Post-quantum WireGuard

Hulsing, Andreas; Ning, Kai Chun; Schwabe, Peter; Weber, Florian; Zimmermann, Philip R.

DOI
10.1109/SP40001.2021.00030

Publication date
2021

Document Version
Final published version

Published in
2021 IEEE Symposium on Security and Privacy (SP)

Post-quantum WireGuard

Andreas Hülsing
Eindhoven University of Technology
The Netherlands
andreas@huelsing.net

Kai-Chun Ning
KPN B.V.
The Netherlands
kaichun.ning@kpn.com

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

Florian Weber
Eindhoven University of Technology
The Netherlands
mail@florianjw.de

Philip R. Zimmermann
Delft University of Technology & KPN B.V.
The Netherlands
prz@mit.edu

2021 IEEE Symposium on Security and Privacy (SP) | 978-1-7281-8934-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/SP40001.2021.00030

***Abstract*—In this paper we present PQ-WireGuard, a post-quantum variant of the handshake in the WireGuard VPN protocol (NDSS 2017). Unlike most previous work on post-quantum security for real-world protocols, this variant does not only consider post-quantum confidentiality (or forward secrecy) but also post-quantum authentication. To achieve this, we replace the Diffie-Hellman-based handshake by a more generic approach only using key-encapsulation mechanisms (KEMs). We establish security of PQ-WireGuard, adapting the security proofs for WireGuard in the symbolic model and in the standard model to our construction. We then instantiate this generic construction with concrete post-quantum secure KEMs, which we carefully select to achieve high security and speed. We demonstrate competitiveness of PQ-WireGuard presenting extensive benchmarking results comparing to widely deployed VPN solutions.**

I. Introduction

WireGuard is a VPN protocol presented by Donenfeld in [1]. It combines modern cryptographic primitives with a simple design derived from the Noise framework [2], a very small codebase, and very high performance.

These properties are achieved partially because WireGuard is "cryptographically opinionated" [1]: instead of supporting multiple cipher suites, WireGuard fixes X25519 [3][1] for elliptic-curve Diffie-Hellman key exchange, Blake2 [4] for hashing, and ChaCha20-Poly1305 [5], [6], [7] for authenticated encryption. Not only are those primitives known for their outstanding software performance, fixing those primitives eliminates the need for an algorithm-negotiation phase, which keeps the protocol simple and its codebase small, and avoids any potential negotiation attacks. Also, high performance is achieved by implementing the protocol in the Linux kernel space, which eliminates the need for moving data between user and kernel space.

In addition to its superior performance and small codebase, WireGuard was designed to provide security properties that are not supported by other VPN software, e.g., identity hiding, and DoS-attack mitigation. The security considerations that lead to the design of WireGuard are laid out in [1]. Donenfeld and Milner give a computer-verified proof of the protocol in the symbolic model in [8]. In [9] Dowling and Paterson present a computational proof of the WireGuard handshake with an additional key-confirmation message.

Given its properties it is thus not surprising to see that WireGuard is becoming increasingly popular. For example, CloudFlare is working on "BoringTun", a WireGuard-based userspace VPN solution written in Rust [10]. Torvalds called WireGuard's codebase a "work of art" compared to OpenVPN and IPsec and advocated for its inclusion in Linux [11]. WireGuard is scheduled to become part of the next mainline Linux kernel (version 5.6).

As WireGuard aims to be the next-generation VPN protocol, it is natural to see that security against quantum attackers played a role in its design as well, albeit a small one. Specifically, it allows users to include a symmetric shared key into the handshake, which protects against an attacker who records handshake transcripts now and attacks them in the future with a quantum computer [1, Sec. V.B]. Post-quantum asymmetric schemes are explicitly declared as *"not practical for use here"* by Donenfeld and are thus not included in the handshake. Recently, Appelbaum, Martindale, and Wu took another look at post-quantum security of WireGuard and proposed a small tweak to the protocol that aims at protecting against pretty much the same future quantum attacker with recorded transcripts [12], but without requiring a long-term secure pre-shared key. The tweak assumes that public keys are typically not actually known to the attacker. If this is the case, then transmitting the hash of the public key instead of the public key itself prevents a future quantum attacker from ever learning the public key and thus from computing the corresponding secret key.

A. Contributions of this paper.

In this paper we present PQ-WireGuard, a post-quantum variant of the WireGuard handshake protocol. Unlike the mitigation techniques described above and unlike various earlier works aiming at transitioning protocols to post-quantum security, we do not only aim for *confidentiality* against quantum attackers, but target full post-quantum security *including*

Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf.

[1]For naming of X25519, see https://mailarchive.ietf.org/arch/msg/cfrg/-9LEdnzVrE5RORux3Oo_oDDRksU.

DOI 10.1109/SP40001.2021.00030

authentication. The main design goal of PQ-WireGuard is to stay as close as possible to the original WireGuard protocol in terms of security and performance characteristics, i.e., PQ-WireGuard should

- achieve all the security properties of WireGuard, but now also resist attacks using a large-scale quantum computer;
- make a concrete choice of high-security, efficient cryptographic primitives instead of including an algorithm negotiation phase;
- finish the handshake in just one round trip;
- fit each of the two handshake messages into just one unfragmented IPv6 packet of at most 1280 bytes; and
- achieve much higher computational performance than other VPN solutions such as IPsec or OpenVPN.

PQ-WireGuard manages to tick all these boxes and thus shows that the assessment from the original WireGuard paper stating that post-quantum security is "not practical for use here" is no longer correct.

From Diffie-Hellman to KEMs. The original WireGuard protocol is heavily based on (non-interactive) Diffie-Hellman key exchange, which is not easy to replace straight-forwardly with post-quantum primitives. The only somewhat practical post-quantum non-interactive key exchange is CSIDH [13], which is both very young and rather inefficient. Furthermore, the security of concrete CSIDH parameters is still heavily debated [14], [15], [16], [17]. We therefore take a different approach and first transform the WireGuard protocol to a version using only interactive key-encapsulation mechanisms (KEMs). This approach is based on the KEM-based authenticated key exchange described in [18].

Security. Security of WireGuard is supported by the symbolic proof of Donenfeld and Milner [8] and the computational proof by Dowling and Paterson [9]. The symbolic proof covers more security properties than the computational proof and is computer verified. However, a correct computational proof gives stronger security guarantees as the proof makes less idealizing assumptions. We adapt both proofs to the case of PQ-WireGuard and thereby establish the same level of security guarantees as WireGuard. On the way, we point out (and fix) a few small mistakes in the computational proof. In order to allow for a standalone proof of the handshake we add an explicit key confirmation message to the PQ-WireGuard handshake as suggested in [9].

A concrete instantiation. The generic KEM-based approach allows us in principle to use any post-quantum KEM submitted to the NIST post-quantum project as a proposal for future standardization[2]. Now the main challenge becomes one of public-key and ciphertext sizes: WireGuard operates over UDP and the existing codebase assumes that all handshake messages fit into one unfragmented IPv6 packet. The reason for this requirement is that increasing the number of packets in a handshake would make the state machine of the protocol more complex and contradict WireGuard's aim for simplicity in both protocol design and codebase. Fragmenting and reassembling IPv6 packets comes with various issues. For example a denial-of-service (DoS) attack can fill up the reassembly buffer with fragments of packets that are never completed. This is just one example of IP fragmentation attacks [19]. To prevent such attacks, some firewalls drop fragmented IPv6 packets, so avoiding fragmentation ensures that the protocol remains robust against such firewall configurations.

IPv6 packets are guaranteed not to be fragmented as long as they do not exceed 1280 bytes [20]. With the IPv6 header occupying 40 bytes and the UDP header occupying 8 bytes, there are 1232 bytes left for the content of handshake messages. In both, initiation message and response, those 1232 bytes need to fit several MACs and protocol-specific fields alongside a public key and a ciphertext (for the initiator's packet) respectively two ciphertexts (for the responder's packet). For some of the schemes proposed to NIST, this is not much of a problem. For example, compressed SIKE [21] uses only 331 bytes for the public key and 363 bytes for the ciphertext, even at the highest security level. However, SIKE is not exactly known for its high computational performance; for example, it is more than an order of magnitude slower than most lattice-based KEMs.

PQ-WireGuard uses a combination of two KEMs, namely Classic McEliece [22] and a passively secure variant of Saber [23], [24]. One advantage of this solution for actual applications is that most security properties are guaranteed by the Classic McEliece scheme, considered by many as the most conservative choice among all NIST candidates. Another advantage is the computational efficiency (see below). Finally, our approach allows us to give a concrete example of an application that

1) works extremely efficiently with Classic McEliece, a cryptosystem that is often discarded as "impractical" because of its large public keys; and
2) heavily benefits from the savings in public-key and ciphertext size that lattice-based KEMs can achieve if they do not aim for active security.

The second point may be seen as new insight into the question whether or not KEMs which only provide passive security really offer any benefits for real-world applications, which was repeatedly raised by Bernstein on the NIST pqc-forum mailing list [25], [26]. The parameters our proposal uses achieve the "AES-192-equivalent" security level (NIST level 3).

Performance evaluation. To evaluate the performance of PQ-WireGuard, we compare the handshake efficiency of PQ-WireGuard with that of WireGuard, the strongSwan implementation of IPsec, and OpenVPN. We show that a PQ-WireGuard handshake is less than 60% slower than a WireGuard handshake, is more than 5 times faster than an IPsec handshake using Curve25519, and more than 1000 times faster than an OpenVPN handshake.

B. Related Work.

Related work can be grouped in four categories.

[2]See https://csrc.nist.gov/Projects/Post-Quantum-Cryptography.

First, there is ongoing effort for post-quantum security in the Noise framework [2] that the WireGuard handshake is based on. Currently this effort only covers "transitional post-quantum security" (i.e., no post-quantum authentication), which is achieved by combining ephemeral-ephemeral ECDH with a post-quantum KEM (currently NewHope-Simple [27]). Noise calls this approach hybrid forward secrecy (HFS); the details are described in [28]. As the WireGuard handshake is one of the more complex Noise key-exchange patterns, our work may also be seen as a first step towards fully post-quantum Noise.

Second, there is a large body of work on authenticated key exchange including works on generic KEM-based constructions. Most important for this work is the generic KEM-based approach by Fujioka, Suzuki, Xagawa, Yoneyama [18] (which can be seen as a generalization of "Efficient one-round key exchange in the standard model" [29]). All currently considered actively secure post-quantum KEMs start in their construction from a passively secure encryption scheme and obtain active security through variants of the Fujisaki-Okamoto (FO) transform [30]. In [31], Hövelmanns, Kiltz, Schäge, and Unruh present a generic AKE construction that starts directly from passively secure encryption schemes and moves some of the FO machinery into the AKE construction. A somewhat similar idea of reconsidering the FO transform in the context of authenticated key exchange is presented by Xue, Lu, Li, Liang, and He in [32]. However, the primitive they start from in their generic construction is what they call a "2-key KEM". Also more specialized, non-generic, constructions of post-quantum AKEs have been described in the literature. In [33], Zhang, Zhang, Ding, Snook, and Dagdelen describe a lattice-based AKE (which, however, was later outperformed by instantiating a generic construction with the lattice-based KEM Kyber in [34, Sec. 5]). Isogeny-based constructions were presented by Longa in [35], by Xu, Xue, Wang, Au, Liang, and Tian in [36], and by Fujioka, Takashima, Terada, and Yoneyama in [37].

Third, there have been additional efforts on proving security properties of WireGuard and more generally Noise. Most notably, in [38], Lipp, Blanchet, and Bhargavan present a computer-verified proof of security of the WireGuard handshake in the computational model. The proof is in the ROM; a meaningful translation to PQ-WireGuard would require first moving this proof to the QROM or the standard model. In [39], Dowling, Rösler, and Schwenk introduce a generalization of the ACCE model from [40] and prove 8 out of the fundamental 15 Noise AKE patterns secure in this generalized ACCE model; the IK pattern used by WireGuard is not one of those 8 patterns. In [41], Kobeissi, Nicolas, and Bhargavan present "Noise Explorer", a tool that fully automatically proves certain security properties of Noise AKE patterns in the symbolic model using ProVerif [42]. Adapting Noise Explorer to support KEM-based AKE such as the one we use in this paper would certainly be interesting, but for the concrete case of PQ-WireGuard would not provide any more insight than our adaptation of the Tamarin [43] proof.

Finally, there are proposals to upgrade other VPN solutions to post-quantum security. Specifically, we are aware of two independent efforts to migrate OpenVPN [44] to post-quantum cryptography. One of these efforts is described in the Master's thesis by de Vries, which adds transitional security to OpenVPN through the use of McEliece as additional key exchange [45]. The other effort is PQCrypto-VPN by Easterbrook, Kane, LaMacchia, Shumow, and Zaverucha at Microsoft Research [46]. We give a performance comparison between our proposal and PQCrypto-VPN in Section VI.

C. Availability of Software.

Just like the Linux kernel module implementing the original WireGuard protocol, we make all software described in this paper available under the GPLv2 license. The software is available online from https://cryptojedi.org/crypto/#pqwireguard. Note that the optimized Classic-McEliece and the Saber software we make use of has been placed into the public-domain.

D. Organization of this paper.

Section II gives a brief summary of the cryptographic primitives involved in the WireGuard handshake and then reviews the full handshake. Section III introduces the abstract, KEM-based construction of the PQ-WireGuard handshake and Section IV analyzes its security. Section V describes the instantiation of PQ-WireGuard using McEliece and a passively secure version of Saber. Finally, Section VI presents benchmark results for PQ-WireGuard.

II. Preliminaries

In the following we briefly discuss the security notion under which we analyze PQ-WireGuard. We then recall some cryptographic primitives used by WireGuard and PQ-WireGuard, and eventually provide a brief description of the WireGuard handshake protocol. We start the discussion of security notions with a brief discussion of post-quantum security.

A. A Note on Post-Quantum Security

Our proofs in the computational model analyze the post-quantum security of PQ-WireGuard. This requires definitions of post-quantum security. In our case, the security notions for pre- and post-quantum security only differ with regard to the computational model of the attackers. More precisely, pre-quantum security assumes adversaries to be conventional probabilistic algorithms. For post-quantum security we assume adversaries to be quantum algorithms (which are probabilistic by nature). All honest parties are assumed to be conventional probabilistic algorithms. Consequently, all communication in our models is classical. We provide all definitions below with respect to probabilistic polynomial time (PPT) adversaries. We then obtain the post-quantum version by allowing the adversaries to be quantum polynomial time (QPT) algorithms.

This treatment is possible as our computational proofs are in the so-called standard model, i.e., the proofs do not make use of idealized primitives, like random oracles or ideal ciphers which would require quantum-access to oracles.

Looking at the symbolic model, we are not aware of research that analyzes the implications of considering quantum adversaries against conventional cryptography. Consequently, for now our proofs in the symbolic model are only known to apply to the pre-quantum setting.

