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
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

***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 downgrade 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 resistance. The security considerations that lead to the design of WireGuard are layed 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 consists in transmitting the hash of a long-term public key instead of the public key itself; the quantum attack is prevented with this tweak if static public keys are not known to the attacker.

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 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

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

- 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. Consequently, 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.

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.

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]).

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

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 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 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 [43] 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 [44]. The other effort is PQCrypto-VPN by Easterbrook, Kane, LaMacchia, Shumow, and Zaverucha at Microsoft Research [45]. 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 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 properties WireGuard aims to achieve. Then we recall some cryptographic primitives used by WireGuard and PQ-WireGuard, and eventually provide a brief description of the WireGuard handshake protocol.

A. Security Properties

WireGuard was designed to achieve eCK-PFS-PSK security and a couple of additional properties. WireGuard considers a setting where an *initiator* I initiates a secure connection with a *responder* R. Using this notation, WireGuard aims to achieve the following security goals:

- **Session Key Secrecy**: The established session key is pseudorandom, i.e., it is 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.
- **Authenticity**: Both, initiator and responder, know who they are talking to. Specifically, it is infeasible for a party to impersonate another party.
- **Identity Hiding**: The identities of initiator and responder are only revealed to each other.
- **DoS protection**: The receiver can detect unauthorized connection-attempts early and abort the protocol before performing expensive computations.

These security goals should even be preserved under corruption of secrets. Towards the definition of different corruption models consider the following. 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 parties 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 corruption patterns. In general we consider **maximal exposure (MEX)** attacks [46, Sec. 3.3],[47],[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. E.g., if all secret data is corrupted, there is no way to protect against active adversaries. Below 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 **Forward Secrecy** (also known as pre-compromise security) as ephemeral secrets are deleted after use.

Session Key Uniqueness. The session keys will with overwhelming probability be unique for each execution of the handshake. This is true for all considered passive adversaries that only observe secrets of corrupted parties.

Authenticity. The handshake provides authenticity even under 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 any pre-shared secrets between Alice and Bob authenticity cannot be achieved. Identity is defined as knowledge of long-term secrets. In case of corruption of the long-term secret only a pre-shared key (and thereby the former authentication) could establish authenticity but also the pre-shared key is corrupted in this case.

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.

All other attacks against the authenticity, including **Key Compromise Impersonation Attacks** (KCI), are prevented.

Also related to authenticity are **Unknown Key Share Attacks** (UKS) in which an attacker tricks an honest party into believing that they are communicating with someone else than they actually do. WireGuard is designed to prevent all versions of these attacks.

Identity Hiding. WireGuard is designed to reveal no information about the identity of either the initiator or the responder as long as both long-term secrets and the initiators ephemeral secrets are uncompromised. Note that a compromise of a parties long-term secret is by definition also a reveal of its identity.

DoS Prevention. The receiver will abort early (before issuing a reply) on any connection-attempt where the initiator does not present knowledge of a valid combination of long term key and pre-shared secret.

B. Cryptographic building blocks

In the following we discuss cryptographic building blocks used in WireGuard and 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].

The use of DH is precisely what is vulnerable to Shor's algorithm [48], [49] 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) [50].

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). Instead the proof requires assumptions from the family of PRFODH-assumptions. These essentially combine the DDH-assumption with a prf assumption described below. Roughly they state that for some hash function H and message m, $H(g^{xy}, m)$ is indistinguishable from a random value even if the adversary has (limited) oracle access to $H(a^x, b)$ and $H(a^y, b)$, where he is allowed to choose a and b. The exact limitations on the oracle-access then vary depending on the exact version of the PRFODH-assumption that is used. For more details on both the PRFODH family and their use in the context of WireGuard we refer to [9].

Key-encapsulation mechanisms. A key-encapsulation mechanism (KEM) is a triple of algorithms (KEM.Gen, KEM.Enc,

[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 parties state, especially all used randomness. This turns out to be important when using KEMs.

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 it holds for all $(sk, pk) \leftarrow \mathsf{KEM.Enc}()$ that we get $\Pr[\mathsf{KEM.Dec}(c, sk) = k \mid (c, k) \leftarrow \mathsf{KEM.Enc}(pk, r)] = 1-\delta$. 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. SABER paper [23]. 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 actually used this way. For example, whenever one of the two parties is sending an *ephemeral* DH key to their peer. As a consequence, removing DH from existing protocols is usually not a trivial replacement by KEMs. In most cases it will require more substantial changes and WireGuard is no exception in that regard.

Pseudorandom Functions.

In order to keep the proof as similar as possible to that of the original WireGuard, we use the same prf-definition as Dowling and Paterson [9], except that we also require security against quantum adversaries:

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 QPT 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.

If a function satisfies both the prf-assumption and the $\mathsf{prf}^{\mathsf{swap}}$-assumption we say that it is a 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.

Authenticated Encryption with Associated Data. An authenticated encryption scheme with associated data (AEAD) is a triple of algorithms (AEAD.Gen, AEAD.Enc, AEAD.Dec). The key-generation algorithm AEAD.Gen returns a random key k that can then be used with the other two algorithms. The encryption algorithm AEAD.Enc takes a key k, a nonce n, a message m and associated data a and returns a ciphertext c. The decryption algorithm AEAD.Dec takes a key k, a ciphertext c and associated data a and returns either a message m or $\perp$.

For the purposes of our protocol we require that AEAD.Gen outputs just a random bit string instead of a key that has more structure. We note that this is not a real limitation as all practical AEAD schemes that we are aware of satisfy this property.

For a formal definition of the security properties of an AEAD scheme we refer to the seminal work of Phillip Rogaway [51]. Informally however, an AEAD scheme should provide confidentiality and authenticity. Confidentiality is defined in the sense of IND-CCA which in this case means that no efficient attacker can learn anything about the message m from a ciphertext c (except for its length), even when provided with encryption- and decryption-oracles (as long as it does not repeat nonces or query the decryption of c directly). Authenticity is defined as the inability of any efficient adversary to forge a valid ciphertext given access to the above en- and decryption oracles.

C. 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**

$(\mathtt{esk}_i, \mathtt{epk}_i) \leftarrow \mathsf{DH.Gen}()$
$\mathtt{sid}_i \xleftarrow{\$} \{0,1\}^{32}$
$\mathtt{ltk} \leftarrow \mathsf{AEAD.Enc}(\kappa_3, 0, \mathtt{spk}_i, H_3)$
$now \leftarrow \mathsf{Timestamp}()$
$\mathtt{time} \leftarrow \mathsf{AEAD.Enc}(\kappa_4, 0, H_4, now)$
$\mathtt{m1} \leftarrow \mathsf{MAC}(\mathsf{H}(\mathtt{lbl}_3 \,\|\, \mathtt{spk}_r), \mathtt{type} \,\|\, 0^3 \,\|\, \mathtt{sid}_i \,\|\, \mathtt{epk}_i \,\|\, \mathtt{ltk} \,\|\, \mathtt{time})$
$\mathtt{m2} \leftarrow \mathsf{MAC}(cookie, \mathtt{type} \,\|\, 0^3 \,\|\, \mathtt{sid}_i \,\|\, \mathtt{epk}_i \,\|\, \mathtt{ltk} \,\|\, \mathtt{time} \,\|\, \mathtt{m1})$
$\mathtt{InitHello} \leftarrow \mathtt{type} \,\|\, 0^3 \,\|\, \mathtt{sid}_i \,\|\, \mathtt{epk}_i \,\|\, \mathtt{ltk} \,\|\, \mathtt{time} \,\|\, \mathtt{m1} \,\|\, \mathtt{m2}$

$$\xrightarrow{\quad\mathtt{InitHello}\quad}$$

$(\mathtt{esk}_r, \mathtt{epk}_r) \leftarrow \mathsf{DH.Gen}()$
$\mathtt{sid}_r \xleftarrow{\$} \{0,1\}^{32}$
$\mathtt{zero} \leftarrow \mathsf{AEAD.Enc}(\kappa_9, 0, H_9, \emptyset)$
$\mathtt{m1} \leftarrow \mathsf{MAC}(\mathsf{H}(\mathtt{lbl}_3 \,\|\, \mathtt{spk}_i), \mathtt{type} \,\|\, 0^3 \,\|\, \mathtt{sid}_r \,\|\, \mathtt{sid}_i \,\|\, \mathtt{epk}_r \,\|\, \mathtt{zero})$
$\mathtt{m2} \leftarrow \mathsf{MAC}(cookie, \mathtt{type} \,\|\, 0^3 \,\|\, \mathtt{sid}_r \,\|\, \mathtt{sid}_i \,\|\, \mathtt{epk}_r \,\|\, \mathtt{zero} \,\|\, \mathtt{m1})$
$\mathtt{RespHello} \leftarrow \mathtt{type} \,\|\, 0^3 \,\|\, \mathtt{sid}_r \,\|\, \mathtt{sid}_i \,\|\, \mathtt{epk}_r \,\|\, \mathtt{zero} \,\|\, \mathtt{m1} \,\|\, \mathtt{m2}$

$$\xleftarrow{\quad\mathtt{RespHello}\quad}$$

$tk_i \leftarrow \mathsf{KDF}_1(C_9, \emptyset)$
$tk_r \leftarrow \mathsf{KDF}_2(C_9, \emptyset)$

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 are computed. This is layed 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].

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, we have to replace DH by KEMs for which well-established, efficient post-quantum instantiations exist. To discuss our KEM-based variant, first consider a simplified view on the core of the DH-based WireGuard handshake. 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:

Initiator | **Responder**

$(\mathtt{esk}_i, \mathtt{epk}_i) \leftarrow \mathsf{DH.Gen}()$

$$\xrightarrow{\quad\mathtt{epk}_i\quad}$$

$(\mathtt{esk}_r, \mathtt{epk}_r) \leftarrow \mathsf{DH.Gen}()$

$$\xleftarrow{\quad\mathtt{epk}_r\quad}$$

Initiator	Responder
$k_1 \leftarrow \mathsf{DH.Shared}(\mathtt{ssk}_i, \mathtt{spk}_r)$	$k_1 \leftarrow \mathsf{DH.Shared}(\mathtt{ssk}_r, \mathtt{spk}_i)$
$k_2 \leftarrow \mathsf{DH.Shared}(\mathtt{esk}_i, \mathtt{spk}_r)$	$k_2 \leftarrow \mathsf{DH.Shared}(\mathtt{ssk}_r, \mathtt{epk}_i)$
$k_3 \leftarrow \mathsf{DH.Shared}(\mathtt{ssk}_i, \mathtt{epk}_r)$	$k_3 \leftarrow \mathsf{DH.Shared}(\mathtt{esk}_r, \mathtt{spk}_i)$
$k_4 \leftarrow \mathsf{DH.Shared}(\mathtt{esk}_i, \mathtt{epk}_r)$	$k_4 \leftarrow \mathsf{DH.Shared}(\mathtt{esk}_r, \mathtt{epk}_i)$

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 are largely following their approach in our design. Towards our final proposal, let us first try to straightforwardly translate the DH picture to a KEM picture. The problem is, as described in Subsection II-B, 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. What we get as a naive KEM-based handshake without this static-static DH requires 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:

Initiator | **Responder**

$(\mathtt{esk}_i, \mathtt{epk}_i) \leftarrow \mathsf{CPAKEM.Gen}()$
$r_1 \xleftarrow{\$} \{0,1\}^\lambda, (c_1, k_1) \leftarrow \mathsf{CCAKEM.Enc}(\mathtt{spk}_r, r_1)$

$$\xrightarrow{\quad A_e, c_1\quad}$$

$r_2 \xleftarrow{\$} \{0,1\}^\lambda, (c_2, k_2) \leftarrow \mathsf{CCAKEM.Enc}(\mathtt{spk}_i, r_2)$
$r_3 \xleftarrow{\$} \{0,1\}^\lambda, (c_3, k_3) \leftarrow \mathsf{CPAKEM.Enc}(\mathtt{epk}_i, r_3)$

$$\xleftarrow{\quad c_2, c_3\quad}$$

$k_1 \leftarrow \mathsf{CCAKEM.Dec}(\mathtt{ssk}_r, c_1)$

$k_2 \leftarrow \mathsf{CCAKEM.Dec}(\mathtt{ssk}_i, c_2)$
$k_3 \leftarrow \mathsf{CPAKEM.Dec}(\mathtt{esk}_i, c_3)$

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 secret keys that have (ideally) been generated in a secure environment. However, in the above naive approach we do not obtain any security in this scenario. The reason is that the randomness used by $\mathsf{CCAKEM.Enc}$ is corrupted and consequently, an adversary can recompute the shared secret simply running $\mathsf{CCAKEM.Enc}$.
The general approach to address this issue is to securely combine ephemeral randomness r with some long-term

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

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.

secret σ before using it as protocol input. In [52] 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 $\mathsf{H}(\texttt{spk}_i \oplus \texttt{spk}_r)$. 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 already at this very early stage and consequently abort the handshake. This is not a security property in the cryptographic sense, but helps mitigate the effects of 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 $psk = \mathsf{H}(\texttt{spk}_i \oplus \texttt{spk}_r)$. Users who do not want to rely on this assumption need to set psk to an actual secret shared key that is agreed on out-of-band to achieve the same level of DoS protection as in WireGuard.

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 $KDF_1(\sigma_r, r_r)$ 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 $\mathsf{H}(\texttt{spk}_i \oplus \texttt{spk}_r)$ as default value for psk.
- Instead of feeding $\texttt{spk}_i$ into AEAD.Enc in step 5, we use $\mathsf{H}(\texttt{spk}_i)$. 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

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

Initiator | **Responder**

1: $(\texttt{esk}_i, \texttt{epk}_i) \leftarrow \mathsf{CPAKEM.Gen}()$
2: $\texttt{sid}_i \xleftarrow{\$} \{0,1\}^{32}$
3: $r_i \leftarrow \{0,1\}^{\lambda}$
4: $(\texttt{ct1}, \texttt{shk1}) \leftarrow \mathsf{CCAKEM.Enc}(\texttt{spk}_r, \mathsf{KDF}_1(\sigma_i, r_i))$
5: $\texttt{ltk} \leftarrow \mathsf{AEAD.Enc}(\kappa_3, 0, \mathsf{H}(\texttt{spk}_i), H_3)$
6: $now \leftarrow \mathsf{Timestamp}()$
7: $\texttt{time} \leftarrow \mathsf{AEAD.Enc}(\kappa_4, 0, H_4, now)$
8: $\texttt{m1} \leftarrow \mathsf{MAC}(\mathsf{H}(\texttt{lbl}_3 \,\|\, \texttt{spk}_r), \texttt{type} \,\|\, 0^3 \,\|\, \texttt{sid}_i \,\|\, \texttt{epk}_i \,\|\, \texttt{ct1} \,\|\, \texttt{ltk} \,\|\, \texttt{time})$
9: $\texttt{m2} \leftarrow \mathsf{MAC}(cookie, \texttt{type} \,\|\, 0^3 \,\|\, \texttt{sid}_i \,\|\, \texttt{epk}_i \,\|\, \texttt{ct1} \,\|\, \texttt{ltk} \,\|\, \texttt{time} \,\|\, \texttt{m1})$
10: $\texttt{InitHello} \leftarrow \texttt{type} \,\|\, 0^3 \,\|\, \texttt{sid}_i \,\|\, \texttt{epk}_i \,\|\, \texttt{ct1} \,\|\, \texttt{ltk} \,\|\, \texttt{time} \,\|\, \texttt{m1} \,\|\, \texttt{m2}$

$\xrightarrow{\texttt{InitHello}}$

11: $(\texttt{ct2}, \texttt{shk2}) \leftarrow \mathsf{CPAKEM.Enc}(\texttt{epk}_i)$
12: $r_r \leftarrow \{0,1\}^{\lambda}$
13: $(\texttt{ct3}, \texttt{shk3}) \leftarrow \mathsf{CCAKEM.Enc}(\texttt{spk}_i, \mathsf{KDF}_1(\sigma_r, r_r))$
14: $\texttt{sid}_r \xleftarrow{\$} \{0,1\}^{32}$
15: $\texttt{zero} \leftarrow \mathsf{AEAD.Enc}(\kappa_9, 0, H_9, \emptyset)$
16: $\texttt{m1} \leftarrow \mathsf{MAC}(\mathsf{H}(\texttt{lbl}_3 \,\|\, \texttt{spk}_i), \texttt{type} \,\|\, 0^3 \,\|\, \texttt{sid}_r \,\|\, \texttt{sid}_i \,\|\, \texttt{ct2} \,\|\, \texttt{ct3} \,\|\, \texttt{zero})$
17: $\texttt{m2} \leftarrow \mathsf{MAC}(cookie, \texttt{type} \,\|\, 0^3 \,\|\, \texttt{sid}_r \,\|\, \texttt{sid}_i \,\|\, \texttt{ct2} \,\|\, \texttt{ct3} \,\|\, \texttt{zero} \,\|\, \texttt{m1})$
18: $\texttt{RespHello} \leftarrow \texttt{type} \,\|\, 0^3 \,\|\, \texttt{sid}_r \,\|\, \texttt{sid}_i \,\|\, \texttt{ct2} \,\|\, \texttt{ct3} \,\|\, \texttt{zero} \,\|\, \texttt{m1} \,\|\, \texttt{m2}$

$\xleftarrow{\texttt{RespHello}}$

19: $tk_i \leftarrow \mathsf{KDF}_1(C_9, \emptyset)$
20: $tk_r \leftarrow \mathsf{KDF}_2(C_9, \emptyset)$

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	—	—	$\mathsf{H}(H_9 \parallel \mathtt{zero})$

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.

hash of this public key.

IV. SECURITY ANALYSIS

We provide two proofs of security for PQ-WireGuard: one in the computational and one in the symbolic model. Thereby we establish the same setting as for WireGuard. In the following we outline both proofs. In the computational model we prove that the PQ-WireGuard handshake, like the WireGuard handshake, achieves so called eCK-PFS-PSK-security. While certainly on the stronger end of security notions for authenticated key-exchange, eCK-PFS-PSK only guarantees that the exchanged key is indistinguishable from a random bit string. Further notions that PQ-WireGuard also targets, such as anonymity and DoS-protection, 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 lead to 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 step is to replace proof steps (i.e., game-hops) making use of either the PRFODH- or the DDH-assumptions with generic KEM-security- and prf-assumptions. Most of these changes are straightforward and readers who are familiar with the original proofs 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. Then, 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 an HKDF, 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 either IND-CPA- or IND-CCA-security of the respective KEM.

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 the Diffie-Hellman. As a consequence we had to add multiple new symmetric game hops, particularly around most asymmetric hybrids.

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 the used hash function to be a prf. This turns out to be too weak. The hash is used to combine two inputs. While in different corruption settings there is always one input that is pseudorandom, 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 hash function 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 $\mathcal{A}$:

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

where n_P is the number of parties and n_S is the number of sessions. 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 the eCK-PFS-PSK still promises in this case that the established key remains confidential. (Authenticity is trivially broken however.)

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 the eCK-PFS-PSK-security still promises both authenticity and confidentiality, though without forward secrecy.

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 consequence of this is that there is less need for us to provide a hybrid version of PQ-WireGuard and WireGuard than there would normally be, allowing for a more efficient protocol.

B. The Symbolic Proof

The symbolic proof of PQ-WireGuard uses the Tamarin prover [53], building on the symbolic proof for WireGuard. 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 motivate the use of Tamarin to model several cryptographic protocols, including TLS 1.3 [54] and the 5G protocol[55]. In Appendix C we give a brief introduction to Tamarin. A full tutorial can be found in the Tamarin manual [56].

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 attack resistance, both claimed by WireGuard, which we add. 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.

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)
        ci0   = h(cii, pekI, '1')
        hi0   = h(hii, pekI)
        sct   = aenc{kb}pkR
        ci1   = h(ci0, kb, '1')
        ki1   = h(ci0, kb, '2')
        astat = aead(ki1, <h(pkI), ~pkISurrogate>, hi0)
        hi1   = h(hi0, astat)
        ci2   = h(ci1, ~psk, '1')
        ki2   = h(ci1, ~psk, '2')
        ats   = aead(ki2, <$ts, 'TAI64N'>, hi1)
        hi2   = h(hi1, 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. Appendix D provides the results for the remaining properties.

The full Tamarin proof is part of the supplementary material of this paper.

Replay Attack. 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. Only accepting a particular timestamp once is a relaxation of the actual protection mechanism in the implementation, as the responder would reject an initiation message with a timestamp that is less or equal to the last known timestamp from a particular initiator, while in our model a responder only rejects an initiation message if the timestamp is equal to the last known timestamp. Note that this restriction already prevents an adversary from replaying an initiation message as a whole.

Consequently, we 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, and
        & ISend(<pki, pkr, peki, psk, kb>) @ i1 & #i1 < #i
        & PRFGen(tpk, r, kb) @ i1
        // R receives later another init msg containing the same secret kb
        & RKeys(<pki, pkr, peki2, psk2, cr2, kb, ka2, k2>) @ j & #i < #j
        // with a different timestamp
        & OnlyOnce(pki, pkr, ts2) @ j & not(ts = ts2)
    ==> // then the adversary 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(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 Attack. 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 processed. With lemma *dos_resistance* we prove that an adversary must compromise the pre-shared key in order to launch a DoS attack on a victim.

```
lemma dos_resistance:
    "All pki pkr peki psk cr kb ka k #i.
        // if R accepts an init msg containing secret kb
        RKeys(<pki, pkr, peki, psk, cr, kb, ka, k>) @ i
        // but the init msg was crafted by the adversary,
        // either by encapsulating kb herself or by
        // using a ciphertext created by
        // an honest initiator that contains kb
        & not(Ex pki pkr peki1 psk1 #j.
                ISend(<pki, pkr, peki1, psk1, kb>) @ j & #j < #i)
    ==> // then the PSK was compromised (or not in use)
        Ex #j. Reveal_PSK(psk) @ j & #j < #i"
```

To be more specific, we prove that a responder R will not process an initiation message from the claimed initiator I unless the initiation message was indeed sent by I or the pre-shared key between R and I has been compromised.

In summary, our Tamarin model shows that all the security properties of WireGuard are satisfied except for DoS-attack resistance, which relies on static-static Diffie-Hellman results. We show that by mixing an optional pre-shared symmetric key into the initiation message, one can achieve the same level of DoS resistance as WireGuard.

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 [57, 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);
- 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 both the initiator's and the responder's handshake messages into one IPv6 packet. 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, $\mathtt{sid}_i$, `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, $\mathtt{sid}_i$, $\mathtt{sid}_r$, `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 [58], SIKE [59], and ROLLO-I [60]. 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 [61], 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.

