5	POST

The round-trip latency can be calculated accurately by subtracting the average latency of direct route requests from that of requests routed through Tor. Based on 1900 requests per source, fetching "duckduckgo.com" yields an overhead of 83.1 ms, while accessing the "local.tor" Nginx webserver incurs an additional latency of 82.4 milliseconds. Rounding up these values, the estimated latency overhead is approximately 83 milliseconds.

Regarding OS performance, the data exchange to fetch the test page (as shown in Figure A6) takes approximately 20 ms longer compared to the identically configured Nginx server. Knowing that OS requires going through three more hops than the external resource fetching it, the time per hop extension for fetching via OS is approximately 7 ms, while the circuit building time per hop ranges from 15 to 20 ms (based on stem script results). This could be explained by the optimised ntor handshake initiated in `hs_ntor.c`.

The bandwidth results obtained from `sbws` corroborate the initial bandwidth measurement of 93 Mbits/s made using iPerf. Notably, the first authority guard is excluded from this analysis as it introduces an error. In Figure 5, the first authority guard is omitted as it results in an error due to the fact that `sbws` uses it to fetch data in the network, corrupting its bandwidth assessment.
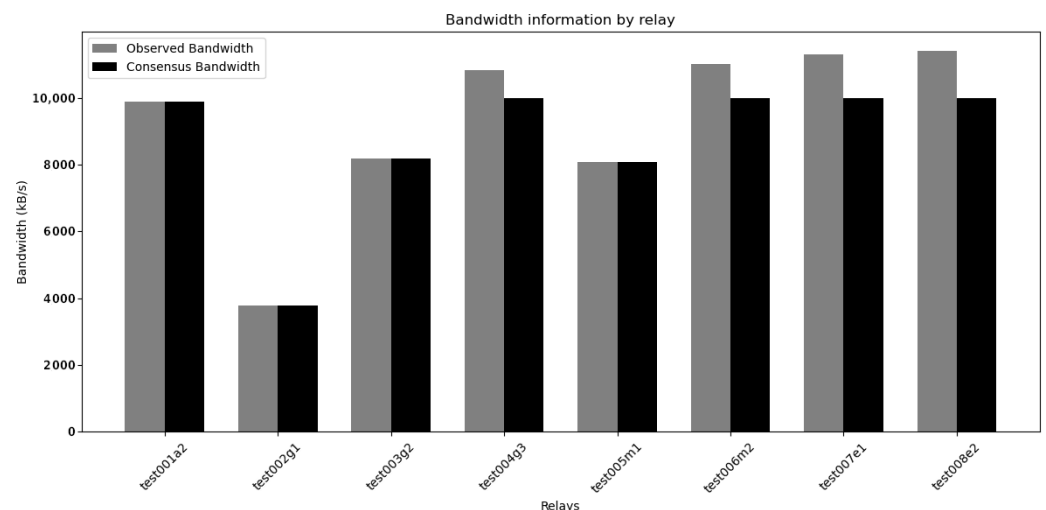

Figure 5. `sbws` upload bandwidth per node.

5.1.2. Cryptography Comparison

The comparison potentially demonstrates that the optimisation of the X25519 implementation allows constrained devices to execute its KE almost as fast as the ML-KEM-512 standard, while on a 32 GB machine, the post-quantum standard is able to perform at least 50% faster than its classical counterpart. In a similar manner, NTRU (sntrup761) performs much better on the client than it does on the Raspberry Pi. Memory usage should be compared for a deeper analysis. Falcon-512 is the fastest on constrained devices. However,

it represents a much larger latency on more powerful hardware (e.g., laptops). It should be noted that the architecture here—`aarch64` for the Raspberry Pis—should be taken into account; the OpenSSL executable for `x86_64`, used on the laptop, provided more accurate computations of each operation per scheme.

In terms of roles, the analysis of results, displayed in Figure 6, reveals distinct functional distributions among Tor nodes, highlighting the specialised responsibilities of each. Guard nodes show the largest proportion of iteration for encryption- and circuit-related operations `relay_encrypt_cell_inbound`, `relay_decrypt_cell`, and `circuit_package_relay_cell`, whereas the middle node, also acting as an OS, mainly displayed handshake-related function iterations. As expected, exit nodes primarily engaged in outbound traffic handling (`relay_encrypt_cell_outbound` and `channel_write_packed_cell`). Authority nodes have a smaller, consistent presence, reflecting their role in network management and verification rather than direct cell processing. Thus, onion service nodes evidently exhibited the highest usage of public key cryptography functions, followed by guard nodes, executing the most handshake-related operations.
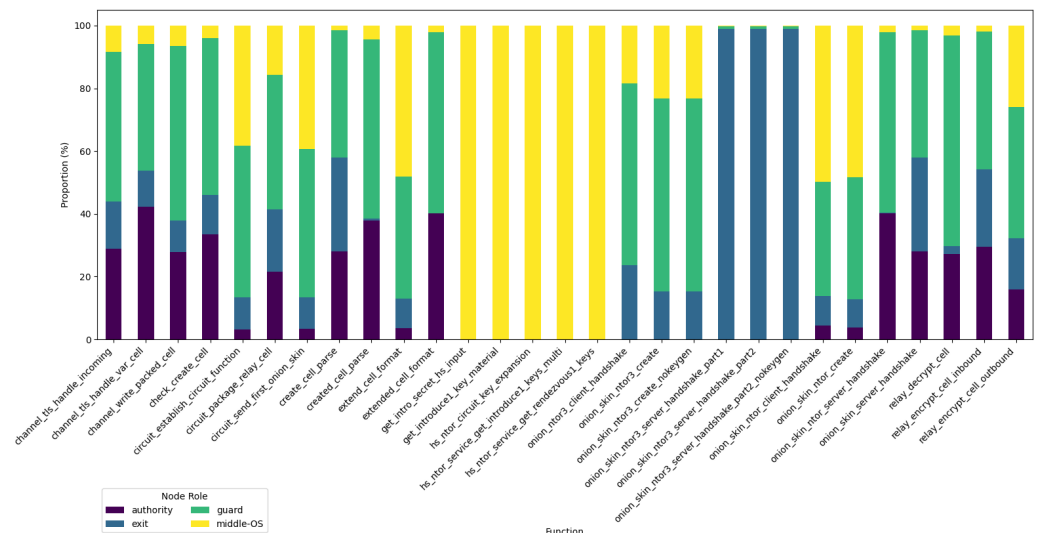

Figure 6. Proportion of iterations per relay role for a given function. In this experiment, OS and middle relay are the same devices.

5.2. Effectiveness

The network benchmarking process displays consistent results. Asymmetric latencies have been identified; however, they are not necessarily problematic. The real Tor network relies on a diversity of hardware, resulting in asymmetric bandwidth. Compared to advertised metrics (see Figure A3), the circuit round-trip latencies share common characteristics in their results, with a higher but faster onion latency compared to when directed to a public server. On the other hand, estimating a precise proportion of public key cryptography processes is difficult by solely analysing the timings of cryptographic functions.