B. Security Properties and Corruption Patterns

Before discussing formal models, we give some intuition on the security that WireGuard was designed to achieve. WireGuard considers a setting where an *initiator* I initiates a secure connection with a *responder* R. The WireGuard handshake aims to achieve the following security goals:

- *Session-Key Secrecy*: The established session key is pseudorandom, i.e., indistinguishable from a random bit string for everyone except the initiator and the responder.
- *Session-Key Uniqueness*: The established session key is, with overwhelming probability, never repeated.
- *Entity authentication*: Both, initiator and responder, know who they are talking to. Specifically, it is practically infeasible for a party to impersonate another party.
- *Identity Hiding*: The identities of initiator and responder are only revealed to each other.
- *DoS Mitigation*: By DoS mitigation we refer to the first message by the initiator being authenticated. This allows a responder to reject forged messages before performing any costly public-key operations.

These security goals should even be preserved under corruption of secrets. All parties have a static long-term secret (usually the secret key of a key-pair). Identity is defined as knowledge of a certain long-term secret. In addition, parties have ephemeral secrets (think of ephemeral keys but also the randomness used during the execution of the protocol[3]) which are only used in a single execution of the protocol and are erased afterwards. We consider these a party's secrets and assume that they may be corrupted independently by an adversary. In addition, every pair of parties may or may not have a pre-shared secret that can be corrupted by the adversary as well. This allows to define different corruption patterns. In general we consider *maximal exposure (MEX) attacks* [47, Sec. 3.3],[48],[18] allowing adversaries to corrupt arbitrary combinations of static and ephemeral secrets.

However, certain corruption patterns allow for trivial, unpreventable attacks against certain security goals. For example, if all long-term secret data is compromised, there is no way to protect against active adversaries. In the following we discuss under which corruption patterns which security goals should still be achieved, explicitly excluding such trivial attacks.

- *Session-Key Secrecy.* The session key remains pseudorandom if either the parties share an uncorrupted pre-shared key or if each party has at least one uncorrupted secret. This notion implies *perfect forward secrecy (PFS)* (also known as pre-compromise security) where an adversary learns the victim's secrets at some point in time and tries to learn the session key of previous sessions. In the case of weak-PFS the adversary is limited to sessions in which it did not actively interfere before.
- *Session-Key Uniqueness.* Session-key uniqueness holds against passive adversaries that only observe secrets of corrupted parties but do not actively change them.
- *Entity Authentication.* The handshake provides entity authentication under arbitrary corruption except for two cases. Assume Eve wants to impersonate Alice towards Bob then there exist two trivial corruption patterns.
  If Eve corrupts Alice's long-term secrets and all pre-shared secrets between Alice and Bob, authentication cannot be achieved.
  In addition to that, the impersonation may succeed if all of Bobs secrets are compromised, that is if Eve knows Bob's long-term and ephemeral secrets as well as the pre-shared secret between Alice and Bob. While this is no trivial attack in the sense that it cannot be prevented, it is often excluded as it describes a setting where Eve has essentially full control over Bob's system. In this case there are more direct ways than breaking cryptography to convince Bob that he is talking to Alice.
  All other attacks against entity authentication are mitigated, including *key-compromise impersonation (KCI) attacks*, where Eve tries to impersonate Alice towards Bob, while knowing all of Bob's long-term secrets. It also covers *unknown-key-share (UKS) attacks* in which Eve tricks an honest party into believing that they are communicating with someone else than they actually do.
- *Identity Hiding.* The identity of the initiator and the responder are hidden as long as both long-term secrets and the initiator's ephemeral secrets are uncompromised. Note that a compromise of a party's long-term secret is by definition also a reveal of its identity.
- *DoS Mitigation.* DoS mitigation can be achieved against adversaries that do not corrupt a pair of long-term key and pre-shared secret.

C. Formal Security Models

We formally analyze the security of the PQ-WireGuard handshake in two models.

The symbolic model. In the symbolic model, security is proven by ruling out the possibility of certain attacks, one by one. The original symbolic analysis by Donenfeld and Milner [8] covered all of the above security goals except DoS mitigation. The analysis of forward secrecy was limited to weak-PFS. We extend their model by DoS mitigation and full PFS. We detail our formalization of this model in Section IV-B.

The computational model. For the analysis of WireGuard in the computational model, Dowling and Paterson introduced the notion of eCK-PFS-PSK security [9]. It extends the eCK-PFS notion of Cremers and Feltz [49] by the treatment of pre-shared keys. The notion of eCK-PFS security in turn is a strengthening of the eCK security notion [50] that integrates

[3]Some definitions limit the meaning of ephemeral secrets to ephemeral key pairs. We use it to refer to all temporary secret data in a party's state, especially all used randomness. This turns out to be important when using KEMs.

perfect forward secrecy. In terms of the informal description above, eCK-PFS-PSK covers session-key secrecy, including PFS, and all the authentication-related security goals. What is not covered are session-key uniqueness, identity hiding, and DoS mitigation.

As Dowling and Paterson did for WireGuard, we prove security of PQ-WireGuard in the computational model with respect to eCK-PFS-PSK. The only difference is that we allow adversaries to be quantum algorithms as discussed above. For a formal description of eCK-PFS-PSK see [9, Sec. 4].

D. Cryptographic building blocks

In the following we discuss cryptographic building blocks used in (PQ-)WireGuard.

Diffie-Hellman key exchange. Strictly speaking Diffie-Hellman key exchange (DH) is not a generic cryptographic building block in the sense of the other building blocks below. Instead it is an actual scheme. However, authenticated key-exchange protocols built using the Noise framework are explicitly based on DH instead of some generic building block. This is what lead to complicated security arguments for such protocols, requiring the introduction of non-standard security assumptions like the PRFODH-assumption discussed below. Nevertheless we describe DH as it is a core ingredient of WireGuard.

We use the multiplicative notation for the group G with generator g in which the DH is carried out. To highlight similarities to the KEM-based approach, we write DH.Gen for DH key generation which returns a keypair (a, g^a). DH shared-key computation is denoted DH.Shared and outputs g^{ab} on input a secret key a and a public key g^b. WireGuard instantiates the DH key exchange with X25519 [3].

DH is vulnerable to Shor's algorithm [51], [52] and thus what makes the WireGuard handshake vulnerable to quantum attacks. Consequently, this is what we have to replace for post-quantum security. Note that from a more abstract point of view, DH supports (and is, in fact, the most common example of) non-interactive key exchange (NIKE) [53].

The way that the Diffie-Hellman key exchange is used in WireGuard seems to prevent a security proof that only uses the (standard) Decisional Diffie-Hellman (DDH) assumption ((g^x, g^y, g^{xy}) being indistinguishable from (g^x, g^y, g^z) for random x, y, z). Known proofs require assumptions from the family of PRFODH-assumptions. These combine the DDH-assumption with a prf assumption described below. Roughly they state that for some prf f and message m, $f(g^{xy}, m)$ is indistinguishable from a random value even if the adversary has (limited) oracle access to $f(a^x, b)$ and $f(a^y, b)$, where it is allowed to choose a and b. Different versions of the PRFODH-assumption are distinguished by the limitations on the oracle-access. For more details on the PRFODH family and its use in the context of WireGuard we refer to [54] and [9] respectively.

Key-encapsulation mechanisms. A key-encapsulation mechanism (KEM) is a triple of algorithms (KEM.Gen, KEM.Enc, KEM.Dec). The probabilistic key-generation KEM.Gen generates a keypair (sk, pk). Encapsulation KEM.Enc is a probabilistic algorithm which takes as input a public key pk and computes a ciphertext c and a shared key k. We make the probabilistic behavior explicit, treating KEM.Enc as deterministic algorithm which takes as additional input random coins r. This is necessary to deal with situations where the local randomness source is compromised. The decapsulation algorithm KEM.Dec takes as input a ciphertext c and a secret key sk and returns a shared key k or a failure symbol $\perp$. A KEM is $(1-\delta)$-correct if $\mathbb{E}\left[k = k' \mid (sk, pk) \leftarrow \mathsf{KEM.Gen}(),\ (c, k) \leftarrow \mathsf{KEM.Enc}(pk, r),\ k' \leftarrow \mathsf{KEM.Dec}(c, sk)\right] = 1 - \delta$, where the expectation is taken over the internal coins of KEM.Gen and KEM.Enc. We call δ the failure probability.

The security notions we need from a KEM in this paper are indistinguishable ciphertexts under chosen-plaintext attacks (IND-CPA) and under adaptive chosen-ciphertext attacks (IND-CCA). For the formal definitions of these notions in the context of KEMs, see e.g., the seminal work by Dent [55]. Intuitively, an IND-CPA-secure KEM allows two parties to agree on a shared key k without any *passive* attacker being able to learn any non-trivial information about that key. An IND-CCA-secure KEM then provides essentially the same notion, but this time for *active* attackers.

Like DH, a KEM can be used to establish a shared key between two parties over an untrusted channel in a confidential way. However, unlike DH, the communication scenario assumes interaction. When using DH, two parties that each know their own secret-key and their peer's public key can derive a shared secret without any further interaction. In contrast, when using KEMs, this does not work generically, since it is not generally possible to combine two keypairs to acquire a shared secret. Instead one party has to encapsulate a key using their peers public key and send the encapsulation to their peer, requiring one interaction.

In many applications, DH is also used in a KEM-like interactive setting; those are the cases where DH can easily be replaced by a KEM. However, many protocols also involve non-interactive applications of DH that are not trivial to replace and WireGuard is no exception in that regard.

Pseudorandom Functions. For the definitions of pseudorandom functions (PRF) and authenticated encryption (see below) we use the definitions given in [9] to keep the computational proof for PQ-WireGuard as close to that of WireGuard as possible. The definitions are verbatim copies and refer to pre-quantum security. For post-quantum security, one has to replace "PPT" (probabilistic polynomial time) by "QPT" (quantum polynomial time) algorithm when talking about the adversary.

Definition 1 (prf Security [9]): A pseudo-random function family is a collection of deterministic functions $\mathsf{PRF} = \{\mathsf{PRF}_\lambda : \mathcal{K} \times \mathcal{M} \to \mathcal{O} : \lambda \in \mathbb{N}\}$, one function for each value of λ. Here, $\mathcal{K}$, $\mathcal{M}$, $\mathcal{O}$ all depend on λ, but we suppress this for ease of notation. Given a key k in the keyspace $\mathcal{K}$ and a bit string $m \in \mathcal{M}$, PRF_λ outputs a value y in the output space $\mathcal{O} = \{0,1\}^\lambda$. We define the security of a pseudo-random function family in the following game between a challenger

$\mathcal{C}$ and an adversary $\mathcal{A}$, with λ as an implicit input to both algorithms:

1) $\mathcal{C}$ samples a key $k \xleftarrow{\$} \mathcal{K}$ and a bit b uniformly at random.
2) $\mathcal{A}$ can now query $\mathcal{C}$ with polynomially-many distinct m_i values, and receives either the output $y_i \leftarrow \mathsf{PRF}_\lambda(k, m_i)$ (when $b = 0$) or $y_i \xleftarrow{\$} \{0,1\}^\lambda$ (when $b = 1$).
3) $\mathcal{A}$ terminates and outputs a bit b'.

We say that $\mathcal{A}$ wins the PRF security game if $b' = b$ and define the advantage of an algorithm $\mathcal{A}$ in breaking the *pseudo-random function security* of a PRF family PRF as $\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{A}}(\lambda) = |2 \cdot \Pr(b' = b) - 1|$. We say that PRF is secure if for all PPT algorithms $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{A}}(\lambda)$ is negligible in the security parameter λ.

Traditionally most authors require that m can have (almost) arbitrary length. We consider a setting where m and k both have the same fixed length. In case a function f becomes a PRF when its arguments are swapped (that is $f(m,k)$ satisfies the prf-assumption), we say that f satisfies the $\mathsf{prf}^{\mathsf{swap}}$-assumption introduced in [56].

A function that satisfies the prf- and the $\mathsf{prf}^{\mathsf{swap}}$-assumption is called dual-PRF and satisfies the dual-prf-assumption. Intuitively this means that if at least one input is random and unknown to the adversary, the resulting bit string is still indistinguishable from a random value.

In PQ-WireGuard a dual-PRF appears in the form of a *key-derivation function* $\mathsf{KDF}(X,Y) = Z$ that takes two inputs, X and Y, and outputs a bit string Z consisting of three blocks $Z = Z_1 \| Z_2 \| Z_3$. We write $\mathsf{KDF}_i(X,Y)$ for the i-th block of output of $\mathsf{KDF}(X,Y)$, i.e., Z_i. The reason why KDF has to be a dual-PRF is discussed in Section IV-A.

Authenticated Encryption with Associated Data. The analysis of the PQ-WireGuard (and WireGuard) handshake only makes use of the authentication security of the used AEAD scheme. Hence, we limit our presentation to this. A discussion of secrecy related properties can e.g. be found in [57]. As discussed above, the following is a definition of pre-quantum security, the definition of post-quantum security is obtained replacing PPT by QPT in the text below.

*Definition 2 (*aead-auth *Security [9]):* An AEAD scheme AEAD is a tuple of algorithms $\mathsf{AEAD} = \{\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}\}$ associated with spaces for keys $\mathcal{K}$, nonces $\mathcal{N} \in \{0,1\}^l$, messages $\mathcal{M} \in \{0,1\}^*$ and headers $\mathcal{H} \in \{0,1\}^*$. These sets all depend on the security parameter λ. We denote by $\mathsf{AEAD.KeyGen}(\lambda) \to k$ a key generation algorithm that takes as input λ and outputs a key $k \in \mathcal{K}$. We denote by $\mathsf{AEAD.Enc}(k,N,H,M)$ the AEAD encryption algorithm that takes as input a key $k \in \mathcal{K}$, a nonce $N \in \mathcal{N}$, a header $H \in \mathcal{H}$ and a message $M \in \mathcal{M}$ and outputs a ciphertext $C \in \{0,1\}^*$. We denote by $\mathsf{AEAD.Dec}(k,N,H,C)$ the AEAD decryption algorithm that takes as input a key $k \in \mathcal{K}$, a nonce $N \in \mathcal{N}$, a header $H \in \mathcal{H}$ and a ciphertext C and returns a string M', which is either in the message space $\mathcal{M}$ or a distinguished failure symbol $\perp$. Correctness of an AEAD scheme requires that $\mathsf{AEAD.Dec}(k,N,H,\mathsf{AEAD.Enc}(k,N,H,M)) = M$ for all k,N,H,M in the appropriate spaces.

Let AEAD be an AEAD scheme, and $\mathcal{A}$ a PPT algorithm with input λ and access to an oracle $\mathsf{Enc}(\cdot,\cdot,\cdot)$. This oracle, given input (N,H,M), outputs $\mathsf{Enc}(k,N,H,M)$ for a randomly selected key $k \in \mathcal{K}$. We say that $\mathcal{A}$ *forges* a ciphertext if $\mathcal{A}$ outputs (N,H,C) such that $\mathsf{Dec}(k,N,H,C) \to M \neq \perp$ and (N,H,M) was not queried to the oracle. We define the advantage of a PPT algorithm $\mathcal{A}$ in forging a ciphertext as $\mathsf{Adv}^{\mathsf{aead\text{-}auth}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)$. We say that an AEAD scheme AEAD is aead-auth-secure if for all PPT algorithms $\mathcal{A}$ $\mathsf{Adv}^{\mathsf{aead\text{-}auth}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)$ is negligible in the security parameter λ.

(PQ-)WireGuard assumes that the keys of the AEAD scheme used are random bit strings. This is the case for all practical AEAD schemes we are aware of. Also in theory this is no limitation as one can always replace the secret key by the coins of AEAD.Gen.

E. The WireGuard handshake

We are now ready to review the handshake protocol of WireGuard. In Algorithm 1 we first give a high-level view of the handshake, largely following the description in [9]. The initiator and responder are identified by their long-term, *static* public keys $\mathtt{spk}_i$ and $\mathtt{spk}_r$ (with corresponding secret keys $\mathtt{ssk}_i$ and $\mathtt{ssk}_r$, respectively). Those key pairs are generated before the first handshake between two parties and WireGuard assumes that the public keys are exchanged in a secure way (guaranteeing at least authenticity) before the first handshake.

Algorithm 1 High-level view on the WireGuard handshake

Initiator		**Responder**
1: $(\mathtt{esk}_i, \mathtt{epk}_i) \leftarrow \mathsf{DH.Gen}()$		
2: $\mathtt{sid}_i \xleftarrow{\$} \{0,1\}^{32}$		
3: $\mathtt{ltk} \leftarrow \mathsf{AEAD.Enc}(\kappa_3, 0, \mathtt{spk}_i, H_3)$		
4: $now \leftarrow \mathsf{Timestamp}()$		
5: $\mathtt{time} \leftarrow \mathsf{AEAD.Enc}(\kappa_4, 0, H_4, now)$		
6: $\mathtt{m1} \leftarrow \mathsf{MAC}(\mathsf{H}(\mathtt{lbl}_3 \| \mathtt{spk}_r), \mathtt{type} \| 0^3 \| \mathtt{sid}_i \| \mathtt{epk}_i \| \mathtt{ltk} \| \mathtt{time})$		
7: $\mathtt{m2} \leftarrow \mathsf{MAC}(cookie, \mathtt{type} \| 0^3 \| \mathtt{sid}_i \| \mathtt{epk}_i \| \mathtt{ltk} \| \mathtt{time} \| \mathtt{m1})$		
8: $\mathtt{InitHello} \leftarrow \mathtt{type} \| 0^3 \| \mathtt{sid}_i \| \mathtt{epk}_i \| \mathtt{ltk} \| \mathtt{time} \| \mathtt{m1} \| \mathtt{m2}$		
	$\xrightarrow{\mathtt{InitHello}}$	
		9: $(\mathtt{esk}_r, \mathtt{epk}_r) \leftarrow \mathsf{DH.Gen}()$
		10: $\mathtt{sid}_r \xleftarrow{\$} \{0,1\}^{32}$
		11: $\mathtt{zero} \leftarrow \mathsf{AEAD.Enc}(\kappa_9, 0, H_9, \emptyset)$
		12: $\mathtt{m1} \leftarrow \mathsf{MAC}(\mathsf{H}(\mathtt{lbl}_3 \| \mathtt{spk}_i), \mathtt{type} \| 0^3 \| \mathtt{sid}_r \| \mathtt{sid}_i \| \mathtt{epk}_r \| \mathtt{zero})$
		13: $\mathtt{m2} \leftarrow \mathsf{MAC}(cookie, \mathtt{type} \| 0^3 \| \mathtt{sid}_r \| \mathtt{sid}_i \| \mathtt{epk}_r \| \mathtt{zero} \| \mathtt{m1})$
		14: $\mathtt{RespHello} \leftarrow \mathtt{type} \| 0^3 \| \mathtt{sid}_r \| \mathtt{sid}_i \| \mathtt{epk}_r \| \mathtt{zero} \| \mathtt{m1} \| \mathtt{m2}$
	$\xleftarrow{\mathtt{RespHello}}$	
15: $tk_i \leftarrow \mathsf{KDF}_1(C_9, \emptyset)$		
16: $tk_r \leftarrow \mathsf{KDF}_2(C_9, \emptyset)$		
	$\xrightarrow{\mathsf{AEAD.Enc}(tk_i, \cdot, \emptyset, \text{application data})}$	

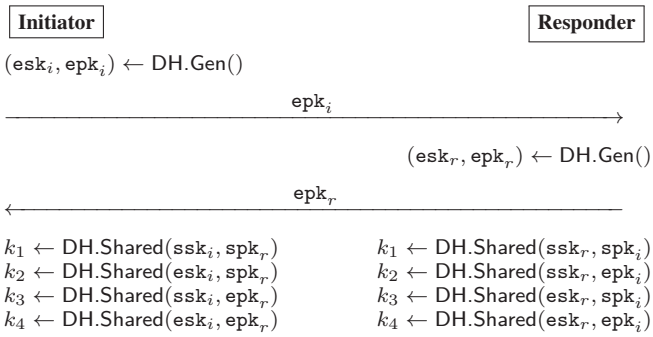
From a cryptographic point of view, and in particular for the context of this paper, what is most interesting is how the values H_k, κ_k, and C_k in Algorithm 1 are computed. This is laid out in Table I, again largely following the description in [9]. The values $\mathtt{lbl}_1$, $\mathtt{lbl}_2$, and $\mathtt{lbl}_3$ are fixed strings (see [1, Sec. V.D]). The value *cookie* is most of the time just 16 zero bytes, except when the server is under load and

is sending out so-called "cookie replies" as denial-of-service countermeasure; for details, see [1, Sec. V.D7]. Note that Algorithm 1 includes the first application-data packet from the initiator. The reason is that this packet also serves as key confirmation of the handhshake. This dual purpose of the first data packet is the reason that WireGuard cannot be proven secure in a modular way (separating handshake from data transport) without modification. For details see [9].

III. From WireGuard to PQ-WireGuard

As outlined in Sections I and II, the WireGuard handshake is heavily based on DH, which does not have an efficient and well established post-quantum equivalent. Hence, in this section we describe how we replace DH by KEMs, for which well-established, efficient post-quantum instantiations exist. We start by considering a simplified view on the core of the DH-based WireGuard handshake.

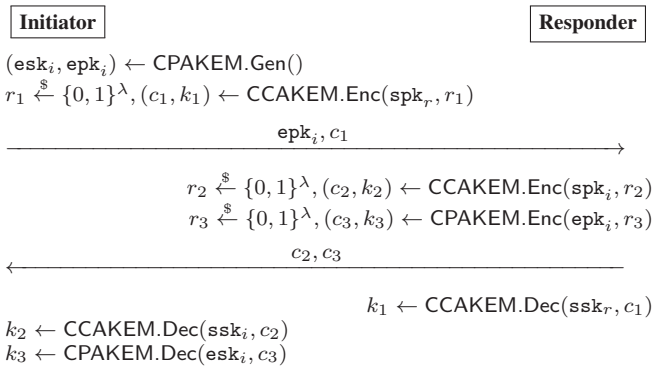
In this simplified view, the initiator has a long-term static DH key pair $(\mathtt{ssk}_i, \mathtt{spk}_i)$ and the responder has a long-term static DH key pair $(\mathtt{ssk}_r, \mathtt{spk}_r)$. The handshake proceeds as follows:



The final session key is computed using the keys k_1, k_2, k_3, and k_4.

A. Moving from DH to KEMs

In [18], Fujioka, Suzuki, Xagawa, and Yoneyama describe an approach to authenticated key exchange using only KEMs; we largely follow their approach in our design. Towards our final proposal, let us first try to straight-forwardly translate the DH-based approach to a KEM-based approach. The problem is, as described in Subsection II-D, that we cannot perform a non-interactive key exchange, i.e., we cannot build an equivalent to the static-static DH computation of k_1. Let us for the moment ignore the static-static DH and start by translating the remainder of the handshake to a KEM-based handshake. For this, we will use an IND-CCA-secure KEM $\mathsf{CCAKEM} = (\mathsf{CCAKEM.Gen}, \mathsf{CCAKEM.Enc}, \mathsf{CCAKEM.Dec})$ and an IND-CPA-secure KEM $\mathsf{CPAKEM} = (\mathsf{CPAKEM.Gen}, \mathsf{CPAKEM.Enc}, \mathsf{CPAKEM.Dec})$. The initiator has a long-term static CCAKEM key pair $(\mathtt{ssk}_i, \mathtt{spk}_i)$ and the responder has a long-term static CCAKEM key pair $(\mathtt{ssk}_r, \mathtt{spk}_r)$. Now, the handshake proceeds as follows:



The role of static-static DH. This naive approach already has lots of the security properties of the WireGuard handshake, but it is lacking three properties that are achieved through the inclusion of the static-static DH.

1) *Security under MEX attacks.* One corruption pattern in MEX attacks reveals all ephemeral secrets to the adversary, including the used randomness. The motivation for this pattern is a situation in which the protocol is executed on a device with a subverted, or simply broken RNG – in this case security can only be derived from the long-term secrets that have (ideally) been generated in a secure environment. In the above naive approach we do not obtain any security in this scenario. The reason is that the randomness used by CCAKEM.Enc is corrupted and consequently, an adversary can recompute the shared secret simply running CCAKEM.Enc.
   The general approach to address this issue is to securely combine ephemeral randomness r with some long-term secret σ before using it as protocol input. In [58] this is done using $\mathsf{PRF}(r, \sigma) \oplus \mathsf{PRF}(\sigma', r')$ for two independent ephemeral values r and r' and two independent long-term secret values σ and σ', where $\oplus$ denotes exclusive or. This "twisted PRF" trick ensures that nothing beyond PRF security is required to prove this approach secure in the standard model. In the case of WireGuard we will see that we require a dual-prf assumption on KDF_1 anyway, so we can use this assumption here as well and simplify the construction to $\mathsf{KDF}_1(\sigma, r)$.
2) *Resistance to unknown-keyshare attacks.* The static-static DH is also the only line of defense in WireGuard against unknown-keyshare attacks. This is because the IDs (or public keys) of the two parties are not hashed into the final session key. As briefly discussed in Section I, WireGuard has the option to hash a pre-shared key psk into the final session key; by default psk is set to the all-zero string. In PQ-WireGuard we instead set psk to $\mathsf{H}(\mathtt{spk}_i \oplus \mathtt{spk}_r)$. This ensures that session keys are linked to the static public keys of the communicating parties and thus prevents unknown-keyshare attacks.
3) *Authenticated initiation.* Finally, the static-static DH ensures that the first message from the initiator is already authenticated. This allows the server to detect illegitimate messages at this very early stage and consequently abort the handshake. This is not a security property in

TABLE I

COMPUTATION OF SEED VALUES, KEYS, AND HASHES THROUGH THE WIREGUARD HANDSHAKE. FOR VALUES OF k WITH TWO ROWS, THE FIRST ROW DENOTES COMPUTATION ON THE INITIATOR SIDE AND THE SECOND ROW THE CORRESPONDING COMPUTATION ON THE RESPONDER SIDE.

k	seed C_k	key κ_k	hash H_k
1	$\mathsf{H}(\mathtt{lbl}_1)$	—	$\mathsf{H}(C_1 \parallel \mathtt{lbl}_2)$
2	$\mathsf{KDF}_1(C_1, \mathtt{epk}_i)$	—	$\mathsf{H}(H_1 \parallel \mathtt{spk}_r)$
3	$\mathsf{KDF}_1(C_2, \mathsf{DH.Shared}(\mathtt{esk}_i, \mathtt{spk}_r))$	$\mathsf{KDF}_2(C_2, \mathsf{DH.Shared}(\mathtt{esk}_i, \mathtt{spk}_r))$	$\mathsf{H}(H_2 \parallel \mathtt{epk}_i)$
	$\mathsf{KDF}_1(C_2, \mathsf{DH.Shared}(\mathtt{ssk}_r, \mathtt{epk}_i))$	$\mathsf{KDF}_2(C_2, \mathsf{DH.Shared}(\mathtt{ssk}_r, \mathtt{epk}_i))$	$\mathsf{H}(H_2 \parallel \mathtt{epk}_i)$
4	$\mathsf{KDF}_1(C_3, \mathsf{DH.Shared}(\mathtt{ssk}_i, \mathtt{spk}_r))$	$\mathsf{KDF}_2(C_2, \mathsf{DH.Shared}(\mathtt{ssk}_i, \mathtt{spk}_r))$	$\mathsf{H}(H_3 \parallel \mathtt{ltk})$
	$\mathsf{KDF}_1(C_3, \mathsf{DH.Shared}(\mathtt{ssk}_r, \mathtt{spk}_i))$	$\mathsf{KDF}_2(C_2, \mathsf{DH.Shared}(\mathtt{ssk}_r, \mathtt{spk}_i))$	$\mathsf{H}(H_3 \parallel \mathtt{ltk})$
5	—	—	$\mathsf{H}(H_4 \parallel \mathtt{time})$
6	$\mathsf{KDF}_1(C_4, \mathtt{epk}_r)$	—	$\mathsf{H}(H_5 \parallel \mathtt{epk}_r)$
7	$\mathsf{KDF}_1(C_6, \mathsf{DH.Shared}(\mathtt{esk}_i, \mathtt{epk}_r))$	—	—
	$\mathsf{KDF}_1(C_6, \mathsf{DH.Shared}(\mathtt{esk}_r, \mathtt{epk}_i))$	—	—
8	$\mathsf{KDF}_1(C_7, \mathsf{DH.Shared}(\mathtt{ssk}_i, \mathtt{epk}_r))$	—	—
	$\mathsf{KDF}_1(C_7, \mathsf{DH.Shared}(\mathtt{esk}_r, \mathtt{spk}_i))$	—	—
9	$\mathsf{KDF}_1(C_8, psk)$	$\mathsf{KDF}_3(C_8, psk)$	$\mathsf{H}(H_6 \parallel \mathsf{KDF}_2(C_8, psk))$
10	—	—	$\mathsf{H}(H_9 \parallel \mathtt{zero})$

the cryptographic sense, but helps mitigate easy DoS attacks. If we follow the argumentation of [12] stating that static public keys of WireGuard users are typically not public and hence not known to attackers, the same level of DoS protection is achieved by the default value of $psk = \mathsf{H}(\mathtt{spk}_i \oplus \mathtt{spk}_r)$. Users who do not want to rely on this assumption need to set psk to a secret shared key that is agreed on out-of-band to achieve the same level of DoS protection as in WireGuard.

Adding key confirmation. As mentioned above, WireGuard uses the first application-data packet from the initiator for implicit key confirmation, which makes it impossible to prove the WireGuard handshake secure in the eCK-PFS-PSK model. The proof in [9], just like our computational proof, requires an explicit and separate `InitConf` key-confirmation message from the initiator at the end of the handshake. In PQ-WireGuard we add this explicit key-confirmation.

Putting it together. Our full proposal for the KEM-based PQ-WireGuard handshake is given in Algorithm 2 and Table II. Aside from translating all DH key exchanges, except for the static-static one, to corresponding KEM operations, we introduce the following changes to the WireGuard handshake:

- We use calls to $\mathsf{KDF}_1(\sigma_i, r_i)$ and $\mathsf{KDF}_1(\sigma_r, r_r)$ in steps 4 and 13 of Alg. 2 to securely mix ephemeral randomness with long-term randomness. This is precisely the countermeasure against MEX attacks discussed above.
- We use $\mathsf{H}(\mathtt{spk}_i \oplus \mathtt{spk}_r)$ as default value for psk.
- Instead of feeding $\mathtt{spk}_i$ into AEAD.Enc in step 5, we use $\mathsf{H}(\mathtt{spk}_i)$. This is essentially the same trick proposed in [12], except that we need it for a very different reason. In [12] the reason is to add some protection against future quantum attackers who are recording handshakes today. For us the reason is simply a size reduction from the potentially large public key of CCAKEM to a 32-byte hash of this public key.
- We add the explicit key-confirmation message `InitConf`.

Algorithm 2 High-level view on our PQ-WireGuard handshake. Highlighted in blue are differences to Alg. 1.

Initiator		**Responder**
1: $(\mathtt{esk}_i, \mathtt{epk}_i) \leftarrow \mathsf{CPAKEM.Gen}()$		
2: $\mathtt{sid}_i \xleftarrow{\$} \{0,1\}^{32}$		
3: $r_i \leftarrow \{0,1\}^{\lambda}$		
4: $(\mathtt{ct1}, \mathtt{shk1}) \leftarrow \mathsf{CCAKEM.Enc}(\mathtt{spk}_r, \mathsf{KDF}_1(\sigma_i, r_i))$		
5: $\mathtt{ltk} \leftarrow \mathsf{AEAD.Enc}(\kappa_3, 0, \mathsf{H}(\mathtt{spk}_i), H_3)$		
6: $now \leftarrow \mathsf{Timestamp}()$		
7: $\mathtt{time} \leftarrow \mathsf{AEAD.Enc}(\kappa_4, 0, H_4, now)$		
8: $\mathtt{m1} \leftarrow \mathsf{MAC}(\mathsf{H}(\mathtt{lbl}_3 \Vert \mathtt{spk}_r), \mathtt{type} \Vert 0^3 \Vert \mathtt{sid}_i \Vert \mathtt{epk}_i \Vert \mathtt{ct1} \Vert \mathtt{ltk} \Vert \mathtt{time})$		
9: $\mathtt{m2} \leftarrow \mathsf{MAC}(cookie, \mathtt{type} \Vert 0^3 \Vert \mathtt{sid}_i \Vert \mathtt{epk}_i \Vert \mathtt{ct1} \Vert \mathtt{ltk} \Vert \mathtt{time} \Vert \mathtt{m1})$		
10: $\mathtt{InitHello} \leftarrow \mathtt{type} \Vert 0^3 \Vert \mathtt{sid}_i \Vert \mathtt{epk}_i \Vert \mathtt{ct1} \Vert \mathtt{ltk} \Vert \mathtt{time} \Vert \mathtt{m1} \Vert \mathtt{m2}$		
	$\xrightarrow{\qquad \mathtt{InitHello} \qquad}$	
		11: $e, r_r \leftarrow \{0,1\}^{\lambda} \times \{0,1\}^{\lambda}$
		12: $(\mathtt{ct2}, \mathtt{shk2}) \leftarrow \mathsf{CPAKEM.Enc}(\mathtt{epk}_i, e)$
		13: $(\mathtt{ct3}, \mathtt{shk3}) \leftarrow \mathsf{CCAKEM.Enc}(\mathtt{spk}_i, \mathsf{KDF}_1(\sigma_r, r_r))$
		14: $\mathtt{sid}_r \xleftarrow{\$} \{0,1\}^{32}$
		15: $\mathtt{zero} \leftarrow \mathsf{AEAD.Enc}(\kappa_9, 0, H_9, \emptyset)$
		16: $\mathtt{m1} \leftarrow \mathsf{MAC}(\mathsf{H}(\mathtt{lbl}_3 \Vert \mathtt{spk}_i), \mathtt{type} \Vert 0^3 \Vert \mathtt{sid}_r \Vert \mathtt{sid}_i \Vert \mathtt{ct2} \Vert \mathtt{ct3} \Vert \mathtt{zero})$
		17: $\mathtt{m2} \leftarrow \mathsf{MAC}(cookie, \mathtt{type} \Vert 0^3 \Vert \mathtt{sid}_r \Vert \mathtt{sid}_i \Vert \mathtt{ct2} \Vert \mathtt{ct3} \Vert \mathtt{zero} \Vert \mathtt{m1})$
		18: $\mathtt{RespHello} \leftarrow \mathtt{type} \Vert 0^3 \Vert \mathtt{sid}_r \Vert \mathtt{sid}_i \Vert \mathtt{ct2} \Vert \mathtt{ct3} \Vert \mathtt{zero} \Vert \mathtt{m1} \Vert \mathtt{m2}$
	$\xleftarrow{\qquad \mathtt{RespHello} \qquad}$	
19: $\mathtt{conf} \leftarrow \mathsf{AEAD.Enc}(\kappa_{10}, 0, H_{10}, \emptyset)$		
20: $\mathtt{m1} \leftarrow \mathsf{MAC}(\mathsf{H}(\mathtt{lbl}_3 \Vert \mathtt{spk}_r), \mathtt{type} \Vert 0^3 \Vert \mathtt{sid}_i \Vert \mathtt{sid}_r \Vert \mathtt{conf})$		
21: $\mathtt{m2} \leftarrow \mathsf{MAC}(cookie, \mathtt{type} \Vert 0^3 \Vert \mathtt{sid}_i \Vert \mathtt{sid}_r \Vert \mathtt{conf} \Vert \mathtt{m1})$		
22: $\mathtt{InitConf} \leftarrow \mathtt{type} \Vert 0^3 \Vert \mathtt{sid}_i \Vert \mathtt{sid}_r \Vert \mathtt{conf} \Vert \mathtt{m1} \Vert \mathtt{m2}$		
	$\xrightarrow{\qquad \mathtt{InitConf} \qquad}$	
		23: $tk_i \leftarrow \mathsf{KDF}_1(C_{10}, \emptyset)$
		24: $tk_r \leftarrow \mathsf{KDF}_2(C_{10}, \emptyset)$

IV. SECURITY ANALYSIS

We provide two proofs of security for PQ-WireGuard: one in the computational and one in the symbolic model. Thereby we provide the same level of security guarantees as for WireGuard. Below we outline both proofs. In the computational model we prove that the PQ-WireGuard hand-

TABLE II
COMPUTATION OF SEED VALUES, KEYS, AND HASHES THROUGH THE PQ-WIREGUARD HANDSHAKE. FOR VALUES OF k WITH TWO ROWS, THE FIRST ROW DENOTES COMPUTATION ON THE INITIATOR SIDE AND THE SECOND ROW THE CORRESPONDING COMPUTATION ON THE RESPONDER SIDE. HIGHLIGHTED IN BLUE ARE DIFFERENCES TO TABLE I.

k	seed C_k	key κ_k	hash H_k
1	$\mathsf{H}(\mathtt{lbl}_1)$	—	$\mathsf{H}(C_1 \parallel \mathtt{lbl}_2)$
2	$\mathsf{KDF}_1(C_1, \mathtt{epk}_i)$	—	$\mathsf{H}(H_1 \parallel \mathtt{spk}_r)$
3	$\mathsf{KDF}_1(C_2, \mathtt{shk1})$	$\mathsf{KDF}_2(C_2, \mathtt{shk1})$	$\mathsf{H}(H_2 \parallel \mathtt{epk}_i)$
	$\mathsf{KDF}_1(C_2, \mathsf{CCAKEM.Dec}(\mathtt{ssk}_r, \mathtt{ct1}))$	$\mathsf{KDF}_2(C_2, \mathsf{CCAKEM.Dec}(\mathtt{ssk}_r, \mathtt{ct1}))$	$\mathsf{H}(H_2 \parallel \mathtt{epk}_i)$
4	$\mathsf{KDF}_1(C_3, psk)$	$\mathsf{KDF}_2(C_2, psk)$	$\mathsf{H}(H_3 \parallel \mathtt{ltk})$
5	—	—	$\mathsf{H}(H_4 \parallel \mathtt{time})$
6	$\mathsf{KDF}_1(C_4, \mathtt{ct2})$	—	$\mathsf{H}(H_5 \parallel \mathtt{ct2})$
7	$\mathsf{KDF}_1(C_6, \mathsf{CPAKEM.Dec}(\mathtt{esk}_i, \mathtt{ct2}))$	—	—
	$\mathsf{KDF}_1(C_6, \mathtt{shk2})$	—	—
8	$\mathsf{KDF}_1(C_7, \mathsf{CCAKEM.Dec}(\mathtt{ssk}_i, \mathtt{ct3}))$	—	—
	$\mathsf{KDF}_1(C_7, \mathtt{shk3})$	—	—
9	$\mathsf{KDF}_1(C_8, psk)$	$\mathsf{KDF}_3(C_8, psk)$	$\mathsf{H}(H_6 \parallel \mathsf{KDF}_2(C_8, psk))$
10	$\mathsf{KDF}_1(C_9, \emptyset)$	$\mathsf{KDF}_2(C_9, \emptyset)$	$\mathsf{H}(H_9 \parallel \mathtt{zero})$

shake, like the WireGuard handshake (with added key confirmation), achieves eCK-PFS-PSK-security. While certainly on the stronger end of security notions for authenticated key-exchange, eCK-PFS-PSK only proves session-key secrecy and authenticity properties. Further notions that PQ-WireGuard also targets, like identity hiding and DoS-mitigation, are not covered by it. These additional notions are covered by the symbolic proof. The symbolic proof not only covers additional security properties but also has the advantage of being computer-verified. However, this comes at the cost of being done in the symbolic model which treats all building blocks as ideal and consequently can only provide a heuristic argument.

A. The Computational Proof

To prove that the PQ-WireGuard handshake achieves eCK-PFS-PSK-security, we adapt the computational proof for WireGuard [9] by Dowling and Paterson (who kindly provided us with their LaTeX-sources) to PQ-WireGuard. The core of this adaptation is to replace proof steps (i.e., game-hops) making use of either the PRFODH- or the DDH-assumptions by generic KEM-security- and prf-assumptions. Most of these changes are straightforward and readers who are familiar with the original proof should find the result familiar.

On a high level both proofs consist of the same case-distinction between whether the adversary tries to impersonate a party or learn information about the established key and the ways in which the adversary is allowed to corrupt parties. For each case the proof uses a sequence of games to show that the adversary has to either directly break the authenticity of the AEAD-scheme for a successful impersonation attack or distinguish two information-theoretically indistinguishable bit strings to learn any non-trivial information about the key.

The majority of game hops are ones where the prf or the $\mathsf{prf}^{\mathsf{swap}}$ assumptions are used. In these game-hops the output of KDF, used to combine two intermediate values, at least one of which is random (which one depends on the adversarial corruption), gets replaced by a random value. These "symmetric game hops" are essentially the same in the WireGuard and the PQ-WireGuard proof.

The other major category of game hops are those where the output of some asymmetric primitive is replaced by a random value. For WireGuard, these are the cases where two DH shares get combined and hashed afterwards. In this case, different versions of the PRFODH assumption are used to argue indistinguishability of the games before and after the hop. For PQ-WireGuard, these steps use KEM encapsulations and decapsulations. In these cases, indistinguishability can be argued using the IND-CPA security of CPAKEM and the IND-CCA security of CCAKEM.

The differences between the proofs for WireGuard and PQ-WireGuard are not just limited to these asymmetric game hops: The ways values are combined in some cases in PQ-WireGuard differ substantially from WireGuard. This is necessary to deal with the more limited abilities of KEMs when compared to Diffie-Hellman. As a consequence we had to add multiple new symmetric game hops, particularly around most asymmetric game hops.

In addition to that we noticed one minor mistake in the WireGuard proof that also directly affects our proof. The WireGuard proof claims that it is sufficient for KDF to be a prf. This turns out to be too weak. KDF is used to combine two inputs. While in different corruption settings there is always one input that is indistinguishable from random for the attacker, but it is not always the same input. Consequently, the function actually has to be a dual-PRF (which can be keyed on either input). For the most part this occurs in asymmetric game hops where the prf-assumption is "hidden" in the PRFODH assumption but it also occurs in one symmetric hop. We notified the authors of the WireGuard proof who acknowledged the issue.

Given these changes, we are able to show that there is no efficient adversary against the eCK-PFS-PSK security of PQ-WireGuard under the assumptions that the used KDF is a secure dual-PRF, that the used KEMs are respectively IND-CCA and IND-CPA secure, and that the used AEAD scheme is secure in terms of authenticity. More specifically, we show that for every (possibly quantum) adversary $\mathcal{A}$ we can

construct a set $\mathcal{R}$ of (possibly quantum) reduction algorithms $\mathcal{R}_i$ that use oracle access to $\mathcal{A}$ to break one of the security assumptions running in about the same time as $\mathcal{A}$ so that:

$$\mathsf{Adv}^{\text{eCK-PFS-PSK}}_{\text{pqWG},\text{clean}_{\text{eCK-PFS-PSK}},n_P,n_S,\mathcal{A}}(\lambda)$$
$$\leq n_P^2 n_S \left(\begin{array}{rrrl} & (7n_S+9) & \cdot & \mathsf{Adv}^{\text{prf}}_{\text{KDF},\mathcal{R}}(\lambda) \\ + & (2n_S+4) & \cdot & \mathsf{Adv}^{\text{prf}^{\text{swap}}}_{\text{KDF},\mathcal{R}}(\lambda) \\ + & (n_S+2) & \cdot & \mathsf{Adv}^{\text{IND-CCA}}_{\text{CCAKEM},\mathcal{R}}(\lambda) \\ + & n_S & \cdot & \mathsf{Adv}^{\text{IND-CPA}}_{\text{CPAKEM},\mathcal{R}}(\lambda) \\ + & 2 & \cdot & \mathsf{Adv}^{\text{auth-aead}}_{\text{AEAD},\mathcal{R}}(\lambda) \\ + & (n_S+2) & \cdot & \frac{n_s}{2^\lambda} \end{array} \right)$$

where n_P is the number of parties and n_S is the number of sessions. Here we use $\mathsf{Adv}^{\text{prop}}_{\text{F},\mathcal{R}}(\lambda)$ for the maximum success probability over all $\mathcal{R}_i \in \mathcal{R}$ against property prop of building block F. Our security proof, including a slightly tighter and more precise bound is available in Appendix B.

Finally we would like to point out a pleasant side-result of the strong security-notion and the use of two different KEMs that correspond to static and ephemeral keys: If we model the break of a KEM as the reveal of all secret keys (and therefore also encapsulated secrets) then a break of either KEM does not break the confidentiality of PQ-WireGuard as long as there is no further corruption:

A break of our CCA-KEM would be equivalent to a corruption of all static secrets, but notably not the ephemeral keys used with the CPA-KEM. As long as no ephemeral secrets are compromised, eCK-PFS-PSK-security still promises that the established key remains confidential. (However, authenticity is trivially broken.)

A break of our CPA-KEM on the other hand would be equivalent to a corruption of all ephemeral secrets, but not of the static secrets that are used with the CCA-KEM. As long as no static secrets are compromised eCK-PFS-PSK-security still promises authenticity and confidentiality, losing PFS though.

The consequence of this is an increased robustness of the scheme which is relevant to us as most post-quantum primitives (in case of our proposed instantiation particularly Dagger) are rather new and therefore more likely to break than more traditional schemes. The practical impact is that PQ-WireGuard already provides some of the properties of hybrid protocols that aim at redundance of security assumptions by combining cryptographic schemes that use different security assumptions (e.g., ECC and lattice-based schemes).

B. The Symbolic Proof

The symbolic proof of PQ-WireGuard uses the Tamarin prover [43], building on the symbolic proof for WireGuard [8]. Tamarin is a formal verification tool for cryptographic protocols. It supports stateful protocols, falsification, and unbounded verification. Those features as well as its built-in support of Diffie-Hellman exponentiation motivated the use of Tamarin to analyze several cryptographic protocols, including TLS 1.3 [59] and the 5G protocol [60]. An extensive description of Tamarin can be found in [61].

Our Symbolic Model. The symbolic model of PQ-WireGuard is based on the Tamarin model of WireGuard [8] but extends it. The Tamarin model of WireGuard does not cover replay resistance and DoS mitigation, both claimed by WireGuard. We add proofs of these properties. Furthermore, the WireGuard model did not allow an adversary to compromise the random-number generator of an honest party, which is allowed in our extended model, e.g. when corrupting the ephemeral state of a party. Moreover, the WireGuard model only covered weak-PFS while our extended model covers full PFS. In the proof of identity hiding, we observe an issue with the applicability of the symbolic model. In the case of KEMs, it turns out that the assumptions implicitly made by the symbolic model for the the proof of identity hiding are not implied by the standard security assumptions. In summary, our Tamarin model shows that the security properties of WireGuard are also satisfied by PQ-WireGuard, although with an additional requirement on the CCAKEM for the case of identity hiding.

We modified the original model to reflect PQ-WireGuard and extended the ability of the adversary. In particular, we analyze the PQ-WireGuard protocol for an unbounded number of concurrent handshakes under MEX attacks.

The DH-based key exchange of WireGuard was modeled as

```
rule Handshake_Init:
  let pkI = 'g'^~ltkI
    pekI  = 'g'^~ekI
    eisr  = pkR^~ekI
    cii   = h('noise')
    hii   = h(<cii, 'id', pkR, pekI>)
    ci0   = h(<cii, pekI, '1'>)
    ci1   = h(<ci0, eisr, '1'>)
    ki1   = h(<ci0, eisr, '2'>)
    astat = aead(ki1, <pkI, ~pkISurrogate>, hii)
    hi0   = h(<hii, astat>)
    ci2   = h(<ci1, sisr, '1'>)
    ki2   = h(<ci1, sisr, '2'>)
    ats   = aead(ki2, $ts, hi0)
    hi1   = h(<hi0, ats>)
    m1    = <'1', ~sidI, pekI, astat, ats, $mac1, $mac2> in
  [ ...
```

For PQ-WireGuard this key exchange is replaced by the KEM-based construction described in Algorithm 2. We model this approach with the following rule:

```
rule Handshake_Init:
  let pkI = pk(~ltkI)
    kb    = prf(~tpk, ~r3)
    pekI  = pk(~ekI)
    cii   = h('noise')
    hii   = h(<cii, 'id', pkR, pekI>)
    ci0   = h(<cii, pekI, '1'>)
    sct   = aenc{kb}pkR
    ci1   = h(<ci0, kb, '1'>)
    ki1   = h(<ci0, kb, '2'>)
    astat = aead(ki1, <h(pkI), ~pkISurrogate>, hii)
    hi0   = h(<hii, astat>)
    ci2   = h(<ci1, ~psk, '1'>)
    ki2   = h(<ci1, ~psk, '2'>)
    ats   = aead(ki2, <$ts, 'TAI64N'>, hi0)
    hi1   = h(<hi0, ats>)
    m1    = <'1', ~sidI, sct, pekI, astat, ats, $mac1, $mac2> in
  [ ...
```

This way we modified both the model and the proofs of the existing security properties to match PQ-WireGuard. In addition, we analyzed the aforementioned missing security properties that were not included in the original model. For each of those security properties, we identify the exact conditions under which the security property holds. The results for the added

security proofs are presented in the rest of this section. The results for the remaining properties can be found in the full version. The full Tamarin proof is part of the supplementary material of this paper.

Replay Attacks. We model the replay-attack protection on the responder as a restriction that only allows a responder to accept an initiation message with a particular timestamp once.

```
restriction OnlyOnce:
  "All i r t #i #j. OnlyOnce(i, r, t) @ i
    & OnlyOnce(i, r, t) @ j ==> #i = #j"
```

This timestamp value is public, which reflects the fact that an adversary can easily infer the timestamp. A responder records the timestamp of all incoming initiation messages and will not accept an initiation message whose timestamp has been recorded.

Note that in the actual implementation, initiation messages whose timestamp has never been recorded, but is older than the timestamp in the latest accepted initiation message will also be rejected. We do not model this, however, since this case is not a replay attack.

This restriction already prevents an adversary from replaying an initiation message as a whole. We further allow an adversary to tamper with arbitrary fields in an initiation message before it is replayed. An initiation message contains the fields $(sid_i, epk_i, ct1, ltk, time)$. sid_i is purely a handshake session identifier and plays no role in the actual handshake. $ct1$ encapsulates $shk1$. The ephemeral public key epk_i is mixed together with $shk1$ to generate the symmetric keys κ_3 and κ_4 used to encrypt ltk and $time$. Therefore, the adversary must compromise $shk1$ in order to tamper the timestamp value encrypted in $time$.

With this notion in mind, the replay-attack protection seems to rely on the secrecy of $shk1$ alone, and we prove with lemma *replay_attack_resistance* that this is indeed the case.

```
lemma replay_attack_resistance:
    "All pki pkr peki peki2 psk psk2 cr cr2 kb ka ka2 k k2
        ts ts2 tpk r #i #i1 #j.
        // if R receives an init msg containing secret kb
        RKeys(<pki, pkr, peki, psk, cr, kb, ka, k>) @ i
        & OnlyOnce(pki, pkr, ts) @ i
        // and the init msg indeed comes from I
        & ISend(<pki, pkr, peki, psk, kb>) @ i1 & #i1 < #i
        & PRFGenI_static(tpk, r, kb) @ i1
        // and R receives later another init msg containing kb
        & RKeys(<pki, pkr, peki2, psk2, cr2, kb, ka2, k2>) @ j
        // but with a different timestamp
        & #i < #j & OnlyOnce(pki, pkr, ts2) @ j & not(ts = ts2)
    ==> // then the attacker crafted the second init msg
        not(Ex #j1. ISend(<pki, pkr, peki2, psk2, kb>) @j1
            & #j1 < #j & #i < #j1) & (
            // by compromising the static key of R
            (Ex #j1. Reveal_AK(pkr) @ j1 & #j1 < #j)
            // or by compromising both I's RNG and I's PRF key
            | ((Ex #j1. Reveal_rnd_I_static(r) @ j1)
              &(Ex #j1. Reveal_prfk(tpk) @ j1)))"
```

In particular, we prove that an adversary cannot trick a responder into accepting an initiation message with an encapsulated secret $shk1$ that the responder has seen before, without compromising

- the responder's static private key, or
- the initiator's random number generator and PRF secret.

DoS Mitigation. In WireGuard, the result of the static-static DH is used to authenticate the initiation message. In PQ-WireGuard, there is no static-static DH to use anymore; instead, the pre-shared key is used for this purpose. Without the pre-shared key, the authenticity of an initiation message cannot be established, and the initiation message will be accepted. With lemma *dos_mitigation* we prove that an adversary must compromise the pre-shared key in order to have its victim carry out expensive public key operations. Such a DoS attack is considered successful when a responder R accepts an initiation message and generates a response message for it, while the initiation message:

- claims to be from honest initiator I but was in fact crafted by the attacker completely
- claims to be intended for R, but was in fact generated by honest initiator I for another honest peer R'

We prove that in those two cases the responder R will not accept the initiation message unless the pre-shared key between R and the claimed initiator I has been compromised.

```
lemma dos_mitigation:
  "(All pki pkr peki psk cr kb ka k #i.
    // if R accepts an init msg (claimed to be) from I with kb
    RKeys(<pki, pkr, peki, psk, cr, kb, ka, k>) @ i
    // and no honest peer has ever sent an init msg with kb
    & not(Ex pki1 pkr1 peki1 psk1 #j.
      ISend(<pki1, pkr1, peki1, psk1, kb>) @ #j & #j < #i)
  ==> // then PSK between R and I was compromised (or not in use)
    Ex #j. Reveal_PSK(psk) @ j & #j < #i
  )
  &(All pki pkr peki psk cr kb ka k #i.
    // if R accepts an init msg (claimed to be) from I
    RKeys(<pki, pkr, peki, psk, cr, kb, ka, k>) @ i
    // and the init msg was generated by honest peer I
    // for another peer R' and is replayed or intercepted
    // by the attacker (fields tampered)
    & (Ex pki pkr1 peki1 psk1 #j.
      ISend(<pki, pkr1, peki1, psk1, kb>) @ j
      & #j < #i & not(pkr = pkr1))
  ==> // then PSK between R and I was compromised (or not in use)
    (Ex #j. Reveal_PSK(psk) @ #j & #j < #i)
  )"
```

Identity Hiding. Indentity hiding is a property that was proven in the symbolic proof for WireGuard. When adapting the proof to the KEM setting of PQ-WireGuard Tamarin successfully proves the property. However, it turns out the model in this case makes an idealizing assumption that is not necessarily true in the KEM setting: The Tamarin model assumes a key hiding property, i.e., that a key encapsulation (in case of DH a key exchange message) does not allow to learn under which public key it was produced. For DH key exchange this can be derived from the DDH assumption (against passive adversaries) or the PRFODH assumption (against active adversaries).

For KEMs this assumption is not implied by the standard security assumptions of IND-CPA and IND-CCA security. For a counter example consider a KEM that makes the public key part of a key encapsulation. This does not invalidate IND-CPA or IND-CCA security. What is formally required to justify the applicability of the model is a KEM version of the notion of indistinguishability of keys (IK) [62]. For active adversaries IK-CCA is requried, for passive adversaries IK-CPA is sufficient. In [63] it is shown that the public key encryption scheme underlying Classic McEliece achieves

IK-CPA security. Based on this we conjecture that the Classic McEliece KEM achieves IK-CCA security but we do not formally prove it.

V. Instantiation with McEliece and Saber

The generic approach for a purely KEM-based variant of WireGuard allows us in principle to instantiate the protocol with any post-quantum KEM(s) with the required security properties. In this section we describe the concrete instantiation we chose. We selected the Classic McEliece [22] IND-CCA KEM and an IND-CPA secure variant of Saber [24], [23]. One could say that this choice—just like the choices of primitives in WireGuard—is "cryptographically opinionated". The criteria by which we made this choice are the following:

- stick to primitives that are in the second round of the NIST PQC project and thus have potential to become a future standard;
- choose parameters that reach NIST security level 3 (see [64, Sec. 4.A.5]);
- do not increase the number of required unfragmented IPv6 packets for the handshake compared to WireGuard (one sent by the initiator and one by the responder, plus the key-confirmation by the initiator that is implicit through application-data transmission in WireGuard and explicit in PQ-WireGuard).
- pick primitives that have high-performance timing-attack protected implementations;
- pick "conservative" primitives, i.e, primitives building on a history of cryptanalytic results;
- stay away from primitives that the submitters declare to be encumbered by patents; and
- do not modify or tweak primitives in any way that would invalidate security reductions.

The most limiting of these criteria is to fit the initiator's and the responder's handshake messages into one IPv6 packet each. IPv6 mandates every link in the internet to support an MTU of at least 1280 bytes [20, Sec. 5]. Out of those 1280 bytes, 40 are required for the IPv6 header and another 8 are required for the UDP header. This leaves 1232 bytes for the WireGuard handshake payloads. In the initiator's message, the fields `type`, 0^3, $\texttt{sid}_i$, `ltk`, `time`, `m1`, and `m2` together occupy 116 bytes, which leaves 1116 bytes for a CPAKEM public key and a CCAKEM ciphertext. In the responder's message, the fields `type`, 0^3, $\texttt{sid}_i$, $\texttt{sid}_r$, `zero`, `m1`, and `m2` together occupy 60 bytes, which leaves 1172 bytes for a CPAKEM ciphertext and a CCAKEM ciphertext.

Classic McEliece as CCAKEM. Note in Alg. 2 that the handshake never sends public keys of CCAKEM; also the computation does not involve any CCAKEM.Gen operations. This means that for the instantiation of CCAKEM we are mainly concerned about ciphertext size with secondary criteria being encapsulation and decapsulation speed. Out of all round-2 NIST PQC candidate KEMs[4], Classic McEliece has the smallest ciphertext by far, weighing in at only 188 bytes for the level-3 parameter set `mceliece460896`. Also, Classic McEliece comes with very fast timing-attack-protected software for encapsulation and decapsulation, which makes it the ideal choice of primitive for our use case. Note that McEliece is often regarded as a conservative, but rather inefficient choice, because of its slow key generation and large public keys – however, these disadvantages are precisely the aspects that do not matter for us here.

Tweaked Saber as CPAKEM. With the rather straightforward choice of Classic McEliece as instantiation of CCAKEM fixed, we need to find an IND-CPA KEM among the NIST candidates that has public keys of at most 928 bytes and ciphertexts of at most 984 bytes for parameters that reach the NIST security level 3. The only KEMs that meet these criteria are Round5 [65], SIKE [66], and ROLLO-I [67]. Unfortunately, none of these three meets our other criteria. Round-5 is covered by patents held by the submitters; SIKE is rather slow, for example more than an order of magnitude slower than most lattice-based KEMs, and ROLLO-I cannot be seen as a particularly conservative choice. Specifically, in the document explaining the choice of round-2 candidates [68], NIST writes about the rank-based candidate ROLLO-I:

> *"Nonetheless rank-based cryptography is quite new and not as well studied as lattice-based cryptography or code-based cryptography using the Hamming metric. More cryptanalysis on rank-based primitives would be valuable."*

However, among the remaining candidates, there are multiple lattice-based KEMs with public keys and ciphertext that are only slightly larger than what we need. Also, most of them aim for IND-CCA security (which we do *not* need to instantiate CPAKEM) and some of them allow to reduce the size of public keys and ciphertexts at the expense of achieving only IND-CPA security and increasing failure probability.

Concretely, Saber already includes public-key and ciphertext compression, and, in order to achieve IND-CCA security, carefully chooses parameters to minimize sizes while keeping the failure probability δ cryptographically negligible. We decided to propose an IND-CPA version of Saber, which compresses public keys and ciphertexts even further. This comes at the additional advantage that the underlying hard lattice problem becomes harder, but at the expense of significantly increased failure probability. Specifically, the Saber specification states that *"a higher choice for parameters p and T, will result in lower security, but higher correctness"* [24, Sec. 2.2]; the parameters p and T are precisely what controls public-key and ciphertext sizes.

The original parameters for the level-3 parameters of Saber use $p = 2^{10}$ and $T = 2^4$; we propose to use $p = 2^9$ and $T = 2^3$ for an IND-CPA variant of Saber. In the following we will refer to this variant of Saber as "Dagger". Compared to Saber, the modifications in Dagger reduce the public-key size from 992 bytes to 896 bytes and the ciphertext size from 1088 bytes to 960 bytes, which is well within our limits. To

[4]For an overview, see https://pqc-wiki.fau.edu/

analyze the failure rate and bit security of Dagger, we adapt the Python script that comes with the Saber submission package to run on the new parameters. This adapted Python script is included with the software package at https://cryptojedi.org/crypto/#pqwireguard. Compared to Saber, the post-quantum bit security of Dagger increases from 180 to 198 bits; the failure probability increases from $2^{-136.14}$ to $2^{-25.25}$. Note that on the protocol level such a failure has a similar effect to a failed UDP packet transmission. Essentially it means that about one out of every 40 million handshakes will need to be repeated. In addition to the modified values of p and T, Dagger does not use the Fujisaki-Okamoto transform [30], i.e., the construction that Saber uses to build an IND-CCA KEM from an IND-CPA public-key encryption scheme. For a pseudocode description of Dagger see Appendix A.

VI. Performance Analysis

In this Section, we present performance benchmarks of our proposal of PQ-WireGuard and compare to original WireGuard (version 0.0.20191206), IPsec (strongSwan in version U5.6.2/K4.15.0-72-generic), OpenVPN (version 2.4.4, linked against OpenSSL 1.1.1), OpenVPN-NL (version 2.4.7, linked against mbed TLS 2.16.2), and PQCrypto-VPN (OpenVPN 2.4.4, linked against OQS-OpenSSL 1.0.2 [69]). OpenVPN-NL is a branch of OpenVPN, which is mandated for critical infrastructure in the Netherlands by the Dutch government, while PQCrypto-VPN is the aforementioned VPN software from Microsoft [46] based on OpenVPN and the Open Quantum Safe (OQS) framework [69]. Note that PQCrypto-VPN has optional post-quantum authentication using the Picnic signature scheme [70], [71]; in our experiments we do not use this option, but benchmark PQCrypto-VPN only with post-quantum confidentiality. To achieve this post-quantum confidentiality, PQCrypto-VPN has two options, both provided through OQS: either SIDH-503 as described in [72] or Frodo-752 as described in [73]. For WireGuard we report resources including the first application-data packet, which also serves as key confirmation. In this packet we use zero-length application data. In other words, we consider the handshake finished on the responder (server) side only at the point when the server is ready to send application data.

Our implementation of the PQ-WireGuard software is based on the original WireGuard implementation. For Classic McEliece we use the "`avx2`" software targeting recent 64-bit Intel processors, which has been submitted to SUPERCOP [74] by the Classic McEliece team. For the implementation of Dagger we start from the Saber reference implementation and adapt the files `kem.c` (to remove the CCA transform) and `SABER_params.h` (to change the values of p and T).

We carried out the experiments between two virtual machines managed by VMware's "vSphere" in version 6.7 and connected through a virtual Ethernet link (VMware "vSwitch") with a bandwidth limit of 10 Gbit/s. Both virtual machines are running Linux kernel 4.15.0. The underlying physical machine is powered by Intel Xeon Gold 6130 (Skylake) CPUs running at 2.1 GHz.

We compare the handshake efficiency by the following metrics: the amount of traffic, the number of packets exchanged, and the time span of the handshake. The client time span is the elapsed time between when the client starts any computation for a handshake and when session keys are derived from the handshake on the client side. Similarly, the server time span is the elapsed time between the server receiving an initiation packet from the client and the server being ready to send application data to the client.

The handshake protocol of each VPN software was invoked for 1000 times to compute the average and standard deviation (enclosed by parentheses) of those metrics. The results with IPv4 and IPv6 are presented in Table III and Table IV, respectively. In both tables, the amount of traffic includes the 14-byte Ethernet frame headers.

TABLE III
Resource Requirements for VPN handshake protocols over IPv4, numbers in parentheses are standard deviation

VPN Software	Packet Number	Traffic (bytes)	Client Time (milliseconds)	Server Time (milliseconds)
WireGuard	3 (0)	398 (0)	0.584 (0.508)	0.494 (0.507)
PQ-WireGuard (this paper)	3 (0)	2594 (0)	0.975 (0.442)	0.745 (0.245)
IPsec (RSA-2048)	6 (0)	4123 (0)	17.046 (0.826)	11.823 (0.726)
IPsec (Curve25519)	4 (0)	2145 (0)	5.127 (0.375)	2.807 (0.431)
OpenVPN (RSA-2048)	21.005 (0.071)	7535.507 (7.940)	1150.872 (244.288)	1144.994 (251.304)
OpenVPN (NIST P-256)	19.005 (0.007)	5408.572 (7.997)	1152.238 (242.014)	1150.310 (253.582)
OpenVPN-NL (RSA-2048)	19.005 (0.007)	5685.585 (8.155)	1157.732 (244.015)	1151.446 (246.534)
OpenVPN-NL (NIST P-256)	19.006 (0.078)	5681.711 (8.979)	1159.099 (241.534)	1156.482 (235.703)
PQ-OpenVPN (Frodo-752)	63.001 (0.032)	34348.114 (3.569)	1151.529 (235.234)	1143.337 (238.465)
PQ-OpenVPN (SIDHp503)	23.003 (0.055)	8536.345 (6.188)	1266.838 (258.101)	1265.332 (264.271)

TABLE IV
Resource Requirements for VPN handshake protocols over IPv6, numbers in parentheses are standard deviation

VPN Software	Packet Number	Traffic (bytes)	Client Time (milliseconds)	Server Time (milliseconds)
WireGuard	3 (0)	458 (0)	0.592 (0.399)	0.480 (0.389)
PQ-WireGuard (this paper)	3 (0)	2654 (0)	1.015 (0.618)	0.786 (0.621)
IPsec (RSA-2048)	6 (0)	4299 (0)	17.188 (0.712)	11.912 (0.535)
IPsec (Curve25519)	4 (0)	2281 (0)	5.226 (0.575)	2.822 (0.436)
OpenVPN (RSA-2048)	21.003 (0.055)	7955.409 (7.319)	1148.733 (250.513)	1142.650 (243.184)
OpenVPN (NIST P-256)	19.005 (0.007)	5788.610 (9.423)	1139.140 (247.659)	1133.944 (240.691)
OpenVPN-NL (RSA-2048)	19.005 (0.072)	6065.700 (9.665)	1162.649 (261.078)	1151.790 (246.363)
OpenVPN-NL (NIST P-256)	19.001 (0.003)	6061.138 (4.304)	1159.627 (252.989)	1153.949 (247.470)
PQ-OpenVPN (Frodo-752 [73])	63.006 (0.078)	35608.817 (10.324)	1160.922 (259.246)	1155.713 (245.614)
PQ-OpenVPN (SIDHp503)	23.005 (0.072)	8996.684 (9.449)	1277.172 (251.461)	1269.074 (257.427)

We see that both WireGuard and PQ-WireGuard only require 3 packets. We also see that in PQ-WireGuard, the total time required for the handshake on the client side increases by less than 70% compared to WireGuard, at least when it is run over a high-speed network link as in our experiments. The time required for server-side computations increases by just over 50% compared to WireGuard. The computational effort for both WireGuard and PQ-WireGuard are dominated by public-key cryptography; we would expect that future improvements to the McEliece or Dagger software will bring PQ-WireGuard even closer to the performance of WireGuard.

Just as the original WireGuard software, PQ-WireGuard outperforms the main competitors IPsec and OpenVPN in terms of handshake time, computation time on the server, number of transmitted packets, and amount of transmitted data. Specifically, the PQ-WireGuard handshake is about 5 times faster than the handshake of IPsec and more than three orders of magnitude faster than the handshake of OpenVPN, all while offering full protection against future attackers equipped with large quantum computers.

Acknowledgements

We would like to thank Benjamin Dowling and Kenneth G. Paterson for helpful discussions and the LaTeX sources of their proof. We would also like to thank the anonymous reviewers for valuable comments.

This work has been supported by the European Commission through the ERC Starting Grant 805031 (EPOQUE), and by the Dutch Ministry of Economic Affairs and Climate Policy through the WBSO R&D tax credit.

References

[1] J. Donenfeld, "WireGuard: Next Generation Kernel Network Tunnel," in *24th Annual Network and Distributed System Security Symposium*. Internet Society, 2017.

[2] T. Perrin, "Noise protocol framework," https://noiseprotocol.org/noise.pdf (accessed 2019-10-22).

[3] D. J. Bernstein, "Curve25519: new Diffie-Hellman speed records," in *Public Key Cryptography – PKC 2006*, ser. LNCS, vol. 3958. Springer, 2006, pp. 207–228.

[4] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein, "BLAKE2: Simpler, smaller, fast as MD5," in *Applied Cryptography and Network Security – ACNS 2013*, ser. LNCS, vol. 7954. Springer, 2013, pp. 119–135.

[5] D. J. Bernstein, "The Poly1305-AES message-authentication code," in *Fast Software Encryption*, ser. LNCS, vol. 3557. Springer, 2005, pp. 32–49.

[6] ——, "ChaCha, a variant of Salsa20," in *Workshop Record of SASC 2008: The State of the Art of Stream Ciphers*, 2008, http://cr.yp.to/papers.html#chacha.

[7] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF protocols," IETF RFC 8439, 2018.

[8] Jason Donenfeld and Kevin Milner, "Formal verification of the WireGuard protocol," 2018, version June 7, 2018, https://www.wireguard.com/papers/wireguard-formal-verification.pdf.

[9] B. Dowling and K. G. Paterson, "A cryptographic analysis of the WireGuard protocol," in *Applied Cryptography and Network Security.*, ser. LNCS, vol. 10892. Springer, 2018.

[10] "BoringTun," https://github.com/cloudflare/boringtun.

[11] L. Torvalds, "Re: [GIT] Networking," Posting to the Linux kernel mailing list, 2018, http://lkml.iu.edu/hypermail/linux/kernel/1808.0/02472.html.

[12] J. Appelbaum, C. Martindale, and P. Wu, "Tiny WireGuard tweak," in *Progress in Cryptology – AFRICACRYPT 2019*, ser. LNCS, vol. 11627. Springer, 2019, pp. 3–20.

[13] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes, "CSIDH: An efficient post-quantum commutative group action," in *Advances in Cryptology – ASIACRYPT 2018*, ser. LNCS, vol. 11274. Springer, 2018, pp. 395–427.

[14] X. Bonnetain and A. Schrottenloher, "Quantum security analysis of CSIDH," in *Advances in Cryptology – EUROCRYPT 2020*, ser. LNCS, vol. 12106. Springer, 2020, pp. 493–522.

[15] D. J. Bernstein, T. Lange, C. Martindale, and L. Panny, "Quantum circuits for the CSIDH: Optimizing quantum evaluation of isogenies," in *Advances in Cryptology – EUROCRYPT 2019*, ser. LNCS, vol. 11477. Springer, 2019, pp. 409–441.

[16] C. Peikert, "He gives C-sieves on the CSIDH," in *Advances in Cryptology – EUROCRYPT 2020*, ser. LNCS, vol. 12106. Springer, 2020, pp. 463–492.

[17] D. J. Bernstein, "Re: [pqc-forum] new quantum cryptanalysis of CSIDH," Posting to the NIST pqc-forum mailing list, 2019, https://groups.google.com/a/list.nist.gov/forum/#!original/pqc-forum/svm1kDy6c54/0gFOLitbAgAJ.

[18] A. Fujioka, K. Suzuki, K. Xagawa, and K. Yoneyama, "Strongly secure authenticated key exchange from factoring, codes, and lattices," in *Public-Key Cryptography – PKC 2012*, ser. LNCS. Springer, 2012, pp. 467–484.

[19] A. Atlasis, "Attacking IPv6 implementation using fragmentation," Blackhat Europe, 2012, http://media.blackhat.com/bh-eu-12/Atlasis/bh-eu-12-Atlasis-Attacking_IPv6-WP.pdf.

[20] S. Deering and R. Hinden, "Internet protocol, version 6 (IPv6) specification," IETF RFC 8200, 2017.

[21] D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, L. D. Feo, B. Hess, A. Jalali, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, J. Renes, V. Soukharev, D. Urbanik, and G. Pereira, "Supersingular isogeny key encapsulation," Round-2 submission to the NIST PQC project, 2019.

[22] D. J. Bernstein, T. Chou, T. Lange, I. von Maurich, R. Misoczki, R. Niederhagen, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, and W. Wang, "Classic McEliece: conservative code-based cryptography," Round-2 submission to the NIST PQC project, 2019.

[23] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, "Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM," in *Progress in Cryptology – AFRICACRYPT 2018*, ser. LNCS, vol. 10831. Springer, 2018, pp. 282–305.

[24] ——, "SABER: Mod-LWR based KEM (round 2 submission)," Round-2 submission to the NIST PQC project, 2019.

[25] D. J. Bernstein, "Re: [pqc-forum] ROUND 2 OFFICIAL COMMENT: NewHope," Posting to the NIST pqc-forum mailing list, 2019, https://groups.google.com/a/list.nist.gov/forum/#!original/pqc-forum/u3FoYrN-7fk/3EZwDIvDBQAJ.

[26] ——, "Re: [pqc-forum] ROUND 2 OFFICIAL COMMENT: NewHope," Posting to the NIST pqc-forum mailing list, 2019, https://groups.google.com/a/list.nist.gov/forum/#!original/pqc-forum/u3FoYrN-7fk/MxBVn9M7CQAJ.

[27] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "NewHope without reconciliation," Cryptology ePrint Archive, Report 2016/1157, 2016, https://eprint.iacr.org/2016/1157.

[28] T. Perrin, "KEM-based hybrid forward secrecy for Noise," 2018, https://github.com/noiseprotocol/noise_hfs_spec/blob/master/output/noise_hfs.pdf.

[29] C. Boyd, Y. Cliff, J. G. Nieto, and K. G. Paterson, "Efficient one-round key exchange in the standard model," in *Information Security and Privacy*, ser. LNCS, vol. 5107. Springer, 2008, pp. 69–83.

[30] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *Advances in Cryptology - CRYPTO '99*, ser. LNCS, vol. 1666. Springer, 1999, pp. 537–554.

[31] K. Hövelmanns, E. Kiltz, S. Schäge, and D. Unruh, "Generic authenticated key exchange in the quantum random oracle model," Cryptology ePrint Archive, Report 2018/928, 2018, https://eprint.iacr.org/2018/928.

[32] Understanding and C. A. via Double-key Key Encapsulation Mechanism, "Haiyang xue and xianhui lu and bao li and bei liang and jingnan he," in *Advances in Cryptology – ASIACRYPT 2018*, ser. LNCS, vol. 11274. Springer, 2018, pp. 158–189.

[33] J. Zhang, Z. Zhang, J. Ding, M. Snook, and Ö. Dagdelen, "Authenticated key exchange from ideal lattices," in *Advances in Cryptology –*

EUROCRYPT 2015, ser. LNCS, vol. 9057. Springer, 2015, pp. 719–751.

[34] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé, "CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM," in *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018*. IEEE, 2018, pp. 353–367.

[35] P. Longa, "A note on post-quantum authenticated key exchange from supersingular isogenies," Cryptology ePrint Archive, Report 2018/267, 2018.

[36] X. Xu, H. Xue, K. Wang, , M. H. Au, B. Liang, and S. Tian, "Strongly secure authenticated key exchange from supersingular isogenies," in *Advances in Cryptology – ASIACRYPT 2019*, ser. LNCS, vol. 11921. Springer, 2019, pp. 278–308.

[37] A. Fujioka, K. Takashima, S. Terada, and K. Yoneyama, "Supersingular isogeny diffie-hellman authenticated key exchange," in *Information Security and Cryptology – ICISC 2018*, ser. LNCS, vol. 11396. Springer, 2019, pp. 177–195.

[38] B. Lipp, B. Blanchet, and K. Bhargavan, "A mechanised cryptographic proof of the WireGuard virtual private network protocol," in *IEEE European Symposium on Security and Privacy (EuroS&P'19)*. IEEE, 2019, pp. 231–246.

[39] B. Dowling, P. Rösler, and J. Schwenk, "Flexible authenticated and confidential channel establishment (fACCE): Analyzing the noise protocol framework," Cryptology ePrint Archive, Report 2019/436, 2019, https://eprint.iacr.org/2019/436.

[40] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "On the security of TLS-DHE in the standard model," in *Advances in Cryptology – CRYPTO 2012*, ser. LNCS, vol. 7417. Springer, 2012, pp. 273–293.

[41] N. Kobeissi, G. Nicolas, and K. Bhargavan, "Noise Explorer: Fully automated modeling and verification for arbitrary Noise protocols," in *IEEE European Symposium on Security and Privacy (EuroS&P'19)*. IEEE, 2019, pp. 356–370.

[42] B. Blanchet, "ProVerif: Cryptographic protocol verifier in the formal model," https://prosecco.gforge.inria.fr/personal/bblanche/proverif/ (accessed 2019-10-21).

[43] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN prover for the symbolic analysis of security protocols," in *Computer Aided Verification*, ser. LNCS, vol. 8044. Springer, 2013, pp. 696–701.

[44] O. Inc., "VPN software solutions & services for business | OpenVPN," 2019, https://openvpn.net/ (accessed 2019-10-21).

[45] S. de Vries, "Achieving 128-bit security against quantum attacks in OpenVPN," Master's thesis, University of Twente, 2016, https://essay.utwente.nl/70677/1/2016-08-09%20MSc%20Thesis%20Simon%20de%20Vries%20final%20color.pdf.

[46] K. Easterbrook, K. Kane, B. LaMacchia, D. Shumow, and G. Zaverucha, "Post-quantum cryptography VPN," 2019, https://www.microsoft.com/en-us/research/project/post-quantum-crypto-vpn/.

[47] H. Krawczyk, "HMQV: A high-performance secure Diffie-Hellman protocol," Cryptology ePrint Archive, Report 2005/176, 2005, https://eprint.iacr.org/2005/176, extended abstract published at Crypto'05.

[48] A. Fujioka and K. Suzuki, "Sufficient condition for identity-basedauthenticated key exchange resilientto leakage of secret keys," in *Information Security and Cryptology – ICISC 2011*, ser. LNCS, vol. 7259. Springer, 2011, pp. 490–509.

[49] C. Cremers and M. Feltz, "Beyond eCK: perfect forward secrecy under actor compromise and ephemeral-key reveal," *Designs, Codes and Cryptography*, vol. 74, no. 1, pp. 183–218, 2015.

[50] B. A. LaMacchia, K. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," in *Provable Security*, ser. LNCS, vol. 4784. Springer, 2007, pp. 1–16.

[51] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *SFCS '94 Proceedings of the 35th Annual Symposium on Foundations of Computer Science*. IEEE, 1994, p. 124–134.

[52] ——, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, p. 1484–1509, 1997.

[53] E. S. V. Freire, D. Hofheinz, E. Kiltz, and K. G. Paterson, "Non-interactive key exchange," in *Public-Key Cryptography – PKC 2013*, ser. LNCS, vol. 7778. Springer, 2013, pp. 254–271.

[54] J. Brendel, M. Fischlin, F. Günther, and C. Janson, "PRF-ODH: Relations, instantiations, and impossibility results," Cryptology ePrint Archive, Report 2017/517, 2017, https://eprint.iacr.org/2017/517.

[55] A. W. Dent, "A Designer's Guide to KEMs," in *Cryptography and Coding*, ser. LNCS, vol. 2898. Springer, 2003, pp. 133–151.

[56] M. Bellare and A. Lysyanskaya, "Symmetric and dual prfs from standard assumptions: A generic validation of an hmac assumption," Cryptology ePrint Archive, Report 2015/1198, 2015, https://eprint.iacr.org/2015/1198.

[57] P. Rogaway, "Authenticated-encryption with associated-data," in *CCS '02: Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM, 2002, pp. 98–107.

[58] A. Fujioka, K. Suzuki, K. Xagawa, and K. Yoneyama, "Strongly secure authenticated key exchange from factoring, codes, and lattices," *Design, Codes, and Cryptography*, vol. 76, no. 3, pp. 469–504, 2015.

[59] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, "A comprehensive symbolic analysis of tls 1.3," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. ACM, 2017, pp. 1773–1788.

[60] D. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, and V. Stettler, "A formal analysis of 5g authentication," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. ACM, 2018, pp. 1383–1396.

[61] T. T. Team, "Tamarin-prover manual," https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf.

[62] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval, "Key-privacy in public-key encryption," in *Advances in Cryptology — ASIACRYPT 2001*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 566–582.

[63] Y. Yoshida, K. Morozov, and K. Tanaka, "Cca2 key-privacy for code-based encryption in the standard model," in *Post-Quantum Cryptography*. Cham: Springer International Publishing, 2017, pp. 35–50.

[64] N. I. for Standards and Technology, "Submission requirements and evaluation criteria for the post-quantum cryptography standardization process," 2016, https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[65] H. Baan, S. Bhattacharya, S. Fluhrer, O. Garcia-Morchon, T. Laarhoven, R. Player, R. Rietman, M.-J. O. Saarinen, L. Tolhuizen, J. e Luis Torre-Arce, , and Z. Zhang, "Round5: KEM and PKE based on (ring) learning with rounding," Round-2 submission to the NIST PQC project, 2019.

[66] D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, L. D. Feo, B. Hess, A. Jalali, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, G. Pereira, J. Renes, V. Soukharev, and D. Urbanik, "Supersingular isogeny key encapsulation," Round-2 submission to the NIST PQC project, 2019.

[67] C. A. Melchor, N. Aragon, M. Bardet, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, A. Hauteville, A. Otmani, O. Ruatta, J.-P. Tillich, and G. Zémor, "ROLLO – Rank-Ouroboros, LAKE & LOCKER," Round-2 submission to the NIST PQC project, 2019, https://pqc-rollo.org/documentation.html.

[68] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, "Status report on the first round of the NIST post-quantum cryptography standardization process," NISTIR 8240, 2019, https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8240.pdf.

[69] M. Mosca and D. Stebila, "Open quantum safe," 2020, https://openquantumsafe.org/ (accessed 2020-01-30).

[70] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha, "Post-quantum zero-knowledge and signatures from symmetric-key primitives," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS'17*. ACM, 2017, pp. 1825–1842.

[71] M. Chase, D. Derler, S. Goldfeder, J. Katz, V. Kolesnikov, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, X. Wang, and G. Zaverucha, "The Picnic signature scheme – design document," Round-2 submission to the NIST PQC project, 2019.

[72] C. Costello, P. Longa, and M. Naehrig, "Efficient algorithms for supersingular isogeny diffie-hellman," in *Advances in Cryptology – CRYPTO 2016*, ser. LNCS, vol. 9814. Springer, 2016, pp. 572–601.

[73] J. W. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila, "Frodo: Take off the ring! practical, quantum-secure key exchange from LWE," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS'16*. ACM, 2016, pp. 1006–1018.

[74] D. J. Bernstein and T. Lange, "eBACS: ECRYPT benchmarking of cryptographic systems," http://bench.cr.yp.to (accessed 2020-01-22).

APPENDIX

A. The Dagger IND-CPA-secure KEM

A description of the Dagger KEM is given in Algorithms 3, 4, and 5. These algorithms use as underlying routines Saber.PKE routines and related notation defined in [24, Sec. 2.4]. The parameters Dagger uses to instantiate Saber.PKE are $l = 3$, $n = 256$, $q = 2^{13}$, $p = 2^9$, $T = 2^3$, $\mu = 8$. These are the same parameters as listed for Saber-PKE in [24, Table 1], except that we decrease p and T for smaller public key and ciphertext and higher security of the underlying lattice problem at the cost of increased failure probability.

Algorithm 3 `Dagger.KEM.KeyGen()`

$(seed_{\mathbf{A}}, \mathbf{b}, \mathbf{s}) =$ `Saber.PKE.KeyGen()`
Return $(pk := (seed_{\mathbf{A}}, \mathbf{b}), sk := \mathbf{s})$

Algorithm 4 `Dagger.KEM.Encaps()`

$m \leftarrow \mathcal{U}(\{0,1\}^{256})$
$(\hat{K}, r) = \mathcal{G}(m)$
$c =$ `Saber.PKE.Enc`$(pk, \hat{K}; r)$
$K = \mathcal{H}(\hat{K})$
Return (c, K)

Algorithm 5 `Dagger.KEM.Decaps()`

$\hat{K}' =$ `Saber.PKE.Dec`$(\mathbf{s}, c)$
Return $K' = \mathcal{H}(\hat{K}')$

B. Computational Proof

We prove that the PQ-WireGuard handshake protocol (from now on abbreviated pqWG) is eCK-PFS-PSK-secure with cleanness predicate $\mathsf{clean}_{\text{eCK-PFS-PSK}}$ (capturing perfect forward secrecy and resilience to KCI attacks). That is, for any QPT algorithm $\mathcal{A}$ against the eCK-PFS-PSK key-indistinguishability game $\mathsf{Adv}^{\text{eCK-PFS-PSK}}_{\text{pqWG},\mathsf{clean}_{\text{eCK-PFS-PSK}},n_S,n_P,\mathcal{A}}(\lambda)$ is negligible under the dual-prf, auth-aead, IND-CPA and IND-CCA assumptions.

Our proof largely follows the proof by Dowling and Paterson [9] with as little modification as necessary. The only game hops that are really affected are those that are based on the different PRFODH-assumptions and the DDH-assumption. This concerns **Game 5** of **Case 1**, **Game 5** of **Case 2**, **Game 3** of **Case 3.2**, **Game 3** of **Case 3.3**, **Game 3** of **Case 3.4** and **Game 3** of **Case 3.5** . Of these **Game 3** of **Case 3.5** is the only one relying on the sym-mm-PRFODH assumption and **Game 3** of **Case 3.2** is the only one relying on the DDH-assumption. All others rely on sym-ms-PRFODH assumption.

This distinction has its correspondence in our proof as well: **Case 3.2** which only relied on the DDH assumption is now based on the IND-CPA-security of CPAKEM, whereas the PRFODH cases are now based on the IND-CCA-security of CCAKEM. The case in which the original protocol relied on the stronger sym-mm-PRFODH assumption is special in our case as well and will be discussed later.

Finally we had to modify **Game 3** of **Case 3.1** to fix a bug that we discovered in the original proof.

Given the similarity between the cases that relied on the sym-ms-PRFODH assumption in the original proof we will start by providing a detailed proof for **Game 5** of **Case 1** and discuss the changes necessary for the other cases afterwards.

We perform the following changes: First we rename all occurances of HKDF to KDF in order to stay consistent with our protocol descriptions. Then we seperate between the eCK-PFS-PSK-adversary $\mathcal{A}$ and the adversaries $\mathcal{R}_i$ against the security assumptions. For this we define $\mathcal{R}$ as the set of all Algorithms that are defined by a game-hop to possibly break a property prop of a building block F. With this we furthermore define $\mathsf{Adv}^{\mathsf{prop}}_{\mathsf{F},\mathcal{R}}(\lambda)$ to mean the maximum success probability over all $\mathcal{R}_i \in \mathcal{R}$ against the property prop of the building block F.

In practice both of these changes are essentially just syntactical. We will now go on to describe more substantial changes to the actual game-hops.

Game 5. In this game we replace the computation of C_3, κ_3 with uniformly random and independent values $\widetilde{C_3}, \widetilde{\kappa_3}$. We note that the replacement of the sym-ms-PRFODH assumptions with the more standard IND-CCA assumption for KEMs forces us to split the original game hop into three hops. This is necessary because of the more convoluted combination of the static keys with both the other party's static and ephemeral keys and because the application of the KDF to the shared-secret is not part of the IND-CCA game while it was part of the PRFODH game. As such we first replace the pseudo-random value used for key-encapsulation with CCAKEM with a truly random value (**Game 5a**) and then replace `shk1` with a random value `k`* (**Game 5b**). After that we replace the output of the KDF that this value is passed to with a random one (**Game 5c**). The reason for why we split the game hop instead of inserting new ones is that we want to preserve consistency with the numbering in the original proof.

The one case where we will deviate from the original numbering-scheme is in the labels for the "break"-events in **Case 1**: The original proof numbers these such that $\Pr(break_4)$ is the probability that the fifth hybrid is broken; in all other cases the numbers coincide however. Because we believe that skipping $break_4$ and increasing all following indices by one is more readable and since this is what we do in the full version, the indices in our proof don't mach the ones from the original proof here.

In **Game 5a** we replace the value $\hat{r} := \mathsf{KDF}(\sigma_i, r_i)$ passed to CCAKEM.Enc for the computation of `ct1` and `shk1` with a random bitstring $\hat{r}'$.

By the definition of this case, we know that at least one of r_i and σ_i is random and uncorrupted.

In the first case (r_i is unknown to the adversary), we initialize a $\mathsf{prf}^{\mathsf{swap}}$ challenger, query σ_i, and use the output $\widetilde{r}$ from the $\mathsf{prf}^{\mathsf{swap}}$ challenger to replace the computation of

$\hat{r}$. By the definition of this case r_i is a uniformly random and independent value, therefore this replacement is sound. If the test bit sampled by the $\mathsf{prf}^{\mathsf{swap}}$ challenger is 0, then $\hat{r} \leftarrow \mathsf{KDF}(\sigma_i, r_i)$ and we are in **Game 4**. If the test bit sampled by the $\mathsf{prf}^{\mathsf{swap}}$ challenger is 1, then $\hat{r} \xleftarrow{\$} \{0,1\}^{|\mathsf{KDF}|}$ is a truly random value and we are in **Game 5a**.

For the second case we first establish that r_i, while being (potentially) known to the adversary is still fresh in the sense that $\mathsf{KDF}(\sigma_i, r_i)$ has never been evaluated: Since r_i is a random value, there is a chance that it could be sampled in another session. This probability can be upper-bounded by the total number of sessions divided by the number of possible values, namely $\frac{n_S}{2^{|r_i|}}$.

Given that, we initialize a prf challenger and replace all computations of $\mathsf{KDF}(\sigma_i, \cdot)$ with queries to the challenger. By the definition of this case σ_i is a uniformly random and independent value, therefore this replacement is sound. If the test bit sampled by the prf challenger is 0, then $\hat{r} \leftarrow \mathsf{KDF}(\sigma_i, r_i)$ and we are in **Game 4**. If the test bit sampled by the prf challenger is 1, then $\hat{r} \xleftarrow{\$} \{0,1\}^{|\mathsf{KDF}|}$ is a truly random value. Since we established furthermore that r_i is not used with σ_i in any other session, $\hat{r}$ is furthermore independent of all other $\hat{r}$ in other sessions, therefore we are in **Game 5a**.

Thus any adversary $\mathcal{A}$ capable of distinguishing this change can be turned into a successful adversary against the prf security or the $\mathsf{prf}^{\mathsf{swap}}$ security of KDF, and we find:

$$\begin{aligned}&\Pr(abort_{\mathtt{accept}})\\ &\leq \frac{n_S}{2^{|r_i|}} + \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{KDF},\mathcal{R}}(\lambda) + \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{KDF},\mathcal{R}}(\lambda) + \Pr\left(break_{5a}\right)\end{aligned}$$

In **Game 5b** we replace the computation of `shk1` by sampling the value uniformly at random from the space of shared secrets of the KEM and ignoring the second output of $\mathsf{CCAKEM.Enc}(\mathtt{spk}_r)$. To show that this is undetectable under the IND-CCA-assumption of the used KEM, we interact with an IND-CCA challenger in the following way: Note that by **Game 1**, we know at the beginning of the experiment the index of session π_i^s such that $\mathsf{Test}(i,s)$ is issued by the adversary. Similarly, by **Game 2**, we know at the beginning of the experiment the index of the intended partner P_j of the session π_i^s. Thus, we initialize an IND-CCA challenger and use the received public-key $\mathtt{pk}^*$ as long-term public-key of party P_j and give it with all other (honestly generated) public keys to the adversary. Note that by **Game 4** and the definition of this case, $\mathcal{A}$ is not able to issue a $\mathsf{CorruptASK}(j)$ query, as we abort if $\pi_i^s.\alpha \leftarrow \mathtt{reject}$ and abort if $\pi_i^s.\alpha \leftarrow \mathtt{accept}$. Thus we will not need to reveal the private key $\mathtt{sk}^*$ of the challenge public-key to $\mathcal{A}$. However we must account for all sessions t such that π_j^t must use the private key for computations. In PQ-WireGuard, the long-term private keys are used to compute the following:

- In sessions where P_j acts as the initiator: $C_8 \leftarrow \mathsf{KDF}(C_6, \mathsf{CCAKEM.Dec}(\mathtt{ssk}_i, \mathtt{ct3}))$
- In sessions where P_j acts as the responder: $C_3, \kappa_3 \leftarrow \mathsf{KDF}(C_2, \mathsf{CCAKEM.Dec}(\mathtt{ssk}_r, \mathtt{ct1}))$

(Note that these are fewer cases than in the original proof because we don't combine static and ephemeral keys directly.) Dealing with the challenger's computation of these values will be done in two ways:

- The encapsulation was created by another honest party. The challenger can then use its own internal knowledge of the encapsulated value to complete the computations.
- The encapsulation was not created by another honest party, but by the adversary and the challenger is therefore unaware of the encapsulated value.

In the second case, the challenger can instead use the decapsulation-oracle provided by the CCA-challenger, specifically querying $\mathsf{CCAKEM.Dec}(\mathtt{ctX})$, (where `ctX` is the relevant encapsulation) which will output `shkX` using the CCA challenger's internal knowledge of $\mathtt{sk}^*$.

During session i we request a challenge consisting of a ciphertext and a candidate shared secret $(\mathtt{c}^*, \mathtt{k}^*)$ from the IND-CCA challenger and use those values in place of `ct1` and `shk1`. Given the definition of the IND-CCA game, there are two cases:

- If the test bit sampled by the IND-CCA challenger is 0, then $\mathtt{k}^*$ is indeed the shared secret encapsulated in $\mathtt{c}^*$ and we are in **Game 5a**.
- If the test bit sampled by the IND-CCA challenger is 1, then $\mathtt{k}^*$ is not the shared secret encapsulated in $\mathtt{c}^*$ but sampled uniformly at random from the space of shared secrets and we are in **Game 5b**.

Thus, any adversary $\mathcal{A}$ capable of distinguishing this change can be turned into a successful adversary against the IND-CCA security of the used KEM and we find:

$$\Pr(break_{5a}) \leq \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{R}}(\lambda) + \Pr(break_{5b})$$

In **Game 5c** we replace the values of C_3, κ_3 with uniformly random and independent values $\widetilde{C_3}, \widetilde{\kappa_3} \xleftarrow{\$} \{0,1\}^{|\mathsf{KDF}|}$ (where $\{0,1\}^{|\mathsf{KDF}|}$ is the output space of the KDF) used in the protocol execution of the test session. Specifically, we initialize a $\mathsf{prf}^{\mathsf{swap}}$ challenger and query `shk1`, and use the output $\widetilde{C_3}, \widetilde{\kappa_3}$ from the $\mathsf{prf}^{\mathsf{swap}}$ challenger to replace the computation of C_3, κ_3. Since by **Game 5b**, `shk1` is a uniformly random and independent value, this replacement is sound. If the test bit sampled by the $\mathsf{prf}^{\mathsf{swap}}$ challenger is 0, then $\widetilde{C_3}, \widetilde{\kappa_3} \leftarrow \mathsf{KDF}(C_2, \mathtt{shk1})$ and we are in **Game 5b**. If the test bit sampled by the $\mathsf{prf}^{\mathsf{swap}}$ challenger is 1, then $\widetilde{C_3}, \widetilde{\kappa_3} \xleftarrow{\$} \{0,1\}^{|\mathsf{KDF}|}$ and we are in **Game 5c**.

Thus any adversary $\mathcal{A}$ capable of distinguishing this change can be turned into a successful adversary against the $\mathsf{prf}^{\mathsf{swap}}$ security of KDF, and we find:

$$\Pr(break_{5b}) \leq \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{KDF},\mathcal{R}}(\lambda) + \Pr(break_{5c})$$

Regarding the other games that need to be replaced: **Game 3** of **Case 3.3** is very similar to **Game 5** of **Case 1**. The biggest difference is that the case-distinction in the first sub-game is no longer necessary since the definition of the case ensures

that the ephemeral key of the initiator is uncorrupted. As such the first case can be removed. Furthermore the references to the surrounding games have to be updated as listed below.

Game 5 of **Case 2** and **Game 3** of **Case 3.4**, which are again (except for the first sub-game, their number and the references) almost identical to each other, only differ slightly from **Game 5** of **Case 1**. In order to refit the proof to them perform the following changes, except for leaving the listing after the first paragraph in the second sub-game that lists the uses of the uncorrupted static key alone:

- In the first sub-game replace all occurrences of X_i with X_r for all identifiers X.
- In **Game 3** of **Case 3.4** remove the first case of the first sub-game (as in **Case 3.3**).
- Replace all occurrences of $\widetilde{C_3}, \widetilde{\kappa_3}$ with $\widetilde{C_8}$.
- Replace all occurrences of C_3, κ_3 with C_8.
- Replace all occurrences of `ct1` with `ct3`.
- Replace all occurrences of `shk1` with `shk3`.
- In the third subhybrid replace C_2 with C_7

Game 3 of **Case 3.5** is special in that it can be proven secure in two slightly different ways by slightly modifying either the proof for **Case 1** or the proof for **Case 2**. For the sake of brevity we will only explain the first way: Take the proof for **Game 5** of **Case 1** and only modify **Game 5a** be removing the second case. After that, the entire argument works analogously.

Other than that only the following inconsequential changes are required:

- Replace the phrase "by **Game 4** and the definition" by "by the definition" in all subcases of **Case 3**.
- The reference to **Game 1** in **Case 1** must be replaced by a reference to **Game 2** in all other games.
- The reference to **Game 2** in **Case 1** must be replaced by a reference to **Game 1** in **Case 3.4** and by a reference to **Game 3** in all other games.
- The references to $\Pr(abort_{\texttt{accept}})$ must be replaced with $\Pr(break_2)$ in all sub-cases of case 3.
- The probabilities $\Pr(break_{5a})$, $\Pr(break_{5b})$ and $\Pr(break_{5c})$ must be replaced with $\Pr(break_{3a})$, $\Pr(break_{3b})$ and $\Pr(break_{3c})$ in all sub-cases of case 3.
- The games that follow our modified games must replace their references to $\Pr(break_5)/\Pr(break_3)$ by $\Pr(break_{5c})/\Pr(break_{3c})$, respectively.
- Replace all uses of g^{uv} with psk, g^y with `ct2`, g^{xy} with `shk2`, g^{uy} with `shk3` and g^z with `shk2`.

Game 3 is somewhat special in that both ephemeral keys are assumed to be uncorrupted. In the original version this meant that only the DDH-assumption was necessary, whereas our version is fine with an IND-CPA-secure KEM. We again follow the original proof as closely as possible:

In this game, we replace the value `shk2` computed in the test session π_i^s and its honest contributive keyshare session with a random element from the same keyspace. Note that since the initiator session and the responder session both get key confirmation messages that include derivations based on the encapsulated shared key, both know that the key was received by the other session without modification. We explicitly interact with an IND-CPA challenger, and replace the ephemeral $\texttt{epk}_i$ and `ct2` values sent in the InitiatorHello and ResponderHello messages with the challenge public-key and ciphertext from the IND-CPA challenger. We only require the encapsulated key in one computation (as opposed to three in the original proof): $C_7 \leftarrow \mathsf{KDF}(c_2, \texttt{shk2})$

Here we can replace `shk2` with the supposed shared key $\texttt{k}^*$ from the IND-CPA-challenger. When the test bit sampled by the IND-CPA challenger is 0, then $\texttt{k}^*$ is the actually encapsulated shared key and we are in **Game 2**. When the test bit sampled by the IND-CPA challenger is 1, then $\texttt{k}^* \xleftarrow{\$} \mathcal{K}_{\mathsf{CPAKEM}}$ and we are in **Game 3**. Any adversary that can detect that change can be turned into an adversary against the IND-CPA problem and thus

$$\Pr(break_2) \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{CPAKEM},\mathcal{R}}(\lambda) + \Pr(break_3).$$

Finally, in **Game 3** of **Case 3.1** replace all occurrences of "prf " with "$\mathsf{prf}^{\mathsf{swap}}$ " during the entire hybrid.

After applying all these changes we can compute the complete adversarial advantage $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK}}_{\mathsf{pqWG},\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}},n_P,n_S,\mathcal{A}}(\lambda)$. As required by the security-definition, it is bounded by a polynomial factor of $\mathcal{A}$'s advantage in the dual-prf, IND-CCA, IND-CPA and auth-aead games. Specifically:

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK}}_{\mathsf{pqWG},\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}},n_P,n_S,\mathcal{A}}(\lambda) \\
&\leq n_P^2 n_S \left(\begin{array}{l} 2 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{R}}(\lambda) \;+\; 9 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{KDF},\mathcal{R}}(\lambda) \\ \quad + 4 \cdot \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{KDF},\mathcal{R}}(\lambda) \;+\; 2 \cdot \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{R}}(\lambda) \\ \qquad\qquad + 2 \cdot \frac{n_s}{2^\lambda} \\ + n_S \cdot \max \left\{ \begin{array}{l} \left(\begin{array}{l} \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{CPAKEM},\mathcal{R}}(\lambda) \\ + 4 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{KDF},\mathcal{R}}(\lambda) \\ + \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{KDF},\mathcal{R}}(\lambda) \end{array} \right), \\ \left(\begin{array}{l} \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{R}}(\lambda) \\ + 7 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{KDF},\mathcal{R}}(\lambda) \\ + 2 \cdot \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{KDF},\mathcal{R}}(\lambda) \end{array} \right), \\ \left(\begin{array}{l} \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{R}}(\lambda) \\ + 7 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{KDF},\mathcal{R}}(\lambda) \\ + \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{KDF},\mathcal{R}}(\lambda) \\ + \frac{n_s}{2^\lambda} \end{array} \right) \end{array} \right\} \end{array} \right) \\
&\leq n_P^2 n_S \left(\begin{array}{rcl} (7n_S + 9) & \cdot & \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{KDF},\mathcal{R}}(\lambda) \\ + \quad (2n_S + 4) & \cdot & \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{KDF},\mathcal{R}}(\lambda) \\ + \quad (n_S + 2) & \cdot & \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{R}}(\lambda) \\ + \quad n_S & \cdot & \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{CPAKEM},\mathcal{R}}(\lambda) \\ + \quad 2 & \cdot & \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{R}}(\lambda) \\ + \quad (n_S + 2) & \cdot & \frac{n_s}{2^\lambda} \end{array} \right)
\end{aligned}$$

(The last term is slightly less tight, but we include it here for the sake of simplicity.)

Overall the result is similar to that for WireGuard, except that we have a slight tightness-loss relative to the prf-security and replaced the pre-quantum assumptions with generic KEM-security assumptions.