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

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 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 [62]). 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 [45] based on OpenVPN and the Open Quantum Safe (OQS) framework [62]. Note that PQCrypto-VPN has optional post-quantum authentication using the Picnic signature scheme [63], [64]; 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 [65] or Frodo-752 as described in [66].

Our implementation of the PQ-WireGuard software is based on the original WireGuard implementation. For Classic McEliece we use the "avx" software targeting recent 64-bit Intel processors, which has been submitted to SUPERCOP [67] 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 when the server receives an initiation packet from the client and starts any computation for it and when session keys are derived on the server side.

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.

VPN Software	Packet Number	Traffic (bytes)	Client Time (milliseconds)	Server Time (milliseconds)
WireGuard	2 (0)	324 (0)	0.606 (0.572)	0.187 (0.005)
PQ-WireGuard (this paper)	2 (0)	2492 (0)	0.924 (0.573)	0.296 (0.027)
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 III
RESOURCES REQUIREMENT FOR HANDSHAKE PROTOCOL OVER IPv4, NUMBERS IN PARENTHESES ARE STANDARD DEVIATION

We see that both WireGuard and PQ-WireGuard only require 2 packets, which is optimal for an authenticated key exchange. We also see that in PQ-WireGuard, the total time required for the handshake increases by less than 60% compared to WireGuard, at least when it is run over a high-speed network link as in our experiments. Similarly, the time required

VPN Software	Packet Number	Traffic (bytes)	Client Time (milliseconds)	Server Time (milliseconds)
WireGuard	2 (0)	364 (0)	0.580 (0.628)	0.184 (0.005)
PQ-WireGuard (this paper)	2 (0)	2532 (0)	0.917 (0.544)	0.295 (0.026)
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 [66])	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)

TABLE IV
RESOURCES REQUIREMENT FOR HANDSHAKE PROTOCOL OVER IPv6, NUMBERS IN PARENTHESES ARE STANDARD DEVIATION

for server-side computations increases by a little less than 60% 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 more than 5 times faster than the handshake of IPsec and more than three orders of magnitude faster than the handshake of any variant of OpenVPN,

ACKNOWLEDGEMENTS

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

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, https://www.ndss-symposium.org/wp-content/uploads/2017/09/ndss2017_04A-3_Donenfeld_paper.pdf. 1, 6

[2] T. Perrin, "Noise protocol framework," https://noiseprotocol.org/noise.pdf (accessed 2019-10-22). 1, 2

[3] D. J. Bernstein, "Curve25519: new Diffie-Hellman speed records," in *Public Key Cryptography – PKC 2006*, ser. LNCS, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., vol. 3958. Springer, 2006, pp. 207–228, http://cr.yp.to/papers.html#curve25519. 1, 4

[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, M. Jacobson, M. Locasto, P. Mohassel, and R. Safavi-Naini, Eds., vol. 7954. Springer, 2013, pp. 119–135, . 1

[5] D. J. Bernstein, "The Poly1305-AES message-authentication code," in *Fast Software Encryption*, ser. LNCS, H. Gilbert and H. Handschuh, Eds., vol. 3557. Springer, 2005, pp. 32–49, http://cr.yp.to/papers.html#poly1305. 1

[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. 1

[7] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF protocols," IETF RFC 8439, 2018, https://tools.ietf.org/pdf/rfc8200.pdf. 1

[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. 1, 2, 9

[9] B. Dowling and K. G. Paterson, "A cryptographic analysis of the WireGuard protocol," in *Applied Cryptography and Network Security.*, ser. LNCS, B. Preneel and F. Vercauteren, Eds., vol. 10892. Springer, 2018, https://eprint.iacr.org/2018/080. 1, 2, 4, 5, 6, 8, 15, 23

[10] "BoringTun," https://github.com/cloudflare/boringtun. 1

[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. 1

[12] J. Appelbaum, C. Martindale, and P. Wu, "Tiny WireGuard tweak," in *Progress in Cryptology – AFRICACRYPT 2019*, ser. LNCS, J. Buchmann, A. Nitaj, and T. Rachidi, Eds., vol. 11627. Springer, 2019, pp. 3–20, https://eprint.iacr.org/2019/482. 1, 7

[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, T. Peyrin and S. Galbraith, Eds., vol. 11274. Springer, 2018, pp. 395–427, https://csidh.isogeny.org/csidh-20181118.pdf. 2

[14] X. Bonnetain and A. Schrottenloher, "Submerging CSIDH," Cryptology ePrint Archive, Report 2018/537, 2018, https://eprint.iacr.org/2018/537. 2

[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, Y. Ishai and V. Rijmen, Eds., vol. 11477. Springer, 2019, pp. 409–441, https://eprint.iacr.org/2018/1059. 2

[16] C. Peikert, "He gives C-sieves on the CSIDH," Cryptology ePrint Archive, Report 2019/725, 2019, https://eprint.iacr.org/2019/725. 2

[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. 2

[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, M. Fischlin, J. Buchmann, and M. Manulis, Eds. Springer, 2012, pp. 467–484, https://eprint.iacr.org/2012/211. 2, 3, 4, 6

[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. 2

[20] S. Deering and R. Hinden, "Internet protocol, version 6 (IPv6) specification," IETF RFC 8200, 2017, https://tools.ietf.org/pdf/rfc8200.pdf. 2, 11

[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, https://sike.org/files/SIDH-spec.pdf. 2

[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, https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-2/submissions/Classic-McEliece-Round2.zip. 2, 10

[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, A. Joux, A. Nitaj, and T. Rachidi, Eds., vol. 10831. Springer, 2018, pp. 282–305, https://eprint.iacr.org/2018/230. 2, 5, 10

[24] ——, "SABER: Mod-LWR based KEM (round 2 submission)," Round-2 submission to the NIST PQC project, 2019, https://www.esat.kuleuven.be/cosic/pqcrypto/saber/resources.html. 2, 10, 11, 14

[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. 2

[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. 2

[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. 2

[28] T. Perrin, "KEM-based hybrid forward secrecy for Noise," 2018, https://github.com/noiseprotocol/noise_hfs_spec/blob/master/output/noise_hfs.pdf. 3

[29] C. Boyd, Y. Cliff, J. Gonzalez Nieto, and K. G. Paterson, "Efficient one-round key exchange in the standard model," in *Information Security and Privacy*, Y. Mu, W. Susilo, and J. Seberry, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 69–83. 3

[30] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *Advances in Cryptology - CRYPTO '99*, ser. LNCS, M. Wiener, Ed., vol. 1666. Springer, 1999, pp. 537–554, http://antoanthongtin.vn/Portals/0/UploadImages/kiennt2/KyYeu/DuLieuNuocNgoai/4.Advances%20in%20cryptology-Crypto%201999-LNCS%201666/16660537.pdf. 3, 11

[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. 3

[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, T. Peyrin and S. Galbraith, Eds., vol. 11274. Springer, 2018, pp. 158–189, https://eprint.iacr.org/2018/817. 3

[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, E. Oswald and M. Fischlin, Eds., vol. 9057. Springer, 2015, pp. 719–751, https://eprint.iacr.org/2014/589/. 3

[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, https://eprint.iacr.org/2017/634. 3

[35] P. Longa, "A note on post-quantum authenticated key exchange from supersingular isogenies," Cryptology ePrint Archive, Report 2018/267, 2018, https://eprint.iacr.org/2018/267. 3

[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, S. D. Galbraith and S. Moriai, Eds., vol. 11921. Springer, 2019, pp. 278–308, https://eprint.iacr.org/2018/760. 3

[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, K. Lee, Ed., vol. 11396. Springer, 2019, pp. 177–195, https://eprint.iacr.org/2018/730.pdf. 3

[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, https://prosecco.gforge.inria.fr/personal/bblanche/publications/LippBlanchetBhargavanEuroSP19.html. 3

[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. 3

[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, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Springer, 2012, pp. 273–293, https://eprint.iacr.org/2011/219. 3

[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, https://eprint.iacr.org/2018/766. 3

[42] B. Blanchet, "ProVerif: Cryptographic protocol verifier in the formal model," https://prosecco.gforge.inria.fr/personal/bblanche/proverif/ (accessed 2019-10-21). 3

[43] O. Inc., "VPN software solutions & services for business | OpenVPN," 2019, https://openvpn.net/ (accessed 2019-10-21). 3

[44] 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. 3

[45] 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/. 3, 11

[46] 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. 4

[47] 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, H. Kim, Ed., vol. 7259. Springer, 2011, pp. 490–509. 4

[48] 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, http://www-math.mit.edu/~shor/papers/algsfqc-dlf.pdf. 4

[49] ——, "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. 4

[50] E. S. V. Freire, D. Hofheinz, E. Kiltz, and K. G. Paterson, "Non-interactive key exchange," in *Public-Key Cryptography – PKC 2013*, ser. LNCS, K. Kurosawa and G. Hanaoka, Eds., vol. 7778. Springer, 2013, pp. 254–271, https://eprint.iacr.org/2012/732. 4

[51] P. Rogaway, "Authenticated-encryption with associated-data," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS '02. New York, NY, USA: ACM, 2002, pp. 98–107. [Online]. Available: http://doi.acm.org/10.1145/586110.586125 5

[52] 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, https://eprint.iacr.org/2012/211. 7

[53] 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, N. Sharygina and H. Veith, Eds., vol. 8044. Springer, 2013, pp. 696–701, http://www-oldurls.inf.ethz.ch/personal/basin/pubs/cav13.pdf. 9

[54] 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, https://people.cispa.io/cas.cremers/downloads/papers/CHHSV2017-TLS13.pdf. 9

[55] 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, https://people.inf.ethz.ch/rsasse/pub/5G-CCS18.pdf. 9

[56] T. T. Team, "Tamarin-prover manual," https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf. 9

[57] 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. 10

[58] 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, https://round5.org/#spec. 11

[59] 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, https://sike.org/#resources. 11

[60] 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. 11

[61] 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. 11

[62] M. Mosca and D. Stebila, "Open quantum safe," 2020, https://openquantumsafe.org/ (accessed 2020-01-30). 11

[63] 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, https://eprint.iacr.org/2017/279. 11

[64] 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, version 2.0, https://github.com/microsoft/Picnic/blob/master/spec/design-v2.0.pdf. 11

[65] C. Costello, P. Longa, and M. Naehrig, "Efficient algorithms for supersingular isogeny diffie-hellman," in *Advances in Cryptology – CRYPTO 2016*. 11

[66] 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*, ser. CCS '16, 2016. 11, 12

[67] D. J. Bernstein and T. Lange, "eBACS: ECRYPT benchmarking of cryptographic systems," http://bench.cr.yp.to (accessed 2020-01-22). 12

[68] C. J. F. Cremers and M. Feltz, "Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal," in *Computer Security – ESORICS 2012*, ser. LNCS, S. Foresti, M. Yung, and F. Martinelli, Eds., vol. 7459. Springer, 2012, pp. 734–751, https://eprint.iacr.org/2012/416. 20, 23

[69] B. A. LaMacchia, K. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," in *Provable Security*, ser. LNCS, W. Susilo, J. K. Liu, and Y. Mu, Eds., vol. 4784. Springer, 2007, pp. 1–16, https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/strongake-submitted.pdf. 20

[70] J. Donenfeld, "WireGuard: Next generation kernel network tunnel," in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA*, 2017. [Online]. Available: https://www.internetsociety.org/doc/wireguard-next-generation-kernel-network-tunnel 20

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 $(pk := (seed_{\mathbf{A}}, \mathbf{b}), sk := \mathbf{s})$

B. Computational Proof

We prove that the PQ-WireGuard handshake protocol (from now on abbreviated pqWG) is eCK-PFS-PSK-secure with cleanness predicate $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ (capturing perfect forward secrecy and resilience to KCI attacks). That is,

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 $K' = \mathcal{H}(\hat{K}')$

for any QPT algorithm $\mathcal{A}$ against the eCK-PFS-PSK key-indistinguishability game $\mathsf{Adv}^{\text{eCK-PFS-PSK}}_{\text{pqWG,clean}_{\text{eCK-PFS-PSK}},n_S,n_P,\mathcal{A}}(\lambda)$ 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 by the replacement of our primitives 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 paper.