Related Work

Zsolt et al. [56] highlight the impact of PQC on Tor but focus mainly on the scheme's performances and less on Tor software. Their work acknowledges the advantage of lattice-based KEM cryptosystems but presents SIKE as an optimal choice; it does not cover the efficiency of DSA schemes. In comparison, this paper benefits from a more advanced status in the NIST standardisation process. Evgnosia-Alexandra [57] gives a brief cryptographic analysis of potential candidates and emphasises hybrid Ring-LWE approaches. Yet, Evgnosia-Alexandra's paper also promotes SIKE as the appropriate choice to minimise

overhead. In contrast, this work dived into Tor's software performances in a constrained environment, evaluating the potential PQC load in a pragmatic manner.

5.3. Reflection

Based on the frequency analysis of C function execution, middle-OS and guard nodes appear to perform the most asymmetric cryptographic operations. It can be inferred that considering previous benchmarks of PQC algorithms on Raspberry Pis, their minimum hardware memory requirements may need to be increased. However, reducing the acceptable criteria could also lead to a reduction in network capacity.

6. Conclusions

Overall, this work has presented several methods to evaluate a local Tor network, which can be easily implemented and repeated in future testing environments. It also presented a reference point, providing results of these measurement techniques that one could consider in a similar experiment.

The key deliverables of the paper are as follows:

- Four algorithms have been demonstrated to be more appropriate than others: ML-KEM, and NTRU (sntrup761) as KEMs, followed by ML-DSA and Falcon as signature schemes.
- A local Tor network has been successfully deployed with multiple physical devices; it reached a Tor circuit-round latency of 83 ms for a non-encapsulated bandwidth mean value of 93.8 Mbits/s.
- The C code related to Tor cryptographic operations was timed. However, the relevance of the timings should be interpreted with caution within the context of this experiment.
- Using the obtained results alongside PQC performances, an accurate estimate of the overhead percentage cannot be determined. Numerous unclear variables are likely to influence the final latency evolution. This work has highlighted that the hardware characteristics of Tor nodes will be one of these factors.

Most of the raw results have been made publicly available at https://gitlab.torproject.org/diunisu/testing-network-results (accessed on 15 February 2025).

6.1. Limitations

The drawbacks of this work can be represented by three main points below.

- While trying to satisfy the low-bandwidth requirement, the selected schemes only cover the two first NIST security categories. Furthermore, they lack diversity in their mathematical assumptions, relying solely on lattice problems.
- The benchmarking is heavily subjective.
  Firstly, it is influenced by the technology used, such as the optimisation of cryptosystems based on a CPU architecture. In this context, this theoretical bare-metal Tor network simulation could be improved by using hardware generally used in the real Tor network. This would require more intrusive relay metrics and may go against security and privacy principles. Secondly, the timings are greatly influenced by the configuration. The torrc files were adapted to obtain a functional local network with the given number of nodes. Yet, another blend of settings could result in a more realistic implementation. Adding more nodes should improve the stability of the setup. Going further, the nodes could be distanced geographically. Planet Lab [58] may be an appropriate test bed. Additionally, the methods and tools employed in the measurement process can significantly alter the results; for instance, time is consistently taken as a reference point, and other measuring units, such as CPU cycles, are neglected.

- The estimation of the PQC overhead is only theoretical and misses a multitude of parameters, such as different traffic types or volumes. In a virtual environment, further testing with Shadow could provide valuable insights based on the real Tor traffic, while on physical machines, adapting or creating tools like OnionPerf to target local instantiations could also produce more relevant measures. In the future, Tor's software optimisations and restructuring of the underlying protocols will also impact network performance.

6.2. Future Work

Future work includes advancing the post-quantum circuit-extend handshake design, inter alia, by comparing a DSA-free handshake to a KEM authentication (still requiring DSA for static leaf signature). This should be followed by a software implementation draft to enable the practical evaluation of the integration of these PQC schemes. It would be particularly useful to assess its performance in a realistic scenario. Looking ahead, when the Arti code allows the running of relays, it will be helpful to note the difference in cryptography performance between C Tor and Rust in this context.

Glossary

Acronym	Meaning
ACCE	Authenticated and Confidential Channel Establishment
ACME	Automated Certificate Management Environment
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AKE	Authenticated Key Exchange
ARTI	A pure-Rust Tor Implementation
CA	Certificate Authorities
CCA	Chosen-Ciphertext Attack
SVP	Shortest Vector Problem
CRYSTALS	Cryptographic Suite for Algebraic Lattices
DH	Diffie–Hellman
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DSA	Digital Signature Algorithm
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie–Hellman
EES	Efficient Embedded Security
FALCON	Fast Fourier Lattice-based Compact signatures over NTRU
FO	Fujisaki–Okamoto transform
FS	Forward Secrecy
HHK	Hofheinz–Hövelmanns–Kiltz transform

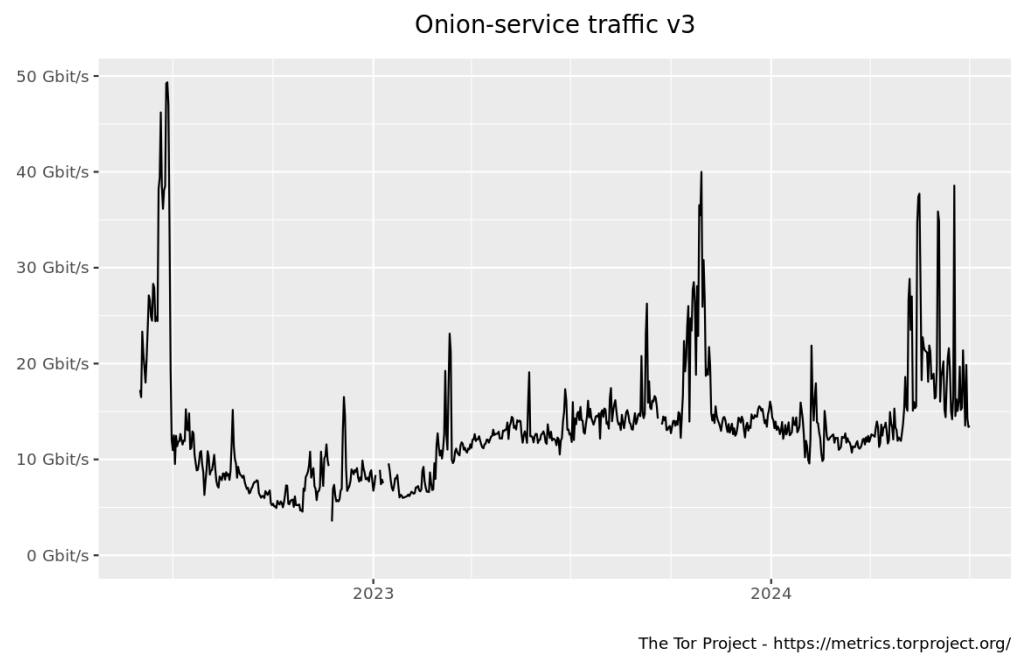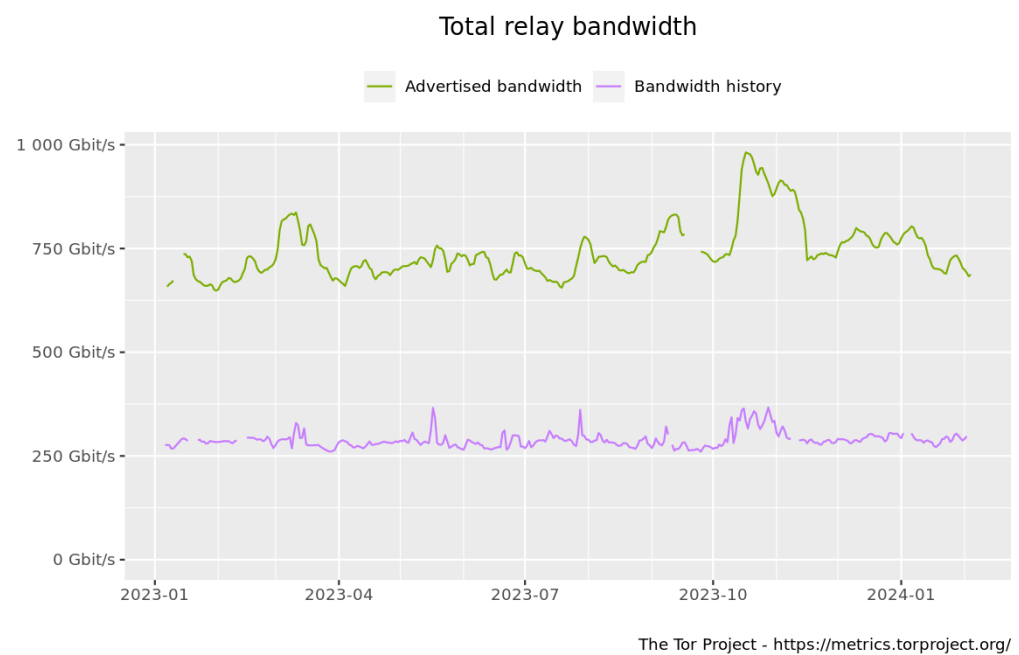|	
HQC	Hamming Quasi-Cyclic
IETF	Internet Engineering Task Force
IND-CCA	INDistinguishability under Chosen-Ciphertext Attack
IND-CCA2	INDistinguishability under Adaptive-Ciphertext Attack
IND-CPA	INDistinguishability under Chosen-Plaintext Attack
IP	Internet Protocol
KEM	Key Encapsulation Mechanism
LMS	Leighton–Micali Signature
MLWE	Module Learning With Error
ML-KEM	Module-Lattice-based Key Encapsulation Mechanism
ML-DSA	Module-Lattice-based Digital Signature Algorithm
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NIST	National Institute of Standards and Technology
NSA	National Security Agency
NTRU	Nth degree TRUncated polynomial ring
OSs	Onion Services
PDK	Pre-Distributed (public) Key
PFS	Perfect Forward Secrecy
PKE	Public Key Encryption
PQC	Post-Quantum Cryptography
PRNG	Pseudo-Random Number Generator
QC	Quantum Computer
QKD	Quantum Key Distribution
RAM	Random-Access Memory
RSA	Rivest–Shamir–Adleman
RTT	Round-Trip Time
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
TAP	Tor Authentication Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TOR	The Onion Routing
XMSS	eXtended Merkle Signature Scheme
1W-AKE	One-Way Authenticated Key Exchange

Appendix A

Onion-service traffic v3

The Tor Project - https://metrics.torproject.org/

Figure A1. V3 onion service traffic over 2023.

Total relay bandwidth

The Tor Project - https://metrics.torproject.org/

Figure A2. Total bandwidth traffic over 2023.

Circuit round-trip latencies to public server

The Tor Project - https://metrics.torproject.org/

Circuit round-trip latencies to onion server

The Tor Project - https://metrics.torproject.org/
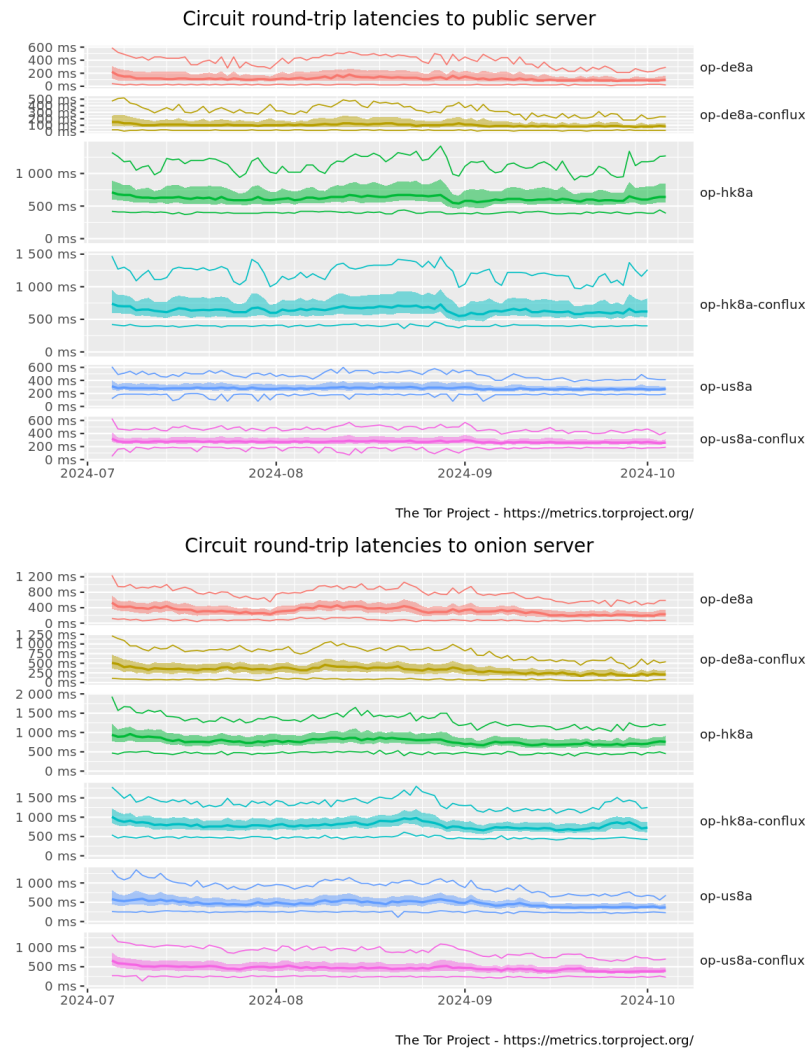
Figure A3. OnionPerf latencies between July 2024 and September 2024.

Table A1. Relay keys. Source: Tor gitlab [10].

Key	Type	Description	Expected Lifetime
relayid_rsa	1024-bit RSA	identity key	long-term, never rotated
onion_tap	1024-bit RSA	TAP onion key	medium-term, at least 1 week
conn_tls	1024-bit RSA	connection key for negotiating TLS connections	short-term, at most 1 day
ntor	x25519 (256 bits)	onion key for handling onion key handshakes	medium-term, at least 1 week
relayid_ed	ed25519 (256 bits)	identity key	long-term, never rotated
relaysign_ed	ed25519 (256 bits)	signing key	medium-term, rotated periodically
relaysign_ed_cert	ed25519 (256 bits)	relaysign_ed signed by relayid_ed	same lifetime as relaysign_ed
link_ed	ed25519 (256 bits)	link auth key, used to authenticate the link handshake	regenerated "frequently"
link_ed_cert	ed25519 (256 bits)	link_ed signed by relaysign_ed	same lifetime as link_ed

Table A2. Onion service keys. Source: Tor's gitlab [10].

Key	Type	Description	Expected Lifetime
hs_id	ed25519 (256 bits)	identity key	long-term, never rotated
hs_blind_id	ed25519 (256 bits)	blinded signing key (derived from hs_id)	1 time period
hs_desc_sign	ed25519 (256 bits)	descriptor signing key	1 time period
hs_desc_sign_cert	ed25519 (256 bits)	descriptor signing certificate (hs_desc_sign signed by hs_blind_id)	short-term (54h)
hsc_desc_enc	x25519 (256 bits)	the client's counterpart to hss_desc_enc	long-term until the client rotates it/service revokes access
hsc_intro_auth	ed25519 (256 bits)	client auth key for use in the introduction protocol	long-term/until the client rotates it/service revokes access

Table A3. Directory authority keys. Source: Tor's gitlab [10] and /var/lib/tor/cached-certs.

Key	Type	Description	Expected Lifetime
v3_ident	RSA (2048 bits)	identity key of the authority	long-term, never rotated
dir_signing	RSA (1024 bits)	signing key used for directory information	medium-term, rotated periodically
dir_signing_cert	RSA (1024 bits)	dir_signing signed by v3_ident	same lifetime as dir_signing

- **AES-128-CTR**: in counter mode as stream cipher; IV of all 0 bytes.
- **RSA**: exponent of 65,537. OAEP-MGF1 padding, with SHA-1 as its digest function (label left unset). Tor paper plans to move away from SHA-1 to align with evolving cryptographic standards and best practices.
- **SHA-1** as "hash of a public key", DER encoding of an ASN.1 RSA public key (as specified in PKCS.1).
- **Relay identity keys**: currently, two types are still deployed: RSA1024 is legacy and is being replaced by Ed25519, which is the one advertised in the table.
- **Tor's TLS 1.2 Ciphers:**
  - TLS1_TXT_DHE_RSA_WITH_AES _256_SHA
  - TLS1_TXT_DHE_RSA_WITH_AES _128_SHA
  - TLS1_TXT_ECDHE_RSA_WITH_ AES_256_GCM_SHA384
  - TLS1_TXT_ECDHE_RSA_WITH_ AES_128_GCM_SHA256
  - TLS1_TXT_ECDHE_RSA_WITH_ AES_256_CBC_SHA
  - TLS1_TXT_ECDHE_RSA_WITH_ AES_128_CBC_SHA
- **Tor's TLS 1.3 AEAD Ciphers:**
  - TLS1_3_TXT_AES_128_GCM _SHA256
  - TLS1_3_TXT_AES_256_GCM _SHA384
  - TLS1_3_TXT_CHACHA20_POLY1305 _SHA256
  - TLS1_3_TXT_AES_128_ CCM_SHA256

Figure A4. Scheme specifications for Table 1. Source: Tor's gitlab (`tortls_openssl.c`, `tortls_nss.c`, `tortls.c`). Specifications [10].

Table A4. DSA standards released by NIST in 2024. Source: FIPS 204, 205 [7]; size displayed in bytes.

Scheme	Public Key Size	Private Key Size	Signature Size
ML-DSA-44	1312	2528	4981
ML-DSA-65	1952	4000	3351
ML-DSA-87	2592	4864	4595
SLH-DSA-128f	32	64	1700
SLH-DSA-192f	48	96	3564
SLH-DSA-256f	64	128	4988

Table A5. Hash-based DSA standardised by NIST in 2020. Source: Open Quantum Safe [35]; size displayed in bytes, with four example settings selected among others.

Scheme	Public Key Size	Private Key Size	Signature Size
XMSS_SHA2_10_256	64	1373	2500
XMSS_SHAKE256_10_192	48	1053	1492
LMS_SHA256_H5_W1	60	64	8688
LMS_SHA256_H5_W8	60	64	1296

Table A6. Falcon DSA key sizes. Source: Open Quantum Safe [35]; size displayed in bytes.

Scheme	Public Key Size	Private Key Size	Signature Size
Falcon-512	897	1281	752
Falcon-1024	1793	2305	1462
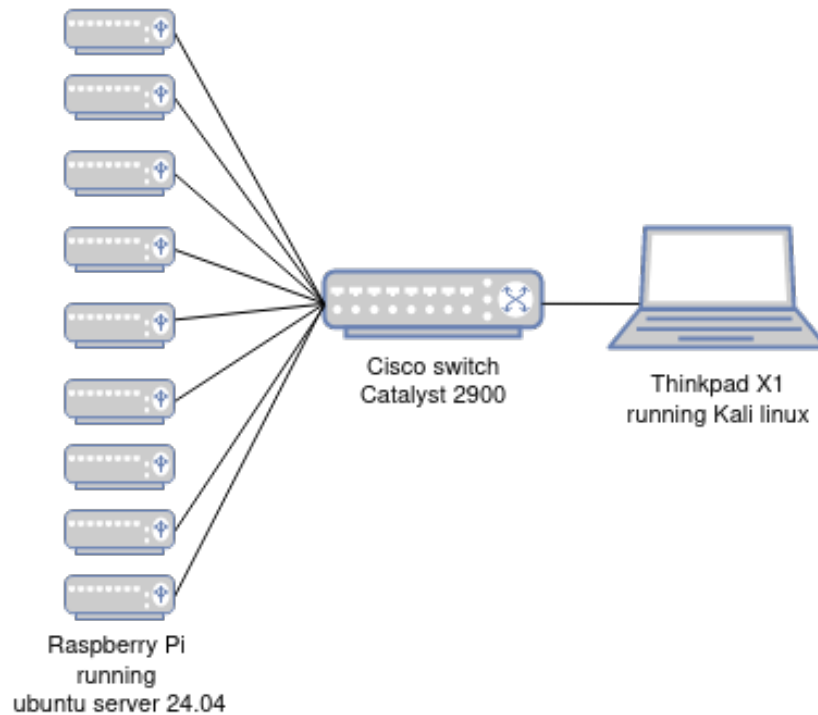
Figure A5. Hardware used for the implementation.

Listing A1. Cisco switch configuration.

```
> enable
configure terminal
ip dhcp pool LAN_POOL
network 192.168.1.0 255.255.255.0
interface vlan1
ip address 192.168.1.1 255.255.255.0
no shutdown
service dhcp
exit
write memory
```

Listing A2. Chutney configuration file.

```
Authority1 = Node(tag=''a1'', authority=1, relay=1, torrc=''authority.tmpl'')
Authority2 = Node(tag=''a2'', authority=1, relay=1, torrc=''authority.tmpl'')

GuardRelay1 = Node(tag=''g1'', relay=1, guard=1, torrc=''relay.tmpl'')
GuardRelay2 = Node(tag=''g2'', relay=1, guard=1, torrc=''relay.tmpl'')
GuardRelay3 = Node(tag=''g3'', relay=1, guard=1, torrc=''relay.tmpl'')

MiddleRelay1 = Node(tag=''m1'', relay=1, middle=1, onion_service=1, torrc=''relay.tmpl
    '')
MiddleRelay2 = Node(tag=''m2'', relay=1, middle=1, rendezvous=1, torrc=''relay.tmpl'')

ExitRelay1 = Node(tag=''e1'', relay=1, exit=1, torrc=''relay.tmpl'')
ExitRelay2 = Node(tag=''e2'', relay=1, exit=1, torrc=''relay.tmpl'')

NODES = [Authority1, Authority2, GuardRelay1, GuardRelay2, GuardRelay3, MiddleRelay1,
    MiddleRelay2, ExitRelay1, ExitRelay2]

ConfigureNodes(NODES)
```

Listing A3. Listing of files generated by node type (chutney).

```
000a1 (Directory authority)
  fingerprint
  fingerprint-Ed25519
  keys
  authority_certificate
  authority_identity_key
  authority_signing_key
  Ed25519_master_id_public_key
  Ed25519_master_id_secret_key
  Ed25519_signing_cert
  Ed25519_signing_secret_key
  secret_id_key
  secret_onion_key
  secret_onion_key_ntor
  lock
  torrc

004g3 (Guard relay)
  fingerprint
  fingerprint-Ed25519
  keys
  Ed25519_master_id_public_key
  Ed25519_master_id_secret_key
  Ed25519_signing_cert
  Ed25519_signing_secret_key
  secret_id_key
  secret_onion_key
  secret_onion_key_ntor
  lock
  torrc

005m1 (Middle relay)
  fingerprint
  fingerprint-Ed25519
  keys
  Ed25519_master_id_public_key
  Ed25519_master_id_secret_key
  Ed25519_signing_cert
  Ed25519_signing_secret_key
  secret_id_key
  secret_onion_key
  secret_onion_key_ntor
  lock
  torrc

007e1 (Exit relay)
  fingerprint
  fingerprint-Ed25519
  keys
  Ed25519_master_id_public_key
  Ed25519_master_id_secret_key
  Ed25519_signing_cert
  Ed25519_signing_secret_key
  secret_id_key
  secret_onion_key
  secret_onion_key_ntor
  lock
  torrc
```

Listing A4. Directory authority torrc file.

```
TestingTorNetwork 1

PathsNeededToBuildCircuits 0.35
TestingDirAuthVoteExit $D36D51189EC2112507DDB293FC166C5E5ADA9B91 ,
    $3E5729E0BE77AAD422BE2D95D72200EF01D1F29F
TestingDirAuthVoteExitIsStrict 1
TestingDirAuthVoteHSDir $BD452B626D1A453BB3E557B363CEC39F1E63D089 ,
    $CCE340CBBFFA07C8E434261FFCA5AB457DD06217 ,
    $D36D51189EC2112507DDB293FC166C5E5ADA9B91
#TestingDirAuthVoteHSDirIsStrict 1
TestingDirAuthVoteGuard $84C8BDC0ECB2A6F7B536A2C6696DE0ADE8AC09A9 ,
    $139726CAC0132D62C1D6E5E2F306315CCFE857A9 ,
    $A0595788A046B7B2DF48B04A02C262DC525A82E2 ,
    $691B61F16F6D48F65183AF7A39BC789645C14C49 ,
    $2EEB5B771A2A177EC0D184D410DDB4F153FF354F
TestingDirAuthVoteGuardIsStrict 1
V3AuthNIntervalsValid 2
V3BandwidthsFile /home/ubuntu/000a1/bwfile.v3bw

MiddleNodes $691B61F16F6D48F65183AF7A39BC789645C14C49 ,
    $CCE340CBBFFA07C8E434261FFCA5AB457DD06217
DataDirectory /home/ubuntu/000a1
RunAsDaemon 1
ConnLimit 60
Nickname test000a1
ShutdownWaitLength 2
DisableDebuggerAttachment 0

AddressDisableIPv6 1
ControlPort 8000
ControlSocket /home/ubuntu/000a1/control
CookieAuthentication 1
PidFile /home/ubuntu/000a1/pid

Log notice file /home/ubuntu/000a1/notice.log
Log info file /home/ubuntu/000a1/info.log
Log debug file debug.log
ProtocolWarnings 1
SafeLogging 0
LogTimeGranularity 1

AuthoritativeDirectory 1
V3AuthoritativeDirectory 1
ContactInfo auth0@test.test
DirAuthority test000a1 orport=5100 no-v2 v3ident=4832
    D1130A4F32E4410E42D33C36B6B87EC37C27 192.168.1.15:7100
    EA30E583EF155930ACE4144D0CFA800725C65D46
DirAuthority test001a2 orport=5101 no-v2 v3ident=43
    DE83E17AB616F71A36394D519732767B2CEF50 192.168.1.16:7101
    BD452B626D1A453BB3E557B363CEC39F1E63D089
SocksPort 9050
OrPort 5100
Address 192.168.1.15
DirPort 7100
ExitRelay 0
DisableNetwork 0
ConfluxEnabled 0
TestingAuthKeyLifetime 3 months
ServerDNSAllowNonRFC953Hostnames 1
ServerDNSDetectHijacking 0
```

Listing A5. Guard relay torrc file.

```
TestingTorNetwork 1
PathsNeededToBuildCircuits 0.45
TestingDirAuthVoteExit *
TestingDirAuthVoteHSDir *
V3AuthNIntervalsValid 2

TestingDirAuthVoteGuard *
TestingMinExitFlagThreshold 0

DataDirectory /home/ubuntu/002g1
RunAsDaemon 1
ConnLimit 60
Nickname test002g1
ShutdownWaitLength 2
DisableDebuggerAttachment 0
AddressDisableIPv6 1
ControlPort 8002
ControlSocket /home/ubuntu/002g1/control
CookieAuthentication 1
PidFile /home/ubuntu/002g1/pid
Log notice file /home/ubuntu/002g1/notice.log
Log info file /home/ubuntu/002g1/info.log
ProtocolWarnings 1
SafeLogging 0
LogTimeGranularity 1

DirAuthority test000a1 orport=5100 no-v2 v3ident=4832
    D1130A4F32E4410E42D33C36B6B87EC37C27 192.168.1.15:7100
    EA30E583EF155930ACE4144D0CFA800725C65D46
DirAuthority test001a2 orport=5101 no-v2 v3ident=43
    DE83E17AB616F71A36394D519732767B2CEF50 192.168.1.16:7101
    BD452B626D1A453BB3E557B363CEC39F1E63D089

OrPort 5102
Address 192.168.1.17
ExitRelay 0
DisableNetwork 0
ConfluxEnabled 0
```

Listing A6. Client torrc file.

```
TestingTorNetwork 1

DirAuthority test000a1 orport=5100 no-v2 v3ident=4832
    D1130A4F32E4410E42D33C36B6B87EC37C27 192.168.1.15:7100
    EA30E583EF155930ACE4144D0CFA800725C65D46
DirAuthority test001a2 orport=5101 no-v2 v3ident=43
    DE83E17AB616F71A36394D519732767B2CEF50 192.168.1.16:7101
    BD452B626D1A453BB3E557B363CEC39F1E63D089
SocksPort 9050
DisableNetwork 0
CircuitBuildTimeout 60
LearnCircuitBuildTimeout 0
DataDirectory /var/lib/tor
LogTimeGranularity 1

NumEntryGuards 1
UseEntryGuards 0
ClientOnly 1

SocksPolicy accept 192.168.0.0/16
RunAsDaemon 1
ConfluxEnabled 0
```
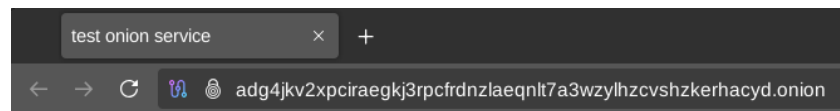
Listing A7. Tor browser command.

```
$ TOR_SKIP_LAUNCH=1 TOR_SOCKS_PORT=9050 TOR_SKIP_CONTROLPORTTEST=1 /home/$USERNAME/.
    local/share/torbrowser/tbb/x86_64/tor-browser/Browser/start-tor-browser
```

Figure A6. Onion service test.

Listing A8. Tor Browser command.

```
for i in {1..500}; do
    curl --socks5-hostname 127.0.0.1:9050 -w "%{time_total}, " -o /dev/null -X POST \
-H ''Content-Type: application/json'' \
-H ''User-Agent: BenchmarkingAgent/1.0'' \
-H "Accept: application/json" \
-d '{
  "cookie": "value1111111111111111111111111111",
  "username": "testuser",
  "password": "7647873645672837463782376463782",
  "session_id": "abc123xyz4567890",
  "data": {
    "field1": "Some random text to increase payload size 1",
    "field2": "Some random text to increase payload size 2",
    "field3": "Some random text to increase payload size 3"
  }
}' \
-s https://local.tor/postpath
done
```
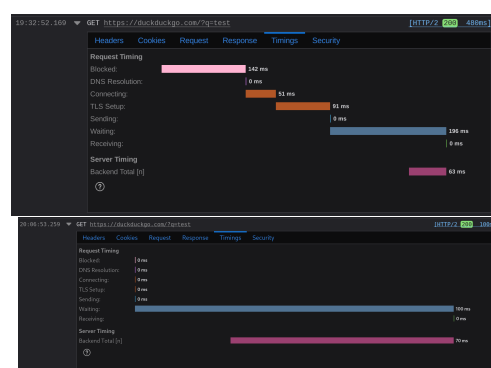

Figure A7. Comparison of request timing between the local Tor network (upper image) and direct internet access (lower image). The upper screenshot is from the Tor browser, while the lower one is from Firefox.
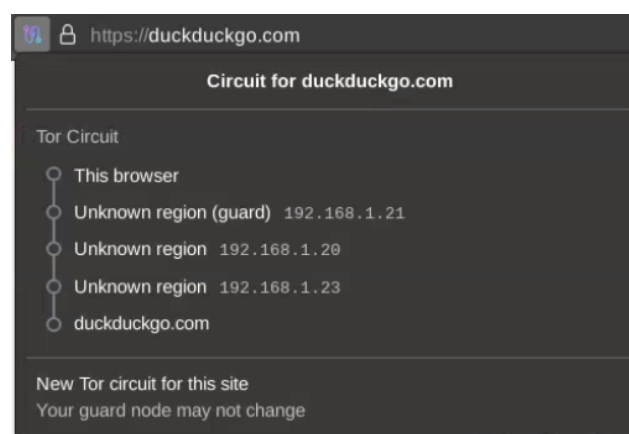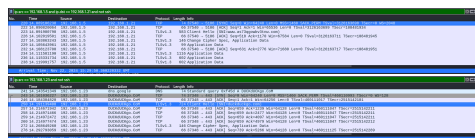

Figure A8. Tor browser displaying circuit.

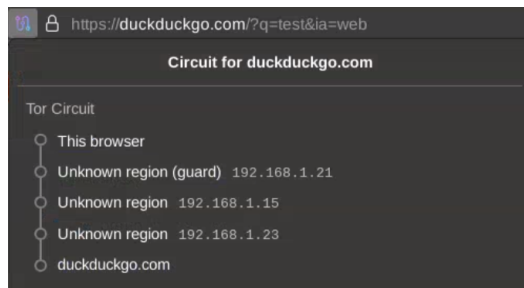**Figure A9.** Wireshark sniffing eth0 interface.

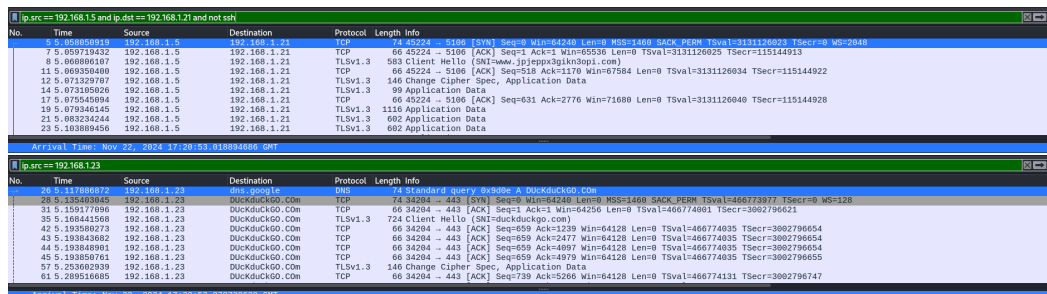**Figure A10.** Tor browser displaying circuit.

Figure A11. Wireshark sniffing eth0 interface.

Figure A12. Nyx overview.

References

1. Shor, P. Algorithms for quantum computation: Discrete logarithms and factoring. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, USA, 20–22 November 1994; pp. 124–134. [CrossRef]
2. Grover, L.K. A fast quantum mechanical algorithm for database search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, PA, USA, 22–24 May 1996.
3. TOR Metrics. Available online: https://metrics.torproject.org/ (accessed on 20 September 2024).
4. Dingledine, R.; Syverson, P. Tor: The Second-Generation Onion Router. In Proceedings of the USENIX Security Symposium, San Diego, CA, USA, 9–13 August 2004.

5. Tor Proposals. Available online: https://spec.torproject.org/proposals/ (accessed on 18 September 2024).

6. Gidney, C.; Ekerå, M. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum* **2021**, *5*, 433. [CrossRef]

7. Raimondo, G.M.; Locascio, L.E. FIPS 204 Federal Information Processing Standards Publication Module-Lattice-Based Digital Signature Standard. *Nist Natl. Inst. Stand. Technol.* **2024**, 55. [CrossRef]

8. Moody, D.; Perlner, R.; Regenscheid, A.; Robinson, A.; Cooper, D. *Transition to Post-Quantum Cryptography Standards*; NIST Internal Report 8547; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2024. [CrossRef]

9. Goldberg, I.; Cheriton, D.R. On the Security of the Tor Authentication Protocol. In *International Workshop On Privacy Enhancing Technologies*; Springer: Berlin/Heidelberg, Germany, 2006.

10. Tor Specifications. Available online: https://spec.torproject.org/tor-spec/index.html (accessed on 21 September 2024).

11. Schanck, J.M.; Whyte, W.; Zhang, Z. Circuit-extension handshakes for Tor achieving forward secrecy in a quantum world. *Cryptol. Eprint Arch.* **2015**, *2016*, 219–236. [CrossRef]

12. Goldberg, I.; Stebila, D.; Ustaoglu, B. Anonymity and one-way authentication in key exchange protocols. *Des. Codes Cryptogr.* **2013**, *67*, 245–269. [CrossRef]

13. Ghosh, S.; Kate, A. Post-Quantum Forward-Secure Onion Routing (Future Anonymity in Today's Budget). *Cryptol. Eprint Arch.* **2015**, *9092*, 263–286.

14. Lauer, S.; Gellert, K.; Merget, R.; Handirk, T.; Schwenk, J. T0RTT: Non-Interactive Immediate Forward-Secret Single-Pass Circuit Construction. *Proc. Priv. Enhancing Technol.* **2020**, *2020*, 336–357. [CrossRef]

15. Degabriele, J.P.; Stam, M. Untagging tor: A formal treatment of onion encryption. In *Proceedings of the Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; LNCS; Springer: Berlin/Heidelberg, Germany, 2018; Volume 10822, pp. 259–293. [CrossRef]

16. Rogaway, P.; Zhang, Y. Onion-AE: Foundations of Nested Encryption. *Proc. Priv. Enhancing Technol.* **2018**, 85–104. [CrossRef]

17. Jaques, S.; Naehrig, M.; Roetteler, M.; Virdia, F. Implementing Grover Oracles for Quantum Key Search on AES and LowMC. In Proceedings of the EUROCRYPT 2020, Virtual Conference, 11–15 May 2020.

18. Dowling, B.; Fischlin, M.; Günther, F.; Stebila, D. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol. *J. Cryptol.* **2021**, *34*, 37. [CrossRef]

19. Schwabe, P.; Stebila, D.; Wiggers, T. Post-Quantum TLS Without Handshake Signatures. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, 9 November 2020; Technical Report 2020/534, IACR Cryptology ePrint Archive.

20. Perrin, T. The Noise Protocol Framework, 2018. Revision 34, Status: Official. Available online: https://noiseprotocol.org/ (accessed on 8 October 2024)

21. Schwabe, P.; Stebila, D.; Wiggers, T. More efficient post-quantum KEMTLS with pre-distributed public keys. In *Computer Security—ESORICS 2021, Proceedings of the European Symposium on Research in Computer Security, Darmstadt, Germany, 4–8 October 2021*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 3–22, Cryptology ePrint Archive, Paper 2021/779. [CrossRef]

22. Stebila, D.; Fluhrer, S.; Gueron, S. Hybrid Key Exchange in TLS 1.3. Internet-Draft Draft-Ietf-Tls-Hybrid-Design-11, Internet Engineering Task Force. 2024. Available online: https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/ (accessed on 11 November 2024).

23. Langley, A. CECPQ2: Post-Quantum Confidentiality in TLS. 2018. Available online: https://www.imperialviolet.org/2018/12/12/cecpq2.html (accessed on 14 September 2024).

24. Baseri, Y.; Chouhan, V.; Ghorbani, A.; Chow, A. Evaluation Framework for Quantum Security Risk Assessment: A Comprehensive Study for Quantum-Safe Migration. *Comput. Secur.* **2025**, *150*, 104272. [CrossRef]

25. Barker, E.; Dang, Q.; Hanon, N. *NIST Special Publication 800-57 Part 1, Revision 5: Recommendation for Key Management, Part 1: General*; Technical Report 800-57pt1r5; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2020. [CrossRef]

26. Barton, J.; Buchanan, W.; Pitropakis, N.; Sayeed, S.; Abramson, W. Post Quantum Cryptography Analysis of TLS Tunneling on a Constrained Device. In Proceedings of the 8th International Conference on Information Systems Security and Privacy—ICISSP, Online, 9–11 February 2022; Volume 1, pp. 551–561. [CrossRef]

27. Alagic, G.; Apon, D.; Cooper, D.; Dang, Q.; Dang, T.; Kelsey, J.; Lichtinger, J.; Liu, Y.K.; Miller, C.; Moody, D.; et al. *Status Report on the Third Round of NIST Post-Quantum Cryptography Standardization Process*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2022. [CrossRef]

28. Kret, E.; Schmidt, R. The PQXDH Key Agreement Protocol. Revision 3. 2023. Available online: https://signal.org/docs/specifications/pqxdh/pqxdh.pdf (accessed on 23 January 2024).

29. Cheng, H.; Großschädl, J.; Rønne, P.B.; Ryan, P.Y.A. AVRNTRU: Lightweight NTRU-based Post-Quantum Cryptography for 8-bit AVR Microcontrollers. In Proceedings of the DATE 2021: Design, Automation and Test in Europe, Virtual Conference, 1–5 February 2021.

30. Bernstein, D.J.; Chuengsatiansup, C.; Lange, T.; van Vredendaal, C. NTRU Prime: Reducing attack surface at low cost. In Proceedings of the Selected Areas in Cryptography—SAC 2017: 24th International Conference, Ottawa, ON, Canada, 16–18 August 2017.

31. Bos, J.W.; Bronchain, O.; Custers, F.; Renes, J.; Verbakel, D.; van Vredendaal, C. Enabling FrodoKEM on Embedded Devices. *Cryptol. Eprint Arch.* **2023**, Paper 2023/158. Available online: https://eprint.iacr.org/2023/158 (accessed on 1 January 2025).

32. Bos, J.W.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.M.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS—Kyber: A CCA-Secure Module-Lattice-Based KEM. In Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroS&P), London, UK, 24–26 April 2018; pp. 353–367. [CrossRef]

33. Raimondo, G.M.; Locascio, L.E. FIPS 203 Federal Information Processing Standards Publication Module-Lattice-Based Key-Encapsulation Mechanism Standard. *Nist Natl. Inst. Stand. Technol.* **2024**, 47. [CrossRef]

34. Kwala, A.K.; Kant, S.; Mishra, A. Comparative analysis of lattice-based cryptographic schemes for secure IoT communications. *Discov. Internet Things* **2024**, *4*, 13. [CrossRef]

35. Open Quantum Safe, Liboqs. Available online: https://openquantumsafe.org (accessed on 8 October 2024).

36. Melchor, C.A.; Aragon, N.; Bettaieb, S.; Bidoux, L.; Blazy, O.; Bos, J.; Deneuville, J.C.; Arnaud Dion, P.G.; Lacan, J.; Persichetti, E.; et al. Hamming Quasi-Cyclic (HQC) Fourth Round Specification. Technical report, SandboxAQ, Univ. of Limoges, TII, Ecole Polytechnique, Worldline, ENAC, ISAE Supaero, Florida Atlantic Univ., Univ. of Toulon, Univ. of Bordeaux. 2024. Available online: https://pqc-hqc.org/ (accessed on 4 November 2024).

37. Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.3. Request for Comments 8446, 2018. Standards Track. Available online: https://datatracker.ietf.org/doc/rfc8446/ (accessed on 4 November 2024).

38. Færøy, A.H. BoringSSL Tor Fork. Branch: Ahf/boringssl. Available online: https://gitlab.torpaper.org/ahf/tor/-/commits/ahf%2Fboringssl (accessed on 11 November 2024).

39. Paquin, C.; Stebila, D.; Tamvada, G. Benchmarking Post-quantum Cryptography in TLS. In *Post-Quantum Cryptography*; Ding, J., Tillich, J.P., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 72–91.

40. Alnahawi, N.; Müller, J.; Oupický, J.; Wiesmaier, A. A Comprehensive Survey on Post-Quantum TLS. *Iacr Commun. Cryptol.* **2024**, *1*, hal-04845617. [CrossRef]

41. Pan, J.; Riepel, D.; Zeng, R. Key Exchange with Tight (Full) Forward Secrecy via Key Confirmation. *Cryptol. Eprint Arch.* **2024**, *14657*, 59–89.

42. Pan, J.; Wagner, B.; Zeng, R. Tighter Security for Generic Authenticated Key Exchange in the QROM. *Cryptol. Eprint Arch.* **2023**, *14441*, 401–433.

43. Eaton, E.; Stebila, D.; Stracovsky, R. Post-Quantum Key-Blinding for Authentication in Anonymity Networks. A Compressed Version Appears in the Proceedings of Latincrypt 2021. Available online: https://eprint.iacr.org/2021/963.pdf (accessed on 16 November 2024).

44. Kampanakis, P.; Childs-Klein, W. The impact of data-heavy, post-quantum TLS 1.3 on the Time-To-Last-Byte of real-world connections. *Cryptol. Eprint Arch.* **2024**, Paper 2024/176. [CrossRef]

45. Group, L.W. Stateless Hash-Based Digital Signature Algorithms (SLH-DSA) in X.509 Public Key Infrastructure (PKI). Internet-Draft Draft-Ietf-Lamps-x509-slhdsa-02, Internet Engineering Task Force. 2024. Available online: https://datatracker.ietf.org/doc/draft-ietf-lamps-x509-slhdsa/ (accessed on 13 November 2024).

46. Sikeridis, D.; Kampanakis, P.; Devetsikiotis, M. Post-Quantum Authentication in TLS 1.3: A Performance Study. *Cryptol. Eprint Arch.* **2020**, Paper 2020/071. [CrossRef]

47. Kampanakis, P.; Kallitsis, M. Faster Post-Quantum TLS Handshakes Without Intermediate CA Certificates. In *Cyber Security, Cryptology, and Machine Learning*; Dolev, S., Katz, J., Meisels, A., Eds.; Springer International Publishing: Cham, Switzerland, 2022; pp. 337–355.

48. Misell, Q. Automated Certificate Management Environment (ACME) Extensions for ".onion" Special-Use Domain Names. Internet-Draft Draft-Ietf-Acme-Onion-04, Internet Engineering Task Force (IETF). Available online: https://acmeforonions.org/ (accessed on 12 November 2024).

49. Preston, R. Applying Grover's Algorithm to Hash Functions: A Software Perspective. *IEEE Trans. Quantum Eng.* **2023**, *3*, 1–10. [CrossRef]

50. Alagic, G.; Bros, M.; Ciadoux, P.; Cooper, D.; Dang, Q.; Dang, T.; Kelsey, J.; Lichtinger, J.; Liu, Y.K.; Miller, C.; et al. *Status Report on the First Round of the Additional Digital Signature Schemes for NIST Post-Quantum Cryptography Standardization Process*; NIST Internal Report 8528; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2024.

51. Baum, C.; Braun, L.; de Saint Guilhem, C.D.; Klooß, M.; Orsini, E.; Roy, L.; Scholl, P. Publicly Verifiable Zero-Knowledge and Post-Quantum Signatures From VOLE-in-the-Head. *Cryptol. Eprint Arch.* **2023**, *14085*, 581–615.

52. Castryck, W.; Decru, T. An efficient key recovery attack on SIDH. *Cryptol. Eprint Arch.* **2022**, *14008*, 423–447.

53. Rahman, M.S.; DiAdamo, S.; Mehic, M.; Fleming, C. Quantum Secure Anonymous Communication Networks. In Proceedings of the 2024 International Conference on Quantum Communications, Networking, and Computing (QCNC), Kanazawa, Japan, 1–3 July 2024.

54. Zhang, Y.; Zhang, Y.; Portokalidis, G.; Xu, J. Towards Understanding the Runtime Performance of Rust. In Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), New York, NY, USA, 10–14 October 2022; pp. 1–6. [CrossRef]

55. DAST; Team, E.D. iPerf: The Network Bandwidth Measurement Tool. NLANR and ESnet. 2003. Available online: https://github.com/esnet/iperf (accessed on 19 November 2024).

56. QSOR: Quantum-Safe Onion Routing. Available online: https://arxiv.org/pdf/2001.03418 (accessed on 21 November 2024).

57. Evgnosia-Alexandra, K. A note on Post Quantum Onion Routing. *Cryptol. Eprint Arch.* **2021**, Paper 2021/111. Available online: https://eprint.iacr.org/2021/111 (accessed on 1 January 2025).

58. PlanetLab Europe. Available online: https://www.planet-lab.eu/ (accessed on 13 November 2024).