An Evaluation of Post-Quantum and Hybrid Noise Protocol Variants on Mobile Devices

Joshua Renckens, Peter B. Rønne, Johann Großschädl, and Peter Y. A. Ryan

SnT and DCS, University of Luxembourg
6, Avenue de la Fonte, L–4364 Esch-sur-Alzette, Luxembourg
joshua.renckens0@gmail.com
{peter.roenne,johann.groszschaedl,peter.ryan}@uni.lu

Abstract. Noise is a framework for the design and security assessment of Authenticated Key Exchange (AKE) protocols between two parties using Diffie-Hellman (DH) as the only public-key cryptosystem. In this paper, we present an evaluation of the computation and communication cost of Noise and PQNoise, a recently introduced post-quantum version of the Noise protocol framework. Furthermore, we present combinations of the 12 fundamental (interactive) Noise patterns and their PQNoise counterparts, thereby obtaining hybrid handshake patterns, and include them in our evaluation. We integrated PQNoise and the novel hybrid patterns into Noise-C, a reference implementation of the Noise protocol framework written in C. In order to evaluate Noise and its variants, we emulated networks with different latency, throughput, and packet-loss settings using Linux network emulation tools. For all Noise handshakes we chose cryptosystems that provide a comparable (pre-quantum) level of security, namely X25519 and Kyber512. We ran our experiments on two different devices, one is a laptop with an Intel Core i5-10210U CPU and the other an Orange Pi One development board with a 32-bit ARM Cortex-A7 processor. The results we collected show that, under normal network conditions, the Noise patterns and their PQNoise counterparts have nearly identical execution times, except when the latter require an additional handshake message. However, under bad network conditions with high packet-loss rates, PQNoise falls behind Noise, mainly because of the relatively large public-key and ciphertext sizes of Kyber512. The execution times of our hybrid handshakes are almost indistinguishable from the corresponding PQNoise handshakes when the packet-loss rates are low, and at higher loss rates the differences are small.

Keywords: End-to-end security · Post-quantum cryptography · Noise protocol · Key encapsulation mechanism · Hybrid key establishment

1 Introduction

The *Transport Layer Security (TLS)* protocol is one of the most common End-to-End (E2E) security protocols on the Internet and secures billions of HTTPS transactions daily. It consists of a few sub-protocols, of which the handshake

protocol and the record protocol are the most important ones. The former is responsible for authentication and key exchange using public-key cryptographic schemes, while the latter ensures the confidentiality and integrity of application data through symmetric cryptosystems [20,24]. Initially designed in the 1990s under the name *Secure Sockets Layer (SSL)*, the TLS protocol has since then undergone a number of revisions to extend its functionality and fix weaknesses and flaws [16]. These over 30 years of evolution have made TLS a very complex protocol, which is overloaded with many legacy versions that still need to be supported for backward compatibility and many niche features and extensions with disputable relevance for real-world use cases. The high complexity of TLS has been the root cause for various types of attacks on the protocol itself and on different implementations [22]. In addition, complex protocols like TLS are unattractive for embedded and mobile devices, which are usually constrained in computational power and energy supply compared to conventional PCs.

Noise [19] was introduced by Trevor Perrin in 2014 as a framework for the design of custom E2E security protocols whose AKE component is solely based on Diffie-Hellman (DH) [6] operations. Noise-based protocols aim to reduce the complexity caused by run-time decisions during protocol execution, which are prevalent in TLS and "interwoven" with the handshake, by relegating them to design-time decisions (e.g., one-way or mutual authentication) within a framework and a thin negotiation layer (i.e., an optional prologue) before the actual handshake to agree on a handshake pattern and related cryptosystems. Also, in contrast to TLS, Noise adopts long-term (static) DH keys instead of long-term signature keys to authenticate the ephemeral DH key exchange, thereby minimizing the number of public-key schemes that need to be supported[1]. Once the negotiation phase is completed, the rest of the handshake is a straight sequence of message transfers and DH operations without any conditional statements like branches, which massively simplifies the implementation testing of the protocol and also reduces the attack surface compared to TLS. The concrete handshake pattern determines many of the security properties the resulting protocol will have; for example, if the two parties do not need to be authenticated, then the so-called `NN` pattern will suffice, but if server authentication (with transmission of the server's static key, similar to TLS) is desired, then the `NX` pattern is the best choice [19]. The existence of several handshake patterns with well-studied security properties allows designers to simply choose the Noise variant that fits the security needs of their target application best, and saves them the time and effort of designing a custom protocol from scratch or trying to figure out how to disable and/or remove unnecessary TLS functionalities.

Early versions of TLS (resp., SSL) used "classical" public-key cryptosystems like RSA, DSA, and DH, but over the years they have become largely replaced by elliptic curve schemes (i.e., ECDH for key exchange and ECDSA for authentication). As mentioned above, Noise uses ECDH (based on either Curve25519 or Curve448 [14]) not only for the establishment of shared secrets, but also the authentication of the ephemeral public keys exchanged during a handshake. In

[1] Note that all DH operations/keys of Noise are, in fact, ECDH operations/keys.

1994, Peter Shor introduced quantum algorithms for efficient (polynomial-time) factorization of large composite numbers and efficient computing of the discrete logarithm problem, thereby enabling quantum computers to break virtually all of the currently prevalent public-key schemes, including DH, RSA, ECDH, and ECDSA [23]. Due to the looming threat of quantum cryptanalysis, it becomes necessary to replace these algorithms by post-quantum alternatives that fulfill the same purpose, but are able to withstand cryptanalytic attacks executed on large-scale quantum computers. A well-known example of such a post-quantum algorithm is *Kyber* [5], a lattice-based KEM selected for standardization by the U.S. National Institute of Standards and Technology (NIST). This transition to *Post-Quantum Cryptography (PQC)* also requires an adaptation and extension of security protocols like TLS and Noise to support the new algorithms. There have been a couple of initiatives for adding post-quantum schemes to TLS; two examples are the crypto provider of the Open Quantum Safe project [25] and KEMTLS [21]. On the other hand, for Noise, the only post-quantum variant we are aware of is *PQNoise* [3], which was published in 2022.

Besides the concrete integration of post-quantum schemes into TLS, a large body of research has been devoted to analyzing how they affect the execution time and the amount of data exchanged in a handshake [15,12,2]. For Noise, on the other hand, the research so far has focussed on proving security properties of different Noise protocols [13,8,3] and automated code generation for various Noise variants [11]. Since PQNoise is still fairly new, relatively little is known about its performance apart from results of a Go implementation for two handshake patterns described in [3]. However, according to [17, Table 4], software developed in the Go language seems to be significantly less efficient, in terms of execution time and energy consumption, than software written in C. A more detailed comparison of pre- and post-quantum Noise, especially a comparison based on efficient C implementations of the handshakes, is still lacking.

In this paper, we study the performance of an optimized C implementation of PQNoise, taking into account the computation and communication cost, respectively, and compare it to Noise as well as TLS. Such kind of performance evaluation is important to assess how the migration to PQC might impact the latency of secure communication of widely-used Internet applications. We are especially interested in measuring and comparing the performance of pre- and post-quantum security protocols on mobile devices that are less powerful than conventional PCs. This means we evaluated these protocols on mobile devices under network conditions (i.e., latency, throughput, packet-loss rate) similar to that of mobile (cellular) networks based on 4G technology. Although Noise has been integrated into a number of applications, e.g., WhatsApp [27] and WireGuard [7], it will most likely not replace TLS on a large scale. However, there is a good chance that Noise will find widespread adoption for use cases where one entity controls all the communication endpoints, e.g., the client app running on smart phones and the back-end servers. In such settings, it often makes sense to use a custom protocol (designed and security-evaluated with help of the Noise framework) that is specifically tailored for the target application.

Apart from an evaluation of the performance of PQNoise, we also describe hybrid variants of Noise handshakes combining X25519 and Kyber512. There has been prior work on creating hybrid versions of Noise [18], which focused on defining Noise patterns that offer *Hybrid Forward Secrecy (HFS)* through the introduction of a `"hfs"` pattern-modifier and two new tokens. The new tokens represent KEM operations and the transfer of KEM public keys or ciphertexts to the other party. However, the specification of these HFS patterns for Noise was last updated in 2018 (i.e., four years before the publication of the PQNoise paper) and its current status is still unofficial. *Noise-C* [26], which is intended as a reference implementation of Noise, contains a version of the HFS patterns based on NewHope [1] instead of Kyber. In our work, we combine the classical X25519-based Noise patterns with their PQNoise counterparts to obtain novel hybrid Noise handshakes that use Kyber512 as KEM to achieve post-quantum security. We integrated these hybrid patterns to Noise-C and included them in our evaluation of the computation and communication cost of handshakes on mobile devices. Furthermore, we analyze how the combination of classical and post-quantum algorithms, as well as the additional handshake data that have to be transmitted, slows down the hybrid patterns compared to PQNoise.

2 Preliminaries

2.1 Noise Protocol Framework

As mentioned in the previous section, Noise is framework to support the design and security evaluation of custom protocols between two parties, usually called initiator and responder. A concrete instance of a Noise-based security protocol is defined by a protocol name, which includes the handshake pattern, the DH function (either X25519 or X448), an algorithm for Authenticated Encryption with Associated Data (AEAD), and a hash function [19]. A concrete example for a Noise protocol name is `Noise_NX_25519_AESGCM_SHA256`, where `NX` is the handshake pattern. Handshake pattern names usually consist of two uppercase letters, indicating the status of the static DH keys and how they are conveyed to the other party (as will be further discussed below), followed by one or more optional pattern modifiers. The concrete handshake pattern to be used by the protocol, including the underlying cryptosystems, is normally chosen at design time. However, as mentioned in [19, Sect. 6], it is also possible that the parties agree on it interactively; for example, the initiator can send a list of supported Noise protocols to the responder in a so-called preamble. Once this negotiation is completed, the actual handshake is a linear sequence of message (i.e., public-key) transfers and cryptographic operations that does not involve any run-time decisions apart from error handling [19]. Depending on the chosen handshake pattern, a Noise protocol can have different security properties, e.g., unilateral or mutual authentication, weak or strong forward secrecy, identity hiding, and zero round-trip encryption. All patterns have in common that they exchange public DH keys (static or ephemeral), perform DH operations, and hash all the computed DH results together to derive a shared secret.

Pattern Components. A handshake pattern describes the overall execution of the handshake and consists of a sequence of *message patterns*, which defines the exchange of messages. The message patterns, in turn, are composed of one or more *tokens* that specify the involved DH public keys and DH operations to be executed when sending or receiving the message. Besides such patterns and tokens, the Noise specification also defines a couple of variables, which have to be maintained by each party, irrespective of the used handshake pattern.

- A *handshake hash* value `h` representing a fingerprint of the pre-distributed static DH keys and all exchanged messages. It is initialized by hashing the Noise protocol name. Whenever static public DH keys and/or payloads are AEAD-encrypted, `h` is included in the message as associated data.
- A *chaining key* `ck`, which is also initialized with the hash of the protocol name. The result of every DH operation is "mixed" into `ck` through a hash-based key-derivation function. After completion of the handshake, `ck` serves to generate encryption keys for transport messages.
- An *encryption key* `k` for encryption/decryption of public static DH keys as well as handshake payloads using an AEAD algorithm. This key is derived from the chaining key `ck` and gets updated every time `ck` changes.
- A counter-based *nonce* `n` of 64 bits in length, which is used as input for the AEAD algorithm. The nonce gets reset to zero every time `ck` changes.

There are six different standard tokens that can appear in a Noise message pattern; two represent public keys and the other four DH operations performed on a combination of two DH key-shares, one private and the other public.

- `"e"`: The sender generates a new (ephemeral) DH key pair and then sends the public key to the receiver unencrypted. This public key will be used to update the handshake hash `h` on both sides.
- `"s"`: The sender sends its static public DH key to the receiver. This key is normally encrypted using the encryption key `k` and nonce `n`. However, the DH key is sent in clear when `k` is still empty, which is the case when no DH operation has been carried out yet. Further, the message sent, irrespective of whether it is encrypted or not, is hashed into `h` by both parties.
- `"ee"`, `"es"`, `"se"`, and `"ss"`: These tokens instruct the parties to perform DH computations that involve a key-share of the initiator and a key-share of the responder. The left letter stands for the initiator's key-share and the right letter for the responder's key-share to be used. Each key-share can be ephemeral (`e`) or static (`s`). The result of the DH operation is mixed into the chaining key `ck`, at which point a new `k` is generated and `n` is set to zero.

The tokens can be arranged into a message pattern and several of such message patterns constitute a handshake pattern. How a handshake pattern is executed in detail is best explained using a single example like the `KN` pattern, illustrated in Fig. 1. This handshake pattern contains the delimiter `"..."` (i.e., an ellipsis or three periods), which separates the so-called *pre-message patterns* from the actual message patterns. A pre-message can be used to pre-distribute a static

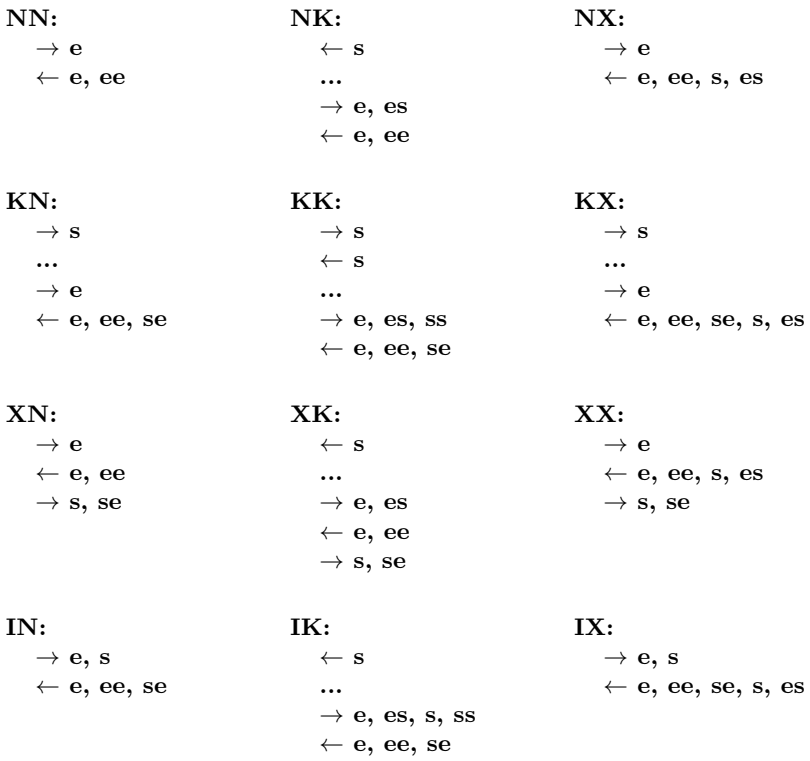


Fig. 1. The 12 fundamental interactive Noise patterns

public DH key, e.g., by means of an out-of-band transmission, before the actual (interactive) part of the handshake starts. In the `KN` pattern, the pre-message consists of a right arrow and an `"s"` token, indicating that the initiator's static public key was already shared with the responder at some point prior to the actual handshake. A right arrow represents a message from the initiator to the responder, and a left arrow a message in the other direction. The first message pattern, a right arrow with an `"e"` token, means that the initiator sends the public share of a freshly-generated ephemeral DH key-pair to the responder. On the other hand, the second message pattern is composed of a left arrow and the tokens `"e"`, `"ee"`, and `"se"`. This means the responder generates an ephemeral DH key-pair and sends its public share to the initiator (`"e"`). Then, both sides carry out an ephemeral-ephemeral (`"ee"`) DH, followed by a static-ephemeral (`"se"`) DH using the initiator's static public key, which was pre-distributed.

Fundamental Interactive Patterns. The tokes and patterns outlined above provide a flexible toolbox for the design of custom handshake protocols. Alternatively, instead of creating a new handshake from scratch, a designer can also use one of the 12 so-called fundamental interactive patterns that are described

in the Noise specification. The security properties of the fundamental patterns are well studied and were even formally proven in [13,8]. As already mentioned earlier, the first letter of a pattern name conveys information on the initiator's static public key, and the second letter on the responder's static public key. In total, there are four possible letters on the initiator's side:

- `N` ("None"): The initiator does not use a static DH key in this pattern and is, therefore, not authenticated during the handshake.
- `K` ("Known"): The initiator's static DH key was pre-distributed, i.e., sent to the responder, before the handshake execution. Hence, the responder knows (and implicitly trusts) the initiator's static key.
- `X` ("Xmitted"): The static DH key of the initiator is sent to the responder in one of the handshake messages, usually encrypted (which can help to hide the initiator's identity).
- `I`: ("Immediate"): The initiator sends its static DH key immediately to the responder (i.e., in the first handshake message), despite the fact that this early transmission reduces or inhibits identity hiding.

The responder's static DH key can have the `N`, `K`, or `X` property, but not `I`. As a consequence, there are 12 possible combinations of initiator/responder static DH keys; the resulting handshake patterns are illustrated in Fig. 1.

2.2 Kyber KEM

CRYSTALS-Kyber (Kyber for short) is a family of post-quantum KEMs whose security is based on the hardness of the Module Learning with Errors (MLWE) problem, which comes with a worst-case to average-case reduction from certain hard problems over module lattices [5]. The NIST is currently working towards standardizing Kyber (under the name "ML-KEM") with three parameter sets providing security levels roughly similar to that of the AES with 128, 192, and 256-bit keys, respectively. The algebraic structure in which Kyber operates is a module[2] of low rank k defined over a polynomial ring $\mathcal{R}_q$. Depending on the security level, k is either 2, 3, or 4. All three instances of Kyber use the same ring, namely $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n - 1)$ with $n = 2^8$ and $q = 13n + 1 = 3329$. The base field $\mathbb{Z}_q$ contains primitive 256-th roots of unity so that multiplication in $\mathcal{R}_q$ can be implemented very efficiently using the Number-Theoretic Transform (NTT). Besides polynomial arithmetic, other building blocks of Kyber include functions for the sampling of polynomials (i.e., coefficients) from uniform and centered binomial distributions, and SHA3-based functions for pseudo-random number generation from a seed, hashing, and shared-secret derivation.

From a high-level perspective, Kyber (as any other KEM) consists of three functions: key generation (KeyGen), encapsulation (Encaps), and decapsulation (Decaps). By using these functions, two parties can establish a shared secret as follows. First, the initiator generates a key pair consisting of an encapsulation key ek, which is public and sent to the responder, and a private decapsulation

[2] A module can be seen as a lattice, but is defined over a ring (other than the ring $\mathbb{Z}$).

key dk. The responder calls the Encaps function to generate a random 32-byte secret s and encapsulate (i.e., encrypt) it using the received ek. Encaps outputs s both in the clear (for use by the responder) and in encapsulated form, i.e., as ciphertext ct, which is sent to the initiator. Finally, the initiator executes the Decaps function with ct and dk as input to obtain the secret s.

2.3 PQNoise: KEM-Based Post-Quantum Noise

Key establishment based on a KEM such as Kyber differs from a key exchange system (e.g., DH, ECDH) in some basic aspects. For example, in DH, the two involved parties exchange their public keys, whereas, when using a KEM, one party sends a public key but the other party a ciphertext (i.e., an encapsulated secret). Furthermore, when executing a DH key exchange, each party can send its public key straight away, without having to wait first for the other party's key. When the public keys have been pre-distributed, it is possible for the two parties to agree on a shared secret without any interaction (the other party can even be "off-line"). In a KEM-based key establishment, on the other hand, the secret to be shared can only be encapsulated after receipt of the other party's key, and decapsulation can only be done after encapsulation. Performing Non-Interactive Key Exchange (NIKE) using a lattice KEM is feasible [10], but (in contrast to DH) unattractive in practice due to very large public keys.

Despite these differences between key exchange and key encapsulation, it is possible to design AKE protocols that use a KEM like Kyber instead of DH as underlying public-key cryptosystem. PQNoise [3] is an attempt to leverage the Kyber KEM for making Noise ready for the post-quantum era. However, since Kyber can not serve as a "drop-in" replacement for DH, most Noise handshake patterns require some substantial modifications. Instead of transmitting public DH keys, one party now sends its public (encapsulation) KEM key, which can be either static or ephemeral, and the other party returns a ciphertext (i.e., an encapsulated secret). Accordingly, the DH operations of classical Noise have to be replaced by executions of Encaps or Decaps. To facilitate the transition from DH to Kyber, the PQNoise designers introduced two new tokens:

- `"ekem"`: The sender generates a secret key and encapsulates it using the receiver's ephemeral public KEM key. Thereafter, a message containing the produced ciphertext is sent to the receiver. Both parties hash this message into the handshake hash `h` and the shared secret key into the chaining key `ck`, which triggers the generation of a new encryption key `k`.
- `"skem"`: This token is the static counterpart to `"ekem"`. The only difference is that the message is encrypted should an encryption key `k` be available.

The two Noise tokens `"e"` and `"s"` also exist in PQNoise and serve the same purpose, but with public KEM keys instead of public DH keys. Bigger changes were necessary for the tokens `"ee"`, `"se"`, `"es"`, and `"ss"`; they were dropped and superseded by `"ekem"` and `"skem"`. An `"ee"` token can be replaced by an `"ekem"` token in a straightforward way. The `"es"` and `"se"` tokens are replaced

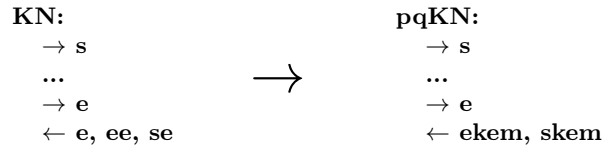


Fig. 2. Transition of Noise (`KN` pattern) to PQNoise (`pqKN` pattern)

by a `"skem"` token at the side that is supposed to have the other party's public static key, i.e., the initiator for `"es"` and the responder for `"se"`. However, this does not work for message patterns where an `"es"` or a `"se"` token is preceded by a `"s"` token (i.e., the static key is transmitted to the other party) since, in PQNoise, a `"skem"` token and the related `"s"` token can not appear in one and the same message pattern. In these cases, the `"skem"` token has to be moved to the next message pattern (going in opposite direction), which can increase the overall number of messages. Finally, the `"ss"` token has to be replaced by two `"skem"`, one in each direction, which may also result in an additional message being necessary for a PQNoise handshake compared to its Noise counterpart.

Figure 2 shows the transition from Noise to PQNoise using the `KN` pattern as example. There is no change at all in the pre-message pattern (`"s"`) and the first message pattern (`"e"`). The second message pattern does not contain the second `"e"` token anymore since PQNoise handshakes generally have only one `"e"` and one `"ekem"` token, respectively. In addition, the `"ee"` token is replaced by `"ekem"` and `"se"` by `"skem"`. Here, the `"skem"` token remains in the same message pattern as `"se"`, which, as mentioned above, is not always the case.

Figure 3 illustrates post-quantum versions of the 12 fundamental handshake patterns as presented in [3, Fig. 2]. `pqNX` is an example for a handshake pattern requiring an additional message (from the initiator to the responder) since the `"es"` token of `NX` was replaced by `"skem"`, and this `"skem"` can not appear in the same message pattern as `"s"`. This is actually the case for all handshakes containing an `X` in the handshake name, meaning that at least one static key is transmitted. However, `IN` and `IK` show that a `"s"` token in a message pattern does not always increase the overall number of messages.

3 Hybrid Noise

The security of classical public-key cryptosystems like DH and ECDH has been scrutinized for decades. Lattice-based post-quantum schemes like Kyber have not (yet) undergone similarly rigorous cryptanalytic effort. Therefore, it makes sense to deploy hybrid security protocols that combine pre- and post-quantum cryptography, especially during a transition period until the confidence in post-quantum cryptosystems is sufficiently high to allow them to stand on their own feet. A properly-designed hybrid protocol remains secure even when either the pre-quantum or the post-quantum component gets broken (but not both).

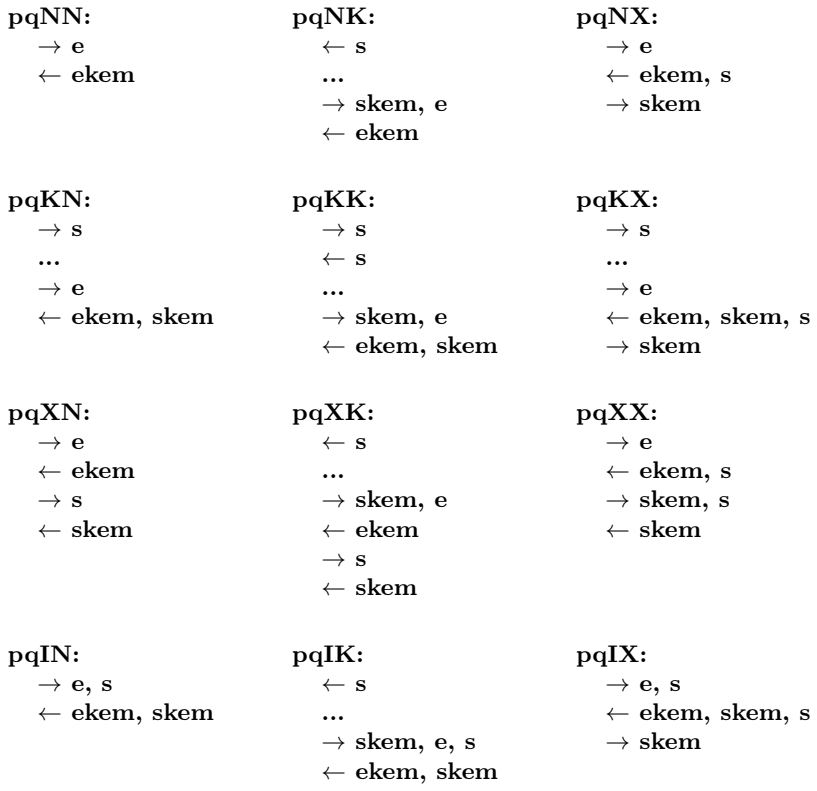


Fig. 3. The 12 fundamental interactive PQNoise patterns

As mentioned in Sect. 1, there was a previous attempt of creating a hybrid version of the Noise protocol framework, which had the goal to achieve Hybrid Forward Secrecy (HFS) [18]. Two new tokens, `"e1"` and `"ekem1"`, were introduced and served the purpose of adding ephemeral KEM keys to the classical (DH-based) Noise handshake patterns. The resulting hybrid handshakes were specified through a `"hfs"` pattern modifier. However, the static keys (used to authenticate the ephemeral keys) were still solely DH keys since post-quantum ephemeral keys are sufficient for post-quantum forward secrecy. Note that, as also mentioned in Sect. 1, the reference C implementation of the HFS patterns uses NewHope instead of the NIST-standardized Kyber KEM.

The main goal of the HFS patterns was to provide protection against the "harvest now decrypt later" threat scenario. Our hybrid Noise handshakes aim for a (slightly) different goal, namely to ensure full post-quantum security, like PQNoise, but with a "safety net" in form of the classical DH-based tokens. The purpose of this safety net is to guarantee pre-quantum attack resistance in the (hopefully unlikely) case that ongoing or future cryptanalysis of Kyber reveals a flaw that could be exploited by a pre-quantum adversary.

KNhyb:
→ **s, sh**
...
→ **e, eh**
← **e, ee, se, ekemh, skemh**

Fig. 4. Hybrid `KN` handshake pattern

Our hybrid handshakes can be seen as a "unification" of the classical (DH-based) Noise handshakes and PQNoise (using the Kyber KEM). In contrast to the HFS patterns, our hybrid handshakes use both static DH keys and static KEM keys. Consequently, each message pattern of our hybrid handshake is the combination of the tokens of the corresponding message patterns of the normal and the post-quantum handshake. The operations associated with each token remain the the same, but we have two sets of tokens, representing classical DH and post-quantum KEM operations, respectively. Furthermore, the ephemeral KEM keys are generally encrypted (when possible). The order of the tokens is kept consistent for each token-set to not interfere with the security properties that they provide on their own. In this way, the individual security guarantees of each pattern carry over to the hybrid pattern. An obvious explanation is the fact that adding KEM-keys to a Noise pattern (or, conversely, adding DH-keys to a PQNoise pattern) can be seen as a "mixing" of pseudo-random data into the hash object on both sides at some stage of the handshake execution. This does not imply that the pre- and post-quantum security properties are always the same in each stage of the handshake, but in the end the hybrid handshake inherits the security guarantees that each of them individually provides.

We created hybrid variants of all 12 fundamental interactive Noise patterns and integrated them to the reference implementation Noise-C [26]. Apart from keeping the tokens of the respective pattern (i.e., Noise, PQNoise) in the same order, we also grouped them into ephemeral and static tokens whenever it was possible. We generally put the ephemeral DH tokens at the very front, followed by the ephemeral KEM tokens, followed by the static tokens (again DH before KEM). Our goal was to enable the encryption of any KEM key prior its transmission, using a symmetric key derived, in part, through the corresponding DH equivalent. Figure 4 depicts a concrete example, namely the hybrid form of the `KN` handshake. The `"h"` tokens here are equivalents of the PQNoise tokens and come from the post-quantum version of the pattern. We just appended `"h"` to distinguish them from the original PQNoise patterns, though they represent, in essence, the same operations. All other tokens are regular Noise tokens.

4 Evaluation

The bulk of past research on the Noise framework has focused on analyzing its security; this includes work on the formal verification of Noise patterns [13] as well as the development of a novel security model to better analyze and prove

their security properties. This model enabled full security proofs for individual (classic and post-quantum) patterns in [8,3]. However, our evaluation will focus on the (relative) efficiency of regular Noise, PQNoise, and our hybrid Noise. As already mentioned earlier, we integrated the latter two to the Noise-C reference implementation, which means all three Noise versions use the same set of low-level cryptographic functions. We built the three Noise versions using the same compiler settings to ensure a fair and consistent evaluation. However, since the overall execution time of a security protocol depends on both its computation cost and its communications cost, it makes sense to do the efficiency evaluation under different network conditions. To achieve this, we utilized a Linux feature called *network namespaces* to set up an emulated network for which numerous parameters can be configured, e.g., Round Trip Time (RTT), throughput, and packet-loss rate. This feature was used before for evaluations of post-quantum TLS [21,15,12] and a comparison of Go implementations of the (regular) Noise `KK` and `XX` patterns with their respective PQNoise counterparts [3].

For our experiments, we ran the regular, post-quantum, and hybrid variants of the `NN`, `NK`, `NX`, and `XX` patterns. We chose those as, in our opinion, they have an interesting mix of properties when it comes to differences between the three variants of each pattern. In the case of `NN`, the message flow is the same across all three variants, but the public-key (resp., ciphertext) sizes differ. Hence, this pattern will illustrate how an increased amount of data transferred between the parties (but with the same number of messages) impacts the overall handshake latency. The `NK` pattern is an indicator for how the presence of pre-distributed static key(s) on one side changes the picture. Similar to `NK`, also the `NX` pattern involves static key(s) on one side, but they are transmitted during handshake execution. Therefore, `NX` allows us to assess how static-key transfers impact the three variants. Finally, `XX` is interesting because its post-quantum and hybrid variants require an additional message, i.e., we can expect significant differences between those two and the regular variant.

In addition, we compared the three Noise variants with TLS 1.3, more concretely OpenSSL version 3.0.2 with the provider from the Open Quantum Safe (OQS) project [25]. We configured OpenSSL such that is uses exactly the same public-key algorithms as Noise, namely X25519 for pre-quantum key exchange and Kyber512 for post-quantum key encapsulation. Note that all protocols we compare execute a Kyber512 implementation from the OQS project, which is configured to utilize AVX2 instructions on modern Intel CPUs. However, there are some caveats to consider when comparing Noise and TLS. The static keys used by regular Noise to authenticate the ephemeral DH key exchange are also DH keys, whereas TLS implements a form of "Signed Diffie-Hellman," i.e., the exchanged DH or ECDH keys are signed with static (i.e., long-term) signature keys. Furthermore, in TLS, the static signature keys are authenticated with the help of certificates, whereas, in Noise, it is "up to the application to determine whether the remote party's static public key is acceptable" [19, Sect. 14]. Since post-quantum signatures are outside the scope of this paper, we evaluated the post-quantum and hybrid variants of TLS with ECDSA signatures.

Table 1. Main characteristics of the devices used for our experiments

Device	Processor	Architecture	Frequency	Operating system
Laptop	Intel Core i5-10210U	64 bit	2.1 GHz	Ubuntu 22.04.4
Board	ARM Cortex-A7	32 bit	1.3 GHz	Armbian 24.5.1

In order to assess the (relative) performance of the protocols, we conducted experiments on a laptop with an Intel Core i5-10210U CPU. Moreover, we ran these experiments also on an Orange Pi One development board housing a 32-bit ARM Cortex-A7 processor, which serves as an example for a less-powerful mobile device and allows us to determine how the computing power impacts the results under the same network conditions. As shown in Table 1, both devices had a recent version of Ubuntu Linux installed on them; for the Orange board we went with a version from Armbian [4] that was optimized for this board. To get results that are as consistent as possible, we set all cores to performance on both devices and disabled hyperthreading and turbo boost on the laptop (the Cortex-A7 on the board lacks these features). We pinned the execution of the server and client to one core each and gave those highest execution priority.

To have control over the network conditions (e.g., latency, throughput), we emulated a network using the Linux tools mentioned before. The overall setup is the same for every experiment; we have two network namespaces, *cli_ns* and *srv_ns*, that ran the client-side and server-side of our experiments. These two namespaces were connected using a pair of virtual ethernet interfaces, i.e., one interface in each namespace, named *cli_ve* and *srv_ve*, respectively. This emulated network was set up separately on both the board and the laptop. For the experiments, we varied three major attributes of the network: RTT, packet-loss rate, and throughput. Since we want to compare the protocols under situations that mobile devices can encounter, we oriented ourselves on typical properties of 4G networks, since, on a global scale, they are the most widely-used mobile network infrastructure. For these networks, a throughput of roughly 10 Mbps and latencies of 60–70 ms (i.e., RTTs of 120–140 ms) seem common [9]. We ran our experiments with the following three combinations of network settings:

- RTT 10 ms, throughput 1000 Mbps: This is a special setting, simulating an ideal network scenario that is not supposed to be representative of a real-world 4G mobile network. Instead, the purpose of this setting is to serve as a baseline to assess how changing the RTT or throughput affects results.
- RTT 100 ms, throughput 10 Mbps: This setting represents a network close to what an average (or slightly above-average) 4G network might be.
- RTT 200 ms, throughput 10 Mbps: This setting represents long-distance connections with high latency, but with the same average throughput.

For all three sets of network conditions, we varied the packet-loss rate from 0% to 20%. In most network usage scenarios, a packet-loss rate of under 1% is deemed acceptable and in the majority of cases remains below 5%; going up to 20% allowed us to simulate even particularly bad network conditions.

Table 2. Average initiator-side computation time in ms

	Laptop				Board			
Variant	`NN`	`NK`	`NX`	`XX`	`NN`	`NK`	`NX`	`XX`
Noise	0.19	0.30	0.30	0.41	2.67	4.81	4.82	7.04
PQNoise	0.13	0.18	0.21	0.29	2.05	3.12	3.21	4.55
Hybrid	0.31	0.45	0.50	0.70	4.71	8.11	8.17	11.54

5 Results and Discussion

To collect experimental results, we ran the handshake of the different protocols 1000 times, excluding warm-up runs, and measured the time taken. The overall handshake time includes the overhead introduced by low(er) network layers, in particular TCP, to establish a connection. For every experiment, we collected the handshake times at the median, the 75th, and the 95th percentile to hint on how different network conditions affect 50%, 75%, and 95% of a userbase.

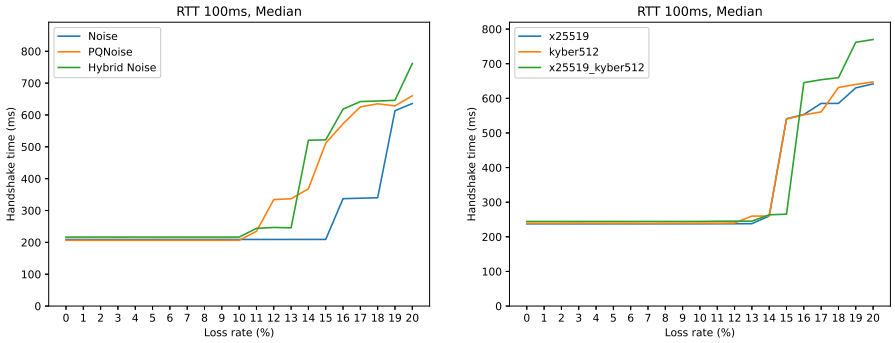
5.1 Computation Costs

We start with analyzing and comparing the computation cost of the Noise protocol variants on the laptop and the board. Table 2 summarizes the execution times of the computational parts (mostly cryptographic operations and various auxiliary functions) of the protocols executed by the initiator. These timings do not include the communication part of the handshake, such as socket functions for initialization of the connection and sending/receving of data. The laptop is between 13 and 18 times faster than the board, which is not very surprising in light of the enormous architectural and micro-architectural differences between a 64-bit Intel CPU and a (much simpler) 32-bit ARM Cortex-A processor. It is important to note that, on the laptop, the computational part of all protocols is less than one hundredth of the RTT of 100 ms, and on the board it is less than one tenth (except for `XXhyb`). When looking at the different patterns, it turns out that the computational cost correlates with the number of tokens (i.e., the number of DH or KEM operations). For example, the computation times of the regular `NN` and `XX` patterns differ by a factor of more than two, and the `NK` and `NX` pattern are roughly in the middle between them.

When comparing the computation times between the Noise variants, we can see that the PQNoise patterns are less costly than the Noise patterns; this is in line with software benchmarking results reported in the literature, according to which Kyber512 outperforms X25519. The computation times of all our hybrid patterns are almost exactly the sum of the related Noise and PQNoise patterns since the way we put them together does not entail additional computations.

5.2 Comparison of Noise variants with TLS variants

Figure 5a shows the median handshake times (including both computation and communication) of the pre-quantum, post-quantum, and hybrid variants of the



(a) Noise NX results (median) (b) TLS results (median)

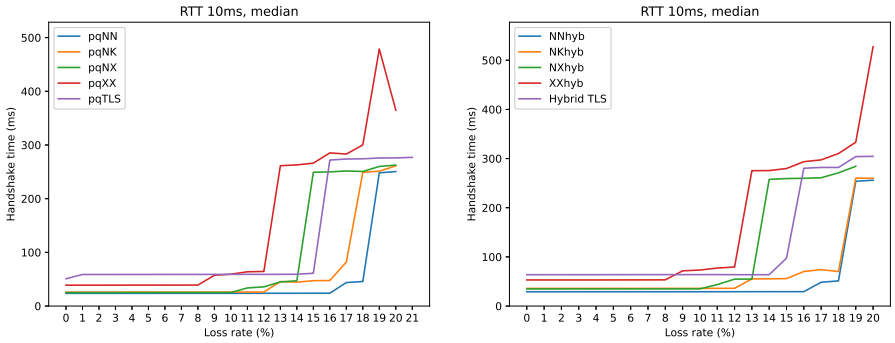
Fig. 5. NX and TLS variants comparison for RTT = 100 ms, measured on the board

NX pattern for an RTT of 100 ms, measured on the board. The handshake times of the variants are very similar (for reasonable packet-loss rates) and dominated by the communication cost, i.e., the time spent for message transmission(s). In fact, the pqNX pattern is marginally faster than regular NX, which is due to the lower computation cost, irrespective of the larger key sizes. A Kyber512 public key is $800 - 32 = 768$ bytes longer than an X25519 public key, but when the throughput is 10 Mbps, the transmission of these additional 768 bytes requires only 0.61 ms and is negligible compared to the RTT. When the packet-loss rate exceeds 10%, the full handshake time of the post-quantum and hybrid variants increases versus the regular NX pattern. The hybrid variant is pretty similar to the post-quantum variant, at both low and high packet-loss rates.

Figure 5b illustrates the handshake execution times of pre-quantum, post-quantum, and hybrid TLS (with server authentication) on the board, measured for a 100-ms RTT. For reasonably low packet-loss rates, the handshake times are very similar across the three variants, but about 12% slower than the Noise NX variants. However, as explained in the previous section, a direct comparison of Noise and TLS is not easily possible due to certain differences between these protocols. First, the post-quantum and hybrid TLS variants use ECDSA signatures to sign the ephemeral public keys and ciphertexts transmitted during the handshake. They protect against "harvest now decrypt later" threats, but, in contrast to PQNoise and hybrid Noise, are vulnerable to impersonation by an adversary capable to forge an ECDSA handshake signature. Second, the static key used in TLS to authenticate the key agreement is itself authenticated by an (in our case self-signed) ECDSA certificate, which is not the case for Noise.

5.3 Pattern Comparisons

In this subsection, we present results for all four Noise handshake patterns we consider in this paper (i.e., NN, NK, NX, and XX) and compare them directly with each other (and with TLS), whereby we focus on the post-quantum and hybrid

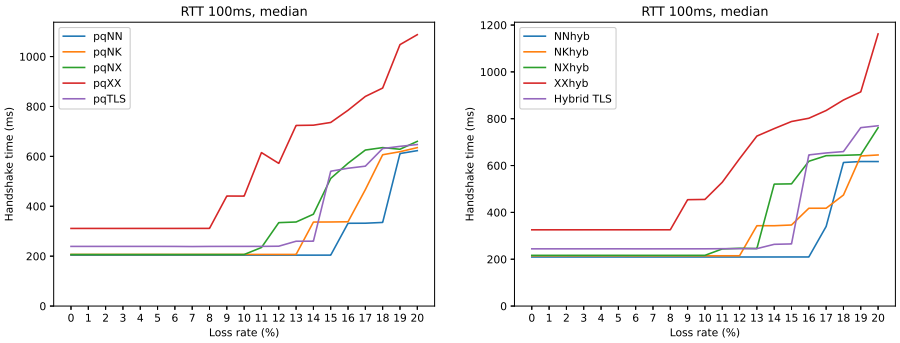


(a) Post-quantum results (median) (b) Hybrid results (median)

Fig. 6. Direct comparison for the post-quantum and hybrid variants of all protocols in the idealized network scenario (i.e., RTT of 10 ms, throughput of 1000 Mbps)

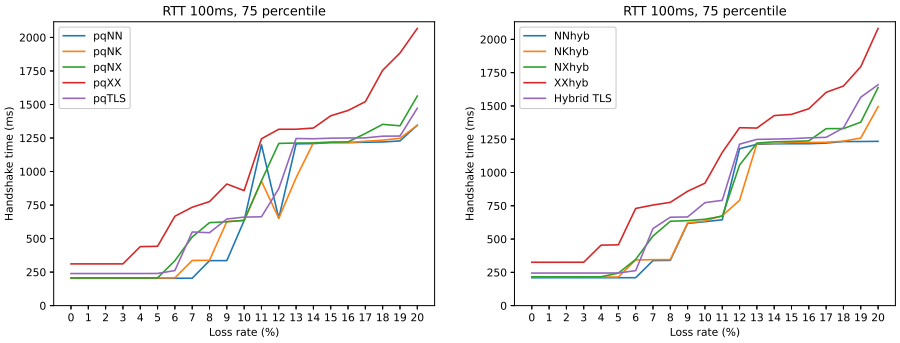
variants. Under realistic 4G network conditions (i.e., an RTT of 100 ms), the total handshake times are largely dominated by the connection latencies, while the computation times play a minor role. Therefore, we decided to evaluate the Noise patterns also under "idealized" network conditions, i.e., an RTT of 10 ms and a throughput of 1000 Mbps, which will make the computation costs more apparent. Figure 6a depicts the median handshake times of the post-quantum variants of the four patterns under these idealized conditions, measured on the ARM board. For low packet-loss rates, the `pqNN`, `pqNK`, and `pqNX` patterns are nearly identical. The `pqXX` pattern is the by far slowest, which can be explained by the fact that it has the highest computational cost (i.e., one KeyGen, three Encaps, and three Decaps in total) and highest communication cost (i.e., three public keys and three ciphertexts in four messages) of all patterns. When the packet-loss rate increases, the `pqXX` pattern is earlier and more severely affected than the other patterns. Two of the four messages of the `pqXX` pattern contain a Kyber512 public key and a ciphertext, i.e, their length exceeds the Maximum Transmission Unit (MTU) of 4G networks, which is a bit below 1500 bytes. As a consequence, two `pqXX` messages get segmented, thereby making this pattern more susceptible to packet losses, which start to become noticeable at 9% loss rate. The post-quantum TLS protocol performs better than `pqXX` for loss rates above 9%, but worse for lower loss rates.

Figure 6b illustrates the handshake times for the hybrid variants of the protocols, again for an RTT of 10 ms. For low packet-loss rates, the order is the same as for the PQNoise patterns: `NNhyb` is the fastest, followed by `NKhyb` and `NXhyb`, which are almost identical. `XXhyb` is the slowest among the four hybrid patterns, mainly for two reasons: (i) it has the highest computational cost (see Table 2), and (ii) similar to `pqXX`, it requires the transmission of four messages (i.e., one message more than the post-quantum/hybrid `NK` and `NX` patterns). In summary, the "hybridization" of PQNoise causes relatively little overhead, and this little overhead will become even less significant for an RTT of 100 ms.



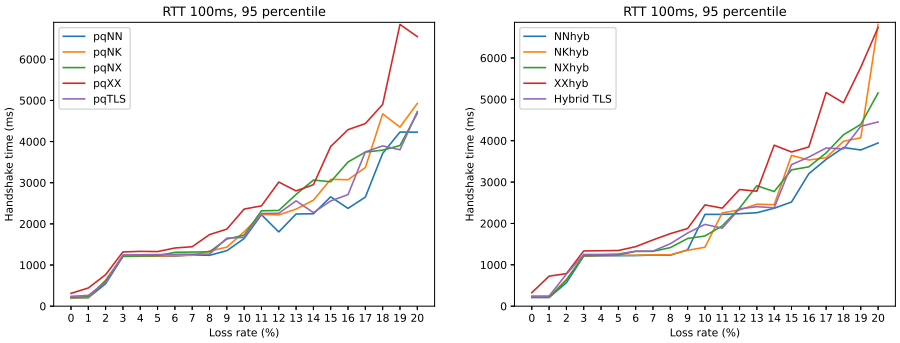
(a) Post-quantum results (median)

(b) Hybrid results (median)



(c) Post-quantum results (75th percentile)

(d) Hybrid results (75th percentile)



(e) Post-quantum results (95th percentile)

(f) Hybrid results (95th percentile)

Fig. 7. Direct comparison for the post-quantum and hybrid variants of all protocols in the realistic network scenario (i.e., RTT of 100 ms, throughput of 10 Mbps)

Figure 7 shows direct comparisons between the different post-quantum protocol variants and their hybrid counterparts for a realistic 4G network scenario (i.e., for an RTT of 100 ms). The graphs represent the handshake times in the average case and for the 75th and the 95th percentile, respectively. Overall, the

median results for the post-quantum Noise patterns are very similar to that in the "idealized" network setting. For low packet-loss rates, the `pqNN`, `pqNK`, and `pqNX` patterns have almost the same execution times, while `pqXX` is by far the slowest PQNoise pattern. The high handshake time of `pqXX` is primarily due to high communication cost, originating from the transmission of four messages in total, half of which are segmented into two packets each. We can also observe that Noise `pqXX` and post-quantum TLS swapped their positions compared to the idealized network scenario, i.e., `pqXX` is now slower than TLS. Again, this can be explained by the high communication cost of `pqXX` (Kyber512 keys and ciphertexts are much larger than ECDSA signatures). The median handshake times of the hybrid protocols are, for low loss rates, basically indistinguishable from their post-quantum counterparts.

When the packet-loss rate increases, the relative order of the four PQNoise patterns in the median case remains basically the same. The `pqNN` pattern has the lowest number of messages, and also the lowest number of packets, transmitted between the two parties, followed by `pqNK`, `pqNX`, and finally `pqXX`. We can observe from Fig. 7a that packet losses start to impact the `pqNN` pattern only at a rather high loss rate of above 15%. On the other hand, the `pqNK` and `pqNX` patterns get affected at loss rates above 13% and 10%, respectively. This relatively big difference between the two patterns is caused by the responder's static key, which is pre-shared in `pqNK`, but transmitted during the handshake in `pqNX`, i.e., packet losses affect `pqNX` more severely. `pqXX` is the most sensitive pattern when it comes to packet losses as it's handshake time starts to increase already at a loss rate of above 8%. Overall, these observations hold true for the median, 75th, and 95th percentile, though with each percentile the differences between the patterns become smaller. The hybrid handshakes, depicted on the right of Fig. 7, suffer from packet losses in a roughly similar way as PQNoise.

As stated in Sect. 4, the typical packet-loss rate of a 4G network is around 1% and rarely exceeds 5%. Our results indicate that for all post-quantum and hybrid patterns except the `XX` variants, 95% of the user base in a network with a 1% loss rate would not experience any impact by lost packets. The `XX` variants start to lose a packet already at a 1% loss rate in the 95th percentile, which is little surprising when considering that they transmit twice as many packets as the `pqNN` pattern. According to Fig. 7e, the `pqNN` handshake pattern starts to lose its first packet at a 2% loss rate in the 95th percentile. Even at a high loss rate of 5%, about 75% of network users are unaffected by packet losses, as can be concluded from Fig. 7c and 7d. The only protocols that start to lose packets below a 5% loss rate in the 75th percentile are the `XX` variants.

6 Conclusions

Using the recently-proposed PQNoise protocol framework as starting point, we developed and implemented hybrid variants of the 12 fundamental interactive Noise patterns. We determined the handshake execution times of regular, post-quantum and hybrid variants of the `NN`, `NK`, `NX`, and `XX` patterns on two mobile

devices, a laptop and an ARM development board. This evaluation was carried out under different network conditions (i.e., RTT, throughput, and packet-loss rate) and took both the computation and communication cost into account.

Overall, the PQNoise and hybrid variants performed very well in comparison to the classical Noise patterns, with only the slightly higher amount of packets becoming an issue at high packet-loss rates. We noticed the biggest differences among the three variants for the `XX` pattern, partly due to high computational cost (in both the pre- and post-quantum setting) and partly because the post-quantum and hybrid variants entail an additional handshake message. We also found that the overheads of the hybrid patterns versus PQNoise are very small and mainly caused by the additional computing time for X25519, especially on the board. The additional communication cost for transmission of X25519 keys is negligible and does also not cause extra packet fragmentation for any of the patterns. Furthermore, the additional computation cost gets less significant the better the computing power of the mobile device and the worse the RTT of the network becomes. Putting everything together, the experiments and results we presented in this paper encourage the deployment of hybrid Noise handshakes rather than purely post-quantum variants.

Acknowledgments. This research was funded in part by the Luxembourg National Research Fund (FNR), grant reference C21/IS/16221219/ImPAKT. The full source code of Noise-C, including the post-quantum and hybrid patterns, and further results are available at `https://github.com/JoshuaRenckens/PQNoise_Master_Thesis`.

References

1. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange – A new hope. In: Proceedings of the 25th USENIX Security Symposium (USS 2016). pp. 327–343. USENIX Association (2016)
2. Alnahawi, N., Müller, J., Oupickỳ, J., Wiesmaier, A.: A comprehensive survey on post-quantum TLS. IACR Communications in Cryptology **1**(2) (Jul 2024)
3. Angel, Y., Dowling, B., Hülsing, A., Schwabe, P., Weber, F.: Post quantum Noise. In: Proceedings of the 29th ACM Conference on Computer and Communications Security (CCS 2022). pp. 97–109. ACM (2022)
4. Armbian Operating System Community: Armbian: Linux for ARM development boards. Available online at `https://www.armbian.com` (2024)
5. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber: Algorithm Specifications and Supporting Documentation (Version 3.02) (2021)
6. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (Nov 1976)
7. Donenfeld, J.A.: WireGuard: Next generation kernel network tunnel. Available for download at `https://www.wireguard.com/papers/wireguard.pdf` (2020)
8. Dowling, B., Rösler, P., Schwenk, J.: Flexible authenticated and confidential channel establishment (fACCE): Analyzing the Noise protocol framework. In: Public-Key Cryptography — PKC 2020. Lecture Notes in Computer Science, vol. 12110, pp. 341–373. Springer (2020)

9. EE Limited: 4G Speed. Available online at https://business.ee.co.uk/help/network-and-coverage/4g-speed-what-you-can-expect/ (2024)
10. Gajland, P., de Kock, B., Quaresma, M., Malavolta, G., Schwabe, P.: SWOOSH: Efficient lattice-based non-interactive key exchange. In: Proceedings of the 33rd USENIX Security Symposium (USS 2024). pp. ??–?? USENIX Association (2024)
11. Ho, S., Protzenko, J., Bichhawat, A., Bhargavan, K.: Noise*: A library of verified high-performance secure channel protocol implementations. In: Proceedings of the 43rd Symposium on Security and Privacy (S&P 2022). pp. 107–124. IEEE (2022)
12. 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. Cryptology ePrint Archive, Paper 2024/176 (2024)
13. Kobeissi, N., Nicolas, G., Bhargavan, K.: Noise explorer: Fully automated modeling and verification for arbitrary Noise protocols. In: Proceedings of the 4th European Symposium on Security and Privacy (EuroS&P 2019). pp. 356–370. IEEE (2019)
14. Langley, A., Hamburg, M., Turner, S.: Elliptic curves for security. Internet Engineering Task Force, Internet Research Task Force, RFC 7748 (Jan 2016)
15. Paquin, C., Stebila, D., Tamvada, G.: Benchmarking post-quantum cryptography in TLS. In: Post-Quantum Cryptography — PQCrypto 2020. Lecture Notes in Computer Science, vol. 12100, pp. 72–91. Springer (2020)
16. Paterson, K.G., van der Merwe, T.: Reactive and proactive standardisation of TLS. In: Security Standardisation Research — SSR 2016. Lecture Notes in Computer Science, vol. 10074, pp. 160–186. Springer (2016)
17. Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J.P., Saraiva, J.: Ranking programming languages by energy efficiency. Science of Computer Programming **205** (2021)
18. Perrin, T.: KEM-based Hybrid Forward Secrecy for Noise. Specification, available for download at https://github.com/noiseprotocol/noise_hfs_spec/blob/master/output/noise_hfs.pdf (2018)
19. Perrin, T.: The Noise Protocol Framework, Revision 34. Specification, available for download at https://noiseprotocol.org/noise.pdf (2018)
20. Rescorla, E.K.: The Transport Layer Security (TLS) Protocol Version 1.3. Internet Engineering Task Force, Network Working Group, RFC 8446 (Aug 2018)
21. Schwabe, P., Stebila, D., Wiggers, T.: Post-quantum TLS without handshake signatures. In: Proceedings of the 27th ACM Conference on Computer and Communications Security (CCS 2020). pp. 1461–1480. ACM (2020)
22. Sheffer, Y., Holz, R., Saint-Andre, P.: Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS). Internet Engineering Task Force, Using TLS in Applications Working Group, RFC 7457 (Feb 2015)
23. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS 1994). pp. 124–134. IEEE (1994)
24. Stallings, W.: Cryptography and Network Security: Principles and Practice. Pearson, 7th edn. (2016)
25. Stebila, D., Mosca, M.: Post-quantum key exchange for the Internet and the Open Quantum Safe project. In: Selected Areas in Cryptography — SAC 2016. Lecture Notes in Computer Science, vol. 10532, pp. 14–37. Springer (2016)
26. Weatherley, R.: Noise-C: A plain C implementation of the Noise protocol. Source code, available online at https://github.com/rweather/noise-c (2023)
27. WhatsApp LLC: WhatsApp encryption overview. Technical white paper, available for download at http://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf (2020)