Given the extreme similarity between the cases that relied on the sym-ms-PRFODH assumption in the original paper we will start by providing a detailed proof for **Game 5** of **Case 1** and then discuss the changes necessary for the other cases afterwards. To highlight the similarity to the Dowling-Paterson proof, we will follow their proof verbatim whenever possible and only deviate where necessary.

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 parties 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 $\mathtt{k}^*$ (**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 $\hat{r} := \mathsf{HKDF}(\sigma_i, r_i)$ 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 $\hat{r} \leftarrow \mathsf{HKDF}(\sigma_i, r_i)$ and we are in **Game 4**. If the test bit sampled by the $\mathsf{prf}^{\mathsf{swap}}$ challenger is 1, then $\hat{r} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ 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 $\mathsf{HKDF}(\sigma_i, r_i)$ 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 $\frac{n_S}{2^{|r_i|}}$ (which when multiplied by the number of sessions results in the famous approximation of the birthday-bound $\frac{n_S^2}{2^{|r_i|}}$).

Given that, we initialize a prf challenger and replace all computations of $\mathsf{HKDF}(\sigma_i, \cdot)$ 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 $\hat{r} \leftarrow \mathsf{HKDF}(\sigma_i, r_i)$ and we are in **Game 4**. If the test bit sampled by the prf challenger is 1, then $\hat{r} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ 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 HKDF, and we find:

$$\Pr(abort_{\mathsf{accept}}) \\ \leq \frac{n_S}{2^{|r_i|}} + \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_{5a})$$

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

(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:

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

In **Game 5c** we replace the values of C_3, κ_3 with uniformly random and independent values $\widetilde{C_3}, \widetilde{\kappa_3} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ (where $\{0,1\}^{|\mathsf{HKDF}|}$ is the output space of the HKDF) 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 $\widetilde{C_3}, \widetilde{\kappa_3} \leftarrow \mathsf{HKDF}(C_2, \mathtt{shk1})$ and we are in **Game 5b**. If the test bit sampled by the $\mathsf{prf}^{\mathsf{swap}}$ challenger is 1, then $\widetilde{C_3}, \widetilde{\kappa_3} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ 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 HKDF, and we find:

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

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 in the manner 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 all occurrences of HKDF with KDF_1.
- 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 analogous.

Other than that only the following inconsequential changes are required:

- The phrase "by **Game 4** and the definition" must be replaced with "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 $\Pr(abort_{\mathtt{accept}})$ 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 $\mathtt{ct2}$, g^{xy} with $\mathtt{shk2}$, g^{uy} with $\mathtt{shk3}$ and g^z with $\mathtt{shk2}$.

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 $\mathtt{shk2}$ 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 $\mathtt{epk}_i$ and $\mathtt{ct2}$ 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):

- $C_7 \leftarrow \mathsf{HKDF}(c_2, \mathtt{shk2})$

Here we can replace $\mathtt{shk2}$ with the supposed shared key $\mathtt{k}^*$ from the IND-CPA-challenger. When the test bit sampled by the IND-CPA challenger is 0, then $\mathtt{k}^*$ 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 $\mathtt{k}^* \xleftarrow{\$} \mathcal{K}_{\mathsf{CPAKEM}}$ 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

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

Finally, in **Game 3** of **Case 3.1** replace all occurrences of "prf " with "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{A}}(\lambda) + 9\cdot\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\
\quad + 4\cdot\mathsf{Adv}^{\mathsf{prf^{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + 2\cdot\mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda) \\
\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{A}}(\lambda)\\ + 4\cdot\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda)\\ + \mathsf{Adv}^{\mathsf{prf^{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda)\end{array}\right),\\
\left(\begin{array}{l}\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda)\\ + 7\cdot\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda)\\ + 2\cdot\mathsf{Adv}^{\mathsf{prf^{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda)\end{array}\right),\\
\left(\begin{array}{l}\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda)\\ + 7\cdot\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda)\\ + \mathsf{Adv}^{\mathsf{prf^{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda)\\ + \frac{n_s}{2^\lambda}\end{array}\right)
\end{array}\right\}
\end{array}\right) \\
&\leq n_P^2 n_S \left(\begin{array}{rrcl}
 & (7n_S+9) & \cdot & \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\
+ & (2n_S+4) & \cdot & \mathsf{Adv}^{\mathsf{prf^{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\
+ & (n_S+2) & \cdot & \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda) \\
+ & n_S & \cdot & \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{CPAKEM},\mathcal{A}}(\lambda) \\
+ & 2 & \cdot & \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda) \\
+ & (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.

C. The Tamarin Prover.

Tamarin operates based on multiset rewriting. A Tamarin model is in essence a state machine whose state is a multiset of *facts*. The transition between states are defined by *rules*. Rules define the behavior of honest parties as well as the ability of an adversary. A Tamarin rule has a left-hand side (*premise*) and a right-hand side (*conclusion*) separated by the arrow symbol. To apply a rule, facts in its premise must exist in the current state. After the rule is applied, the facts in its premise are *consumed* and the facts in its conclusion are *produced*. For example,

```
rule Reveal_ltk:
  [ !Ltk(A, ltk) ]--[ LtkReveal(A) ]->[ Out(ltk) ]
```

defines a rule *Reveal_ltk* with which an adversary can compromise the long term key of a party A. To apply this rule, the fact !Ltk(A, ltk) must exist in the current state, and after the rule is applied, the fact Out(ltk) is added to the state. A fact can be *linear* or *persistent*, which decides how the fact can be consumed. In this case the fact Ltk(A, ltk) is prepended by an exclamation mark, which marks it as a persistent fact. A persistent fact can be consumed multiple times and will stay in the state, while a linear fact can only be consumed once and will disappear from the state afterwards. A rule may also produce one or more *action facts*, which are facts included between the arrow symbol of its definition. Action facts are not added into the state; instead, they are recorded and used to prove security properties, as we will discuss later. In this example LtkReveal(A) is the action fact.

To model cryptographic protocols, one has to model cryptographic primitives, e.g. hashing and symmetric encryption. In Tamarin, cryptographic primitives are modeled using functions and equations. For example, an AEAD scheme can be modeled with two ternary functions, one binary function and two equations:

```
functions: aead/3, decrypt/2, verify/3, true/0
equations: decrypt(aead(key, text, tag), key) = text
equations: verify(aead(key, text, tag), tag, key) = true
```

Some commonly used primitives are defined as *built-ins* in Tamarin, e.g. Diffie-Hellman group operations, symmetric and asymmetric encryption, digital signatures, and hashing. Since Tamarin is a tool for symbolic analysis, those cryptographic primitives are assumed to be *perfect*. In other words,

- Encryption reveals nothing about the plaintext,
- Signatures are not forgeable,
- Hash functions are random oracles with zero collision probability, and
- Random values are truly random and never repeat.

Tamarin also allows to define *restrictions* to restrict the possible state transitions in the protocol analysis. This can be useful for instance to model the verification of signatures. E.g.

```
restriction Equality:
  "All x y #i. Eq(x, y) @ #i ==> x = y"

rule B_receive_message:
  [ !Ltk(B, ltkB), !Pk(A, pkA), In(<m, sig>) ]
  --[ Eq(verify(sig, m, pkA), true)]->
  [ St_B_1(B, ltkB, pkA, A, m) ]
```

defines a restriction *Equality* which specifies that party B must verify the signature sig with the incoming message m. With this restriction all the execution flows where the signature is not valid are ignored.

To prove a security property it has to be formalized as a *lemma*, which is a first-order logic formula. The action facts produced by the rules are used as building blocks of lemmas. For example,

```
lemma client_session_key_honest_setup: exists-trace
  "Ex S k #i. SessKey(S, k) @ #i &
   not(Ex #r. LtkReveal(S) @ #r)"
```

defines a security property which claims that an honest peer S can establish a session key k at timestamp i if the long term key of S is never compromised. Tamarin will either prove the lemma to be true or falsify the claim by raising a counterexample.

D. Remaining proofs in symbolic model

Session Key Secrecy. In WireGuard and PQ-WireGuard, an initiator does not send the first data packet, which is encrypted with the derived session key, until she receives a valid response message from the receiver. The ciphertext of encrypting an empty message ($zero$ in Algorithm 2) is used to authenticate the identity of the responder to the initiator.

Similarly, a responder does not send any encrypted data until she received the first data packet encrypted with the correct session key from the initiator. Consequently, an adversary must wait until a handshake is completed, either by impersonating an honest party or by passively observing a handshake between two honest parties in order to be able to intercept any encrypted packets that contain actual data.

We formally model and analyze the session key secrecy with three Tamarin lemmas.

```
lemma key_init_secrecy[reuse]:
    "All pki pkr peki psk ck kb ka k tpk r #i #j #i1.
        // If I thinks the handshake is done and
        // she can send the first data
        IKeys(<pki, pkr, peki, psk, ck, kb, ka, k>) @ i
        & PRFGen(tpk, r, kb) @ i1
        // and adversary knows the session key
        & K(ck) @ j
    ==> // then the PSK is compromised (or not in use)
        (Ex #j1. Reveal_PSK(psk) @ j1)
        // and R's static key is compromised
        & ((Ex #j1. Reveal_AK(pkr) @ j1)
            // or both I's RNG and PRF key are compromised
            | ((Ex #j1. Reveal_rnd(r) @ j1)
               & (Ex #j1. Reveal_prfk(tpk) @ j1))
        )"

lemma key_resp_secrecy[reuse]:
    "All pki pkr peki psk ck kb ka k tpk r #i #j #i1.
        // If R thinks the key is confirmed and
        // she can accept the first data
        RConfirm(<pki, pkr, peki, psk, ck, kb, ka, k>) @ i
        & PRFGen(tpk, r, ka) @ i1 & #i1 < #i
        // and adversary knows the session key
        & K(ck) @ j
    ==> // then the PSK was compromised (or not in use)
        (Ex #j1. Reveal_PSK(psk) @ j1)
        // and I's static key is compromised
        & ((Ex #j1. Reveal_AK(pki) @ j1)
        // or both R's RNG and PRG key are compromised
            | ((Ex #j1. Reveal_rnd(r) @ j1)
               & (Ex #j1. Reveal_prfk(tpk) @ j1))
        )"

/* This includes forward secrecy */
lemma key_secrecy:
    "All pki pkr peki psk ck kb ka k tpk1 tpk2 r1 r2 r3 #i #i2 #i3 #i4.
        // If I and R agree on keys
        IKeys(<pki, pkr, peki, psk, ck, kb, ka, k>) @ i
        & PRFGen(tpk1, r1, kb) @ i4
        & RKeys(<pki, pkr, peki, psk, ck, kb, ka, k>) @ i2
        & PRFGen(tpk2, r2, ka) @ i2 & PRFGen(tpk2, r3, k) @ i2
        // and adversary knows the session key
        & K(ck) @ i3
    ==> // then the PSK was compromised (or not in use)
        // and all 3 secrets are compromised.
        (Ex #j. Reveal_PSK(psk) @ j) & (
            ( (Ex #j1. Reveal_AK(pki) @ j1)
                | ((Ex #j1. Reveal_rnd(r2) @ j1)
                   & (Ex #j1. Reveal_prfk(tpk2) @ j1))
            )
            & ( (Ex #j1. Reveal_AK(pkr) @ j1)
                | ((Ex #j1. Reveal_rnd(r1) @ j1)
                   & (Ex #j1. Reveal_prfk(tpk1) @ j1))
            )
            & ( (Ex #j1. Reveal_EphK(peki) @ j1)
                | ((Ex #j1. Reveal_rnd(r3) @ j1)
                   & (Ex #j1. Reveal_prfk(tpk2) @ j1))
            )
        )"
```

In the case of a passive attack, we prove that if Alice and Bob established a session and the adversary knows the session keys (tk_i, tk_r), then the adversary must have compromised all the encapsulated secrets $(shk1, shk2, shk3)$. This can be achieved by compromising the static private keys of both Alice and Bob, as well as the ephemeral private key. Alternatively, the adversary can learn the encapsulated secrets by compromising the random number generators of both parties and their PRF secrets. Note that the adversary may not need to compromise the static private keys or the random number generators of both parties at the same time; any combination of the two approaches would be sufficient.

In the case of an active attack, we prove that the adversary must compromise the secret encapsulated with the other party's static public key ($shk1$ or $shk3$) in order to complete the handshake, thereby learning the session keys. For the initiator, either her static private key is compromised, or the random number generator of the responder as well as her PRF secret are compromised. The same applies for the responder. Note that if either the initiator or the responder make use of the pre-shared key, then the adversary must also compromise the pre-shared key(s) to compromise the session keys.

Session Key Uniqueness. For a pair Alice and Bob, we prove with lemma *session_uniq* that the session keys (tk_i, tk_r) of any session between them will be unique with uncompromised random number generators, which follows from the fact that both Alice and Bob mix random nonces to derive the final session keys. To violate session key uniqueness between Alice and Bob, the adversary would have to compromise and manipulate the random number generators of both parties and their PRF secrets in order to enforce the same set of $(epk_i, shk1, ct2, shk2, shk3)$ on two different handshakes.

```
lemma session_uniq[reuse]:
    "(All pki pkr peki psk ck kb ka k #i.
        IKeys(<pki, pkr, peki, psk, ck, kb, ka, k>) @ i
    ==>
        not(Ex peki2 #j1.
            IKeys(<pki, pkr, peki2, psk, ck, kb, ka, k>) @ j1
            & not(#j1 = #i)
        )
    )
    &(All pki pkr peki psk ck kb ka k #i.
        RConfirm(<pki, pkr, peki, psk, ck, kb, ka, k>) @ i
    ==>
        not(Ex peki2 psk2 #j1.
            RConfirm(<pki, pkr, peki2, psk2, ck, kb, ka, k>) @ j1
            & not(#j1 = #i)
        )
    )"
```

On the other hand, an adversary may trick three honest parties into sharing the same session keys under extreme conditions, as we discuss below in UKS attack.

Key Compromise Impersonation Attacks. For PQ-WireGuard, an adversary cannot impersonate an arbitrary party to a party whose static private key is compromised as in TLS, because Diffie-Hellman key exchange is no longer used. We prove KCI in the two lemmas *KCI_on_initiator_resistance* and *KCI_on_responder_resistance* which cover both directions.

```
lemma KCI_on_initiator_resistance[reuse]:
    "All pki pkr peki psk ck kb ka k tpk r #i #i1.
        // If I believes she has completed a handshake with R
        IKeys(<pki, pkr, peki, psk, ck, kb, ka, k>) @ i
        & PRFGen(tpk, r, kb) @ i1 & #i1 < #i
        // but R doesn't have a matching session
        & not(Ex #j. #j < #i
        & RKeys(<pki, pkr, peki, psk, ck, kb, ka, k>) @ j)
    ==> // then the PSK was compromised (or not in use), and
        (Ex #j. Reveal_PSK(psk) @ j & #j < #i)
        & ( // either R's static key was compromised, or
            (Ex #j. Reveal_AK(pkr) @ j & #j < #i)
            // I's RNG and her PRF keys are both compromised
            | ((Ex #j. Reveal_rnd(r) @ j) & (Ex #j. Reveal_prfk(tpk) @ j))
        )"

lemma KCI_on_responder_resistance[reuse]:
    "All pki pkr peki psk ck kb ka k tpk r #i.
        // If R believes she has a confirmed session with I
        RConfirm(<pki, pkr, peki, psk, ck, kb, ka, k>) @ i
        & PRFGen(tpk, r, ka) @ #i
        // but I doesn't have a matching session
        & not(Ex #j. #j < #i
            & IKeys(<pki, pkr, peki, psk, ck, kb, ka, k>) @ j)
    ==> // then the PSK was compromised (or not in use), and
        (Ex #j. Reveal_PSK(psk) @ j & #j < #i)
        & ( // either I's static key was compromised, or
            (Ex #j. Reveal_AK(pki) @ j & #j < #i)
            // R's RNG and her PRF key are both compromised
            | ((Ex #j. Reveal_rnd(r) @ j) & (Ex #j. Reveal_prfk(tpk) @ j))
        )"
```

The lemmas show that in the case of KCI attacks, the adversary also has to compromise the pre-shared key between the party that she impersonates and the intended victim:

- If an initiator I believes that she has completed a handshake with a responder R, but R does not have a matching session (where the ephemeral public key and the 3 shared secrets are identical), then the pre-shared key between I and R is compromised or not in use. In addition, either the static private key of R has been compromised, or both the RNG and the PRF secret of I are compromised.
- If a responder R believes that she has a confirmed session with an initiator I, but I does not have a matching session (where the ephemeral public key as well as the 3 shared secrets are identical), then the pre-shared key between I and R is compromised or not in use. In addition, either the static private key of I has been compromised, or both the RNG and the PRF secret of R are compromised.

Unknown Key Share Attacks. We prove full UKS security proving unilateral UKS security in both directions. We prove that unilateral UKS on the responder is not possible with lemma *UKS_on_responder_resistance*.

```
lemma UKS_on_responder_resistance[reuse]:
    "not(Ex pki1 pki2 pkr peki1 peki2 psk1 psk2 ck kb ka k #i #j.
        IKeys(<pki1, pkr, peki1, psk1, ck, kb, ka, k>) @ i
        & RKeys(<pki2, pkr, peki2, psk2, ck, kb, ka, k>) @ j
        & not(pki1 = pki2)
    )"
```

In other words, we prove that a responder cannot be tricked into believing that she shares the session with another party other than the actual initiator. This is because the responder encapsulates the random secret $shk3$ in Algorithm 2 with the static public of the (claimed) initiator and mixes the ciphertext into the session key material. Consequently, bilateral UKS is also not possible.

With lemma *UKS_on_initiator_resistance* we prove UKS security for the initiator.

```
lemma UKS_on_initiator_resistance[reuse]:
```

```
"All pki pkr1 pkr2 peki1 peki2 psk1 psk2 ck kb ka k
    tpk1 tpk2 r1 r2 r3 #i #j #i1.
    // If I and R agree on keys, and it's not the case that
    IKeys(<pki, pkr1, peki1, psk1, ck, kb, ka, k>) @ i
    & PRFGen(tpk1, r3, kb) @ i1 & #i1 < #i
    & RKeys(<pki, pkr2, peki2, psk2, ck, kb, ka, k>) @ j
    & PRFGen(tpk2, r1, ka) @ j & PRFGen(tpk2, r2, k) @ j & not(
        // I's intended responder's static key is compromised
        ((Ex #j1. Reveal_AK(pkr1) @ j1)
            // or I's RNG and her PRF key are both compromised
            | ( (Ex #j1. Reveal_rnd(r3) @ j1)
                & (Ex #j1. Reveal_prfk(tpk1) @ j1) )
        )
        // I's static key is compromised
        & ((Ex #j1. Reveal_AK(pki) @ j1)
            // or R's RNG and her PRF key are both compromised
            | ( (Ex #j1. Reveal_rnd(r1) @ j1)
                & (Ex #j1. Reveal_prfk(tpk2) @ j1) )
        )
        // and the ephemeral key from I is compromised
        & ((Ex #j1. Reveal_EphK(peki1) @ j1)
            // or R's RNG and her PRF key are both compromised
            | ( (Ex #j1. Reveal_rnd(r2) @ j1)
                & (Ex #j1. Reveal_prfk(tpk2) @ j1) )
        )
        // and both PSK's are compromised (or not in use)
        & (Ex #j2. Reveal_PSK(psk1) @ j2)
        & (Ex #j2. Reveal_PSK(psk2) @ j2)
    )
==> // then UUKS on initiator is not possible
    pkr1 = pkr2 & peki1 = peki2 & psk1 = psk2"
```

UKS security for the initiator holds due to the initialization of the shared secret string $psk$with a default value that – although not necessarily secret – identifies both communicating parties even if no shared secret is established. As the adversary cannot influence this value, a UKS attack will always be detected. We prove this with the following rule and lemma.

```
/* Generate one default PSK */
rule DefaultPSKGen:
    let pkA = pk(~ltkA)
        pkB = pk(~ltkB)
        psk = h(pkA XOR pkB) in
    [ !F_AgentKey(~ltkA)
    , !F_AgentKey(~ltkB) ]--[
      DefaultPSK(pkA, psk)
    , DefaultPSK(pkB, psk)
    , Reveal_PSK(psk)
    ]->
    [
      Out(psk) // the adversary can easily derive the same psk
    , !F_AgenPSK(psk)
    ]

/* UKS on initiator is not possible with the default PSK */
lemma UKS_on_initiator_with_default_psk[reuse]:
    "not(Ex pki pkr1 pkr2 peki1 peki2 psk1 psk2 ck kb ka k #i #j #i1.
        IKeys(<pki, pkr1, peki1, psk1, ck, kb, ka, k>) @ i
        & RKeys(<pki, pkr2, peki2, psk2, ck, kb, ka, k>) @ j
        & DefaultPSK(pki, psk1) @ i1 & DefaultPSK(pkr1, psk1) @ i1
        & not(pkr1 = pkr2)
    )"
```

Identity Hiding. The identity of an initiator is encrypted in the initiation message as in WireGuard. We prove that the adversary cannot learn the identity of the initiator, i.e. decrypt the identity, without learning the random secret $shk1$ encapsulated with the intended responder's static public key. In other words, the adversary must compromise the static private key of the intended responder, or the random number generator and the PRF secret of the initiator.

E. Security-Model

The following section is essentially copied verbatim from the original paper by Benjamin Dowling and Kenneth G. Paterson with the only change being that our model explicitly considers quantum-adversaries. All our changes are highlighted in the same way as this paragraph.

We propose a modification to the eCK-PFS security model introduced by Cremers and Feltz [68] that incorporates pre-shared keys and strengthens the security definitions accordingly. We explain the framework and give an algorithmic description of the security model in Section E1, and describe the corruption abilities of the adversary in Section E2. We then describe the modifications necessary to capture the exact security guarantees that WireGuard attempts to achieve by explaining the differences between our partnering definitions and traditional notions of partnering in Section E3. We then give our modified cleanness definitions in Section E4. Given that WireGuard uses a mix of long-term identity keys, ephemeral keys and pre-shared secrets in its key exchange protocol, it is appropriate to use an extended-Canetti-Krawcyzk model (as introduced in [69]), wherein the adversary is allowed to reveal subsets of these secrets. It is claimed in [70] that WireGuard "achieves the requirements of authenticated key exchange (AKE) security, avoids key-compromise impersonation, avoids replay attacks, provides perfect forward secrecy," [70]. These are all notions captured by our extended eCK-PFS model, so our subsequent security proof will formally establish that WireGuard meets its goals.

1) Execution Environment: Consider an experiment $\mathsf{Exp}^{\mathsf{eCK\text{-}PFS\text{-}PSK}}_{\mathsf{KE},n_P,n_S,\mathcal{A}}(\lambda)$ played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. $\mathcal{C}$ maintains a set of n_P parties $P_1,\ldots,P_{n_P}$ (representing users interacting with each other via the protocol), each capable of running up to n_S sessions of a probabilistic key-exchange protocol KE, represented as a tuple of algorithms $\mathsf{KE} = (f, \mathsf{ASKeyGen}, \mathsf{PSKeyGen}, \mathsf{EPKeyGen})$. We use π_i^s to refer to both the identifier of the s-th instance of the KE being run by party P_i and the collection of per-session variables maintained for the s-th instance of KE run by P_i. We describe the algorithms below:

$\mathsf{KE}.f(\lambda, pk_i, sk_i, \pi, m) \xrightarrow{\$} (m', \pi')$ is a (potentially) probabilistic algorithm that takes a security parameter λ, the long-term asymmetric key pair pk_i, sk_i of the party P_i, a collection of per-session variables π and an arbitrary bit string $m \in \{0,1\}^* \cup \{\emptyset\}$, and outputs a response $m' \in \{0,1\}^* \cup \{\emptyset\}$ and an updated per-session state π', acting in accordance with an honest protocol implementation.

$\mathsf{KE}.\mathsf{ASKeyGen}(\lambda) \xrightarrow{\$} (pk, sk)$ is a probabilistic asymmetric-key generation algorithm taking as input a security parameter λ and outputting a public-key/secret-key pair (pk, sk).

$\mathsf{KE}.\mathsf{PSKeyGen}(\lambda) \xrightarrow{\$} (psk, pskid)$ is a probabilistic symmetric-key generation algorithm that also takes as input a security parameter λ and outputs a symmetric pre-shared secret key psk and (potentially) a pre-shared secret key identifier $pskid$.

$\mathsf{KE}.\mathsf{EPKeyGen}(\lambda) \xrightarrow{\$} (ek, epk)$ is a probabilistic ephemeral-key generation algorithm that also takes as input a security parameter λ and outputs an asymmetric public-key/secret-key pair (ek, epk).

$\mathcal{C}$ runs $\mathsf{KE.ASKeyGen}(\lambda)$ n_P times to generate a public-key/secret-key pair (pk_i, sk_i) for each party $P_i \in \{P_1, \ldots, P_{n_P}\}$ and delivers all public-keys pk_i for $i \in \{1, \ldots, n_P\}$ to $\mathcal{A}$. The challenger $\mathcal{C}$ then randomly samples a bit $b \xleftarrow{\$} \{0,1\}$ and interacts with the adversary via the queries listed in Section E2. Eventually, $\mathcal{A}$ terminates and outputs a guess b' of the challenger bit b. The adversary wins the eCK-PFS-PSK key-indistinguishability experiment if $b' = b$, and additionally if the session π_i^s such that $\mathsf{Test}(i,s)$ was issued satisfies a cleanness predicate clean, which we discuss in more detail in Section E4. We give an algorithmic description of this experiment in Figure 1.

Each session maintains the following set of per-session variables:

- $\rho \in \{\texttt{init}, \texttt{resp}\}$ – the role of the party in the current session. Note that parties can be directed to act as `init` or `resp` in concurrent or subsequent sessions.
- $pid \in \{1, \ldots, n_P, \star\}$ – the intended communication partner, represented with $\star$ if unspecified. Note that the identity of the partner session may be set during the protocol execution, in which case pid can be updated once.
- $m_s \in \{0,1\}^* \cup \{\bot\}$ – the concatenation of messages sent by the session, initialized by $\bot$.
- $m_r \in \{0,1\}^* \cup \{\bot\}$ – the concatenation of messages received by the session, initialized by $\bot$.
- $kid \in \{0,1\}^* \cup \{\bot\}$ – the concatenation of public keyshare information received by the session, initialized by $\bot$.
- $\alpha \in \{\texttt{active}, \texttt{accept}, \texttt{reject}, \bot\}$ – the current status of the session, initialized with $\bot$.
- $k \in \{0,1\}^* \cup \{\bot\}$ – the computed session key, or $\bot$ if no session key has yet been computed.
- $ek \in \{0,1\}^* \times \{0,1\}^* \cup \{\bot\}$ – the ephemeral key pair used by the session during protocol execution, initialized as $\bot$.
- $psk \in \{0,1\}^* \times \{0,1\}^* \cup \{\bot\}$ – the pre-shared secret and identifier used by the session during protocol execution, initialized as $\bot$.
- $st \in \{0,1\}^*$ – any additional state used by the session during protocol execution.

Finally, the challenger manages the following set of corruption registers, which hold the leakage of secrets that $\mathcal{A}$ has revealed.

- pre-shared keys $\{\vec{\mathsf{PSKflag}}_i, \vec{\mathsf{PSKflag}}_2, \ldots, \vec{\mathsf{PSKflag}}_{n_P}\}$ where for each element $\vec{\mathsf{PSKflag}}_i[j] \in \vec{\mathsf{PSKflag}}_i$, $\vec{\mathsf{PSKflag}}_i[j] \in \{\texttt{corrupt}, \texttt{clean}, \bot\}\ \forall\ i,j \in [n_P]$ and $\vec{\mathsf{PSKflag}}_i[j] = \bot$ for $i = j$
- long-term keys $\{\mathsf{SKflag}_1, \ldots, \mathsf{SKflag}_{n_P}\}$, where $\mathsf{SKflag}_i \in \{\texttt{corrupt}, \texttt{clean}, \bot\}\ \forall\ i \in [n_P]$
- ephemeral keys $\{\vec{\mathsf{EKflag}}_1, \ldots, \vec{\mathsf{EKflag}}_{n_P}\}$, where $\vec{\mathsf{EKflag}}_i[s] \in \{\texttt{corrupt}, \texttt{clean}, \bot\}\ \forall\ i \in [n_P]$ and $s \in [n_S]$.

We formalize the advantage of a (potentially quantum) algorithm $\mathcal{A}$ in winning the eCK-PFS-PSK key indistinguishability experiment in the following way:

Definition 1 (eCK-PFS-PSK Key Indistinguishability): Let KE be a key-exchange protocol, and $n_P, n_S \in \mathbb{N}$. For a particular given predicate clean, and a (potentially quantum) algorithm $\mathcal{A}$, we define the advantage of $\mathcal{A}$ in the eCK-PFS-PSK key-indistinguishability game to be:

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK,clean}}_{\mathsf{KE},n_P,n_S,\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{eCK\text{-}PFS\text{-}PSK,clean}}_{\mathsf{KE},n_P,n_S,\mathcal{A}}(\lambda) = 1] - \frac{1}{2}|.$$

We say that KE is eCK-PFS-PSK-secure if, for all $\mathcal{A}$ in QPT, $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK,clean}}_{\mathsf{KE},n_P,n_S,\mathcal{A}}(\lambda)$ is negligible in the security parameter λ.

2) Adversarial Interaction: Our security model is intended to be as generic as possible, in order to capture eCK-like security notions, but to also include long-term pre-shared keys. This would allow our model to be used in analyzing (for example) the Signal protocol, where users exchange both long-term Diffie-Hellman keyshares used in many protocol executions, but also many ephemeral Diffie-Hellman keyshares that are only used within a single session. Another example would be TLS 1.3, where users may have established pre-shared keys to reduce the protocol's computational overheads, or to enable 0-RTT confidential data transmission.

Our attacker is a standard key-exchange model adversary, in complete control of the communication network, able to modify, inject, delete or delay messages. They can also compromise several layers of secrets:

- long-term private keys, modeling the misuse or corruption of long-term secrets in other sessions, and additionally allowing our model to capture forward-secrecy notions.
- ephemeral private keys, modeling the use of bad randomness generators.
- pre-shared symmetric keys, modeling the leakage of shared secrets, potentially due to the misuse of the pre-shared secret by the partner, or the forced later revelation of these keys.
- session keys, modeling the leakage of keys by their use in bad cryptographic algorithms.

The adversary interacts with the challenger via the queries below. An algorithmic description of how the challenger responds is in Figure 1.

- $\mathsf{Create}(i, j, role) \to \{(i,s), \bot\}$: allows the adversary to begin new sessions. The challenger $\mathcal{C}$ creates a new session π_i^s with $\pi_i^s.pid \leftarrow j$, $\pi_i^s.\rho \leftarrow role$, $\pi_i^s.\alpha \leftarrow \texttt{active}$, $\pi_i^s.T \leftarrow \bot$, $\pi_i^s.sid \leftarrow \bot$, $\pi_i^s.k \leftarrow \bot$. $\mathcal{C}$ also computes $\mathsf{KE.EKeyGen}(\lambda) \xrightarrow{\$} (ek, epk)$ and sets $\pi_i^s.ek \leftarrow ek$. If a session π_i^s has already been created, $\mathcal{C}$ returns $\bot$. Otherwise, $\mathcal{C}$ returns (i,s) to $\mathcal{A}$.
- $\mathsf{CreatePSK}(i,j) \to \{pskid, \top, \bot\}$: allows the adversary to direct parties to generate a pre-shared key for use in future protocol executions. The challenger $\mathcal{C}$ checks that $i \neq j$ and that $\vec{\mathsf{PSK}}_i[j] = \vec{\mathsf{PSK}}_j[i] = \bot$. $\mathcal{C}$ then computes $\mathsf{KE.PSKeyGen}(\lambda) \xrightarrow{\$} psk$ and sets $\vec{\mathsf{PSK}}_i[j] =$

$\mathsf{Exp}^{\mathsf{eCK\text{-}PFS\text{-}PSK\text{-}ind}}_{\mathsf{KE},\mathsf{clean},n_P,n_S,\mathcal{A}}(\lambda)$:

$b \xleftarrow{\$} \{0,1\}$
tested $\leftarrow$ `false`
for $i = 1$ **to** n_P **do**
  $(pk_i, sk_i) \xleftarrow{\$} \mathsf{ASKeyGen}(\lambda)$
  $\mathsf{ASKflag}_i \leftarrow$ `clean`
  $\vec{\mathsf{PSK}}_i[1], \ldots, \vec{\mathsf{PSK}}_i[n_P] \leftarrow \perp$
  $\vec{\mathsf{PSKflag}}_i[1], \ldots, \vec{\mathsf{PSKflag}}_i[n_P] \leftarrow \perp$
  $\vec{\mathsf{EPKflag}}_i[1], \ldots, \vec{\mathsf{EPKflag}}_i[n_S] \leftarrow \perp$
  $\vec{\mathsf{RSKflag}}_i[1], \ldots, \vec{\mathsf{RSKflag}}_i[n_S] \leftarrow \perp$
  $ctr_i \leftarrow 0$
end for
$b' \xleftarrow{\$} \mathcal{A}^{\mathsf{Send},\mathsf{Create}*,\mathsf{Corrupt}*,\mathsf{Reveal},\mathsf{Test}}(pk_1, \ldots, pk_{n_P})$
if $\mathsf{clean}(\pi_i^s) \wedge (b = b')$ **then**
  return 1
else
  $b' \xleftarrow{\$} \{0,1\}$
  return b'
end if

$\mathsf{Create}(i, j, role)$:

$ctr_i \leftarrow ctr_i + 1$
$s \leftarrow ctr_i$
$\pi_i^s.pid \leftarrow j$
$\pi_i^s.\rho \leftarrow role$
$\pi_i^s.ek \leftarrow \mathsf{KE.EPKeyGen}(\lambda)$
$\pi_i^s.psk \leftarrow \vec{\mathsf{PSK}}_i[j]$ (i, s)

$\mathsf{Send}(i, s, m)$:

if $\pi_i^s = \perp$ **then** $\perp$
else
  $\pi_i^s.m_r \leftarrow \pi_i^s.m_r \| m$
  $(\pi_i^s, m') \leftarrow \mathsf{KE}.f(\lambda, pk_i, sk_i, \pi_i^s, m)$
  $\pi_i^s.m_s \leftarrow \pi_i^s.m_s \| m'$
  $\pi_i^s.T \leftarrow \pi_i^s.T \| m \| m'$ m'
end if

$\mathsf{CreatePSK}(i, j)$:

if $(i = j) \vee (\vec{\mathsf{PSKflag}}_i[j] \neq \perp)$ **then** $\perp$
end if
$(psk, pskid) \leftarrow \mathsf{KE.PSKeyGen}(\lambda)$
$\vec{\mathsf{PSK}}_i[j] \leftarrow (psk, pskid)$
$\vec{\mathsf{PSK}}_j[i] \leftarrow (psk, pskid)$
$\vec{\mathsf{PSKflag}}_i[j], \vec{\mathsf{PSKflag}}_j[i] \leftarrow$ `clean`
if $pskid \neq \emptyset$ **then** $pskid$
else $\top$
end if

$\mathsf{Reveal}(i, s)$:

if $\pi_i^s.\alpha \neq$ `accept` **then** $\perp$
else
  $\vec{\mathsf{RSKflag}}_i[s] \leftarrow$ `corrupt` $\pi_i^s.k$
end if

$\mathsf{CorruptASK}(i)$:

$\mathsf{ASKflag}_i \leftarrow$ `corrupt` sk_i

$\mathsf{CorruptEPK}(i, s)$:

$\vec{\mathsf{EKflag}}_i[s] \leftarrow$ `corrupt` $\pi_i^s.ek$

$\mathsf{CorruptPSK}(i, j)$:

if $\vec{\mathsf{PSK}}_i[j] = \perp$ **then** $\perp$
end if
if $\vec{\mathsf{PSKflag}}_i[j] \neq$ `clean` **then** $\perp$
else
  $\vec{\mathsf{PSKflag}}_i[j] \leftarrow$ `corrupt`
  $\vec{\mathsf{PSKflag}}_j[i] \leftarrow$ `corrupt` $\vec{\mathsf{PSK}}_i[j]$
end if

$\mathsf{Test}(i, s)$:

if (tested $=$ `true`) $\vee$ $(\pi_i^s.\alpha \neq$ `accept`$)$ **then** $\perp$
end if
tested $\leftarrow$ `true`
if $b = 0$ **then** $\pi_i^s.k$
else
  $k \xleftarrow{\$} \mathcal{K}$ k
end if

Fig. 1. eCK-PFS-PSK experiment for adversary $\mathcal{A}$ against the key-indistinguishability security of protocol KE.

$\vec{\mathsf{PSK}}_j[i] \leftarrow psk$, and the PSK register $\vec{\mathsf{PSKflag}}_i[j] = \vec{\mathsf{PSKflag}}_j[i] \leftarrow$ `clean`. If $pskid \neq \emptyset$, then $\mathcal{C}$ returns $pskid$ to $\mathcal{A}$, otherwise $\mathcal{C}$ returns $\top$ (where $\top$ is a generic success flag) to $\mathcal{A}$. If $\vec{\mathsf{PSK}}_i[j] \neq \perp$ or $\vec{\mathsf{PSK}}_j[i] \neq \perp$ (i.e. if $\mathcal{A}$ has previously issued a $\mathsf{CreatePSK}(i, j)$ or $\mathsf{CreatePSK}(j, i)$ query), then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.

- $\mathsf{Reveal}(i, s)$: allows the adversary access to the secret session key computed by a session during protocol execution. The challenger checks whether the cleanness of the session π_i^s has been upheld and $\pi_i^s.\alpha =$ `accept` and if so, returns $\pi_i^s.k$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.
- $\mathsf{CorruptPSK}(i) \rightarrow \{psk, \perp\}$: allows the adversary access to the secret pre-shared key jointly shared by parties prior to protocol execution. The challenger $\mathcal{C}$ checks that $\vec{\mathsf{PSK}}_i[j] = \vec{\mathsf{PSK}}_j[i] \neq \perp$, and that $\vec{\mathsf{PSKflag}}_i[j] = \vec{\mathsf{PSKflag}}_j[i] =$ `clean`. If so, $\mathcal{C}$ returns $PSK \leftarrow \vec{\mathsf{PSK}}_i[j]$ to $\mathcal{A}$ and sets $\vec{\mathsf{PSKflag}}_i[j] = \vec{\mathsf{PSKflag}}_j[i] \leftarrow$ `corrupt`. If $\vec{\mathsf{PSK}}_i[j] = \vec{\mathsf{PSK}}_j[i] = \perp$ or $\vec{\mathsf{PSKflag}}_i[j] = \vec{\mathsf{PSKflag}}_j[i] \neq$ `clean`, (i.e. that the adversary has either not previously created a psk between the two parties P_i and P_j, or has previously issued a $\mathsf{CorruptPSK}(i, j)$/$\mathsf{CorruptPSK}(j, i)$ query), then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.
- $\mathsf{CorruptASK}(i) \rightarrow \{sk_i, \perp\}$: allows the adversary access to the secret long-term key generated by a party prior to protocol execution. The challenger $\mathcal{C}$ checks that $\mathsf{ASKflag}_i \neq$ `corrupt`. If so, $\mathcal{C}$ returns sk_i to $\mathcal{A}$. If $\mathsf{ASKflag}_i =$ `corrupt` (i.e. $\mathcal{A}$ has previously issued a $\mathsf{CorruptASK}(i)$ query), then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.
- $\mathsf{CorruptEPK}(i, s) \rightarrow \{ek, \perp\}$: allows the adversary access to the secret ephemeral key generated by a session during protocol execution. The challenger $\mathcal{C}$ checks that $\vec{\mathsf{EPKflag}}_{i,s} =$ `clean`. If so, $\mathcal{C}$ returns $\pi_i^s.ek$ to $\mathcal{A}$, and sets $\vec{\mathsf{EPKflag}}_{i,s} \leftarrow$ `corrupt`. If $\vec{\mathsf{EPKflag}}_{i,s} =$ `corrupt`, (i.e. $\mathcal{A}$ has previously issued a $\mathsf{CorruptEPK}(i, s)$ query), then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.
- $\mathsf{Send}(i, s, m) \rightarrow \{m', \perp\}$: allows the adversary to send messages to sessions for protocol execution and receive their output. If a session π_i^s has not been previously created, or $\pi_i^s.\alpha \neq$ `active`, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.

Otherwise, $\mathcal{C}$ computes $\mathsf{KE}.f(\lambda, m, \pi_i^s) \to (m', \pi_i^s)$, sets $\pi_i^s \leftarrow \pi_i^{s\prime}$, and returns m' to $\mathcal{A}$.

- $\mathsf{Test}(i,s) \to \{k, \perp\}$: sends the adversary a real-or-random session key used in determining the success of $\mathcal{A}$ in the key-indistinguishability game. If a session π_i^s exists and $\pi_i^s.\alpha = \texttt{accept}$, then the challenger $\mathcal{C}$ samples a key $k_0 \xleftarrow{\$} \mathcal{D}$ where $\mathcal{D}$ is the distribution of the session key, and sets $k_1 \leftarrow \pi_i^s.k$. $\mathcal{C}$ then returns k_b (where b is the random bit sampled during set-up) to $\mathcal{A}$. If a session π_i^s does not exist, or $\pi_i^s.\alpha \neq \texttt{accept}$, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.

3) Partnering Definitions: In order to evaluate which secrets the adversary is able to reveal without trivially breaking the security of the protocol, key-exchange models must define how sessions are *partnered*. Otherwise, an adversary would simply run a protocol between two sessions, faithfully delivering all messages, Test the first session to receive the real-or-random key, and Reveal the session partner's key. If the keys are equal, then the Test key is real, and otherwise the session key has been sampled randomly. BR-style key-exchange models traditionally use *matching conversations* in order to do this. When introducing the eCK-PFS model, Cremers and Feltz [68] used the relaxed notion of *origin sessions*. However, both of these are still too restrictive for analyzing WireGuard, because this protocol does not explicitly authenticate the full transcript. Instead, for WireGuard, we are concerned matching only on a subset of the transcript information – the honest contributions of the keyshare and key-derivation materials. We introduce the notion of *contributive keyshares* to capture this intuition.

Definition 2 (Contributive keyshares): Recall that $\pi_i^s.kid$ is the concatenation of all keyshare material sent by the session π_i^s during protocol execution. We say that π_j^t is a *contributive keyshare session* for π_i^s if $\pi_j^t.kid$ is a substring of $\pi_i^s.m_r$.

This definition is protocol specific: in WireGuard $\pi_i^s.kid$ consists only of the long-term public Diffie-Hellman value and the ephemeral public Diffie-Hellman value provided by the initiator and responder; in TLS 1.3 (for example) it would consist of the long-term public keys, the ephemeral public Diffie-Hellman values and any pre-shared key identifiers provided by the client and selected by the server.

4) Cleanness Predicates: We now define the exact combinations of secrets that an adversary is allowed to leak without trivially breaking the protocol. The original cleanness predicate of Cremers and Feltz [68] allows the reveal of long-term secrets for the test session's party P_i at any time, which places us firmly in the setting where the adversary has key-compromise-impersonation abilities, but only allowed the reveal of long-term secrets of the intended peer after the test session has established a secure session, which captures perfect forward secrecy.

We now turn to modifying the cleanness predicate $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ for the pre-shared secret setting.

Definition 3 ($\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$): A session π_i^s such that $\pi_i^s.\alpha = \texttt{accept}$ in the security experiment defined in Figure 1 is $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ if all of the following conditions hold:

1) The query $\mathsf{Reveal}(i,s)$ has not been issued.
2) For all $(j,t) \in n_P \times n_S$ such that π_i^s *matches* π_j^t, the query $\mathsf{Reveal}(j,t)$ has not been issued.
3) If $\mathsf{PSK}\vec{\mathsf{flag}}_i[\pi_i^s.pid] = \texttt{corrupt}$ or $\pi_i^s.psk = \perp$, the queries $\mathsf{CorruptASK}(i)$ and $\mathsf{CorruptEPK}(i,s)$ have not *both* been issued.
4) If $\mathsf{PSK}\vec{\mathsf{flag}}_i[\pi_i^s.pid] = \texttt{corrupt}$ or $\pi_i^s.psk = \perp$, and for all $(j,t) \in n_P \times n_S$ such that π_j^t is a *contributive keyshare session* for π_i^s, then $\mathsf{CorruptASK}(j,t)$ and $\mathsf{CorruptEPK}(j,t)$ have not *both* been issued.
5) If there exists no $(j,t) \in n_P \times n_S$ such that π_j^t is a contributive keyshare session for π_i^s, $\mathsf{CorruptASK}(j)$ has not been issued before $\pi_i^s.\alpha \leftarrow \texttt{accept}$.

We specifically forbid the adversary from revealing the long-term and ephemeral secrets if the pre-shared secret between the test session and its intended partner has already been revealed. Since pre-shared keys are optional in our framework, we also must consider the scenario where a pre-shared secret does not exist between the test session π_i^s and its intended partner. Similarly, we forbid the adversary from revealing the long-term and ephemeral secrets if there exists no pre-shared secret between the two parties. Finally, since WireGuard does not authenticate the full transcript, but relies instead on implicit authentication of derived session keys based on secret information, we must use our contributive keyshare partnering definition instead of the origin sessions of [68]. Like eCK-PFS, we capture perfect forward secrecy under key-compromise-impersonation attack in condition 5, where the long-term secret of the test session's intended partner is allowed to be revealed only after the test session has accepted. Additionally, we allow for the optional incorporation of pre-shared secrets in conditions 3 and 4, where the adversary falls back to eCK-PFS leakage paradigm if the pre-shared secret between the test session and its peer either does not already exist, or has been already revealed.

F. Full Proof

In this section we present the full security-proof of our scheme. Most of it is taken verbatim from the WireGuard-proof[9] by Benjamin Dowling and Kenneth G. Paterson whom we thank for kindly providing us with their LaTeX-sources; the highlighted parts (such as this one) are our own modifications to that proof so that it also works with PQ-WireGuard. This was done with the intention of allowing readers who are already familiar with the older proof to concentrate on our changes to it. For the same reason we also tried to keep the style of our additions as close to the original paper as possible.

One thing that we would like to point out concerns the game-hops that use the prf- and the $\mathsf{prf}^{\mathsf{swap}}$ assumptions: One might intuitively assume that keys or messages could potentially collide and result in a tightness-loss (like the ones that actually occur in **Game 5a** of **Case 1**, **Game 5a** of **Case 2** and **Game 3a** of **Case 3.5**). This is not the case however: Since the key is random and independent from any other key in all these cases, any potential collision of the key is already part of the adversarial advantage against the prf/$\mathsf{prf}^{\mathsf{swap}}$-security. As any (non-colliding) key is furthermore only used once (except for the aforementioned cases), there cannot be any colliding messages. The later part actually strengthens the real security of the protocol, since it massively reduces the kinds of attacks against the HKDF that can be converted into attacks against PQ-WireGuard.

While Dowling and Paterson don't make this statement as explicitly (possibly because they considered it obvious), it applies to their proof just as well.

Theorem 1: The modified WireGuard handshake protocol pqWG is eCK-PFS-PSK-secure with cleanness predicate $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ (capturing forward secrecy and resilience to KCI attacks). That is, for any (potentially quantum) algorithm $\mathcal{A}$ against the eCK-PFS-PSK key-indistinguishability game (defined in Figure 1) the adversarial advantage $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK}}_{\mathsf{pqWG},\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}},n_P,n_S,\mathcal{A}}(\lambda)$ 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}{r} \frac{n_s}{2^\lambda} + \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda) \ +\ 6\cdot\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ +\ 2\cdot\mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \ +\ \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda) \end{array} \right) \\
&+\ n_P^2 n_S \left(\begin{array}{r} \frac{n_s}{2^\lambda} + \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda) \ +\ 3\cdot\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ +\ 2\cdot\mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \ +\ \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda) \end{array} \right) \\
&+\ \max \left\{ \begin{array}{l}
\left(n_P^2 n_S^2 \left(\begin{array}{l} 2\cdot\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ +\ \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \end{array} \right) \right), \\
\left(n_P^2 n_S^2 \left(\begin{array}{l} \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{CPAKEM},\mathcal{A}}(\lambda) \\ +\ 4\cdot\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ +\ \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \end{array} \right) \right), \\
\left(n_P^2 n_S^2 \left(\begin{array}{l} \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda) \\ +\ 7\cdot\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ +\ 2\cdot\mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \end{array} \right) \right), \\
\left(n_P^2 n_S^2 \left(\begin{array}{l} \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda) \\ +\ 3\cdot\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ +\ 2\cdot\mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \end{array} \right) \right), \\
\left(n_P^2 n_S^2 \left(\begin{array}{l} \frac{n_s}{2^\lambda} + \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda) \\ +\ 7\cdot\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ +\ \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \end{array} \right) \right)
\end{array} \right\}
\end{aligned}
$$

By combining some terms we can simplify this equation to the following, simpler one:

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

At the cost of a (remarkably small) loss in tightness, we can further simplify this to the following:

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

Note that for readability reasons, we drop the convention of including the cleanness predicate $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ in the advantage notation in what follows. We begin by dividing the proof into three separate cases (and denote with $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_l}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda)$ the advantage of the adversary in winning the key-indistinguishability game in Case l) where the query $\mathsf{Test}(i,s)$ has been issued:

1) The session π_i^s (where $\pi_i^s.\rho = \mathtt{init}$) has no contributive keyshare session.
2) The session π_i^s (where $\pi_i^s.\rho = \mathtt{resp}$) has no contributive keyshare session.
3) The session π_i^s has a contributive keyshare session.

It follows then that $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}} \leq \big(\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_1}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_2}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_3}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda)\big)$. We then bound the probability of each case, and show that under certain assumptions, the probability of the adversary winning in the key-indistinguishability game is negligible.

In the first two cases, we show that the adversary's probability in getting the session π_i^s to reach an "accept" state (and thus generate keys used in the real-or-random key indistinguishability game) is negligible, and since the adversary cannot cause the test session π_i^s to reach the accept state, the experiment will act identically regardless of whether the test bit b is 0 or 1, and thus the adversary's probability in winning the key indistinguishability game is negligible.

In the third case, we show that under certain assumptions, replacing the session keys with uniformly random, independent keys from the same distribution has a negligible chance of being detected and thus, the adversary's advantage in distinguishing the real-or-random key-indistinguishability game is also negligible. We begin with the first case.

1) Case 1: Test `init` *session without contributive keyshare session:* In this case we bound the probability that a test initiator session will accept when there exists no contributive keyshare session. Recall that a contributive keyshare session π_j^t exists for a session π_i^s when $\pi_j^t.kid$ is a substring of $\pi_i^s.m_r$. Informally, the test session π_i^s has not received keying material from an honest partner session, having either been modified or injected wholesale by the adversary.

a) ***Proof Sketch****:* We begin first by adding an abort rule that triggers if there is ever a hash collision during the challenger's execution of any honest session. We follow by guessing the index of the test session, and adding an abort event that occurs if a Test query is directed to a session that does not have the index of the guessed session, and similarly, guess the party index of the intended partner session. Afterwards, we add another abort event that occurs if the guessed test session π_i^s reaches the `reject` status. Since we already abort if the guessed session is not the session indicated by the Test query, and if the session π_i^s has reached the reject status, the $\mathsf{Test}(i, s)$ query will always respond with $\perp$, there is no difference in the adversary's advantage in the two games - any further queries that the adversary makes is responded to identically regardless of the sampling of the random test bit b.

We define an abort event $abort_{\mathtt{accept}}$ that will occur if $\pi_i^s \leftarrow \mathtt{accept}$. The following games then are designed to bound the probability of $abort_{\mathtt{accept}}$ occurring to be negligibly close to zero. Note that from this game onwards, the adversary is unable to make a $\mathsf{CorruptASK}(j)$ query, since we now abort the game when the session π_i^s reaches a status that is not `active`, and by the Case 1 definition (a test session without a contributive keyshare session) and the cleanness predicate $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$, the adversary can only win by not issuing a $\mathsf{CorruptASK}(j)$ query before the test session completes. We can now (cleverly) embed CCAKEM challenge values from the IND-CCA challenger into the long-term asymmetric keys of the party P_j without needing to address the adversary's ability to issue a $\mathsf{CorruptASK}(j)$ query.

We then replace the values C_3, κ_3 with uniformly random and independent values $\widetilde{C_3}, \widetilde{\kappa_3}$, and argue that any adversary capable of distinguishing this change would be able to break either the prf or the IND-CCA assumption. In the next game we replace the values C_4, κ_4 with uniformly random and independent values $\widetilde{C_4}, \widetilde{\kappa_4}$, and argue that any adversary capable of distinguishing this change would be able to break the prf security of HKDF.

In a similar fashion, we use a chain of prf challengers to replace C_6, C_7, C_8 and finally C_9, tmp, κ_9 with uniformly random and independent values $\widetilde{C_6}$, $\widetilde{C_7}$, $\widetilde{C_8}$ and $\widetilde{C_9}$, $\widetilde{tmp}$, $\widetilde{\kappa_9}$. We argue that any adversary $\mathcal{A}$ capable of distinguishing these changes can be turned into a successful distinguishing adversary against the prf security of HKDF.

In the final game hop, we use the fact that $\widetilde{\kappa_9}$ is a uniformly random and independent value to embed $\widetilde{\kappa_9}$ within an aead challenger, and add an abort rule $abort_{dec}$ that triggers when the test session π_i^s decrypts a `zero` ciphertext received in the `RespHello` message. To do so, we use the aead decryption oracle to replace concrete decryptions performed in the test session. Logically then, since the $\widetilde{\kappa_9}$ value is internal to the aead challenger, if `zero` decrypts correctly, then $\mathcal{A}$ has managed to produce a ciphertext $\mathsf{AEAD.Enc}(\widetilde{\kappa_9}, 0, \emptyset, H_9)$ that has not been the result of an encryption oracle query on $(\emptyset, H_9)$, and we can use `zero`, to break the aead security of the AEAD scheme. We note that since $\widetilde{\kappa_9}$ is already a uniformly random and independent value, that this change is sound, and that the probability of $abort_{dec}$ triggering is bound by the probability of adversary breaking the aead security of AEAD.

Since a session with role $\pi_i^s.\rho = \mathtt{init}$ will only accept if it receives a ciphertext `zero` that decrypts correctly, and $abort_{dec}$ triggers if such a ciphertext decrypts correctly, then the probability of π_i^s reaching an `accept` state is 0 in the final game, and the adversary cannot force a session π_i^s to accept without an honest partner π_j^t. We show this using the following sequence of games: **Game 0** This is a standard eCK-PFS-PSK game. Thus we have

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_1}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda) = \Pr(break_0).$$

Game 1 In this game, we guess the index (i, s) of the session π_i^s, and abort if during the execution of the experiment, a query $\mathsf{Test}(i^*, s^*)$ is received and $(i^*, s^*) \neq (i, s)$. Thus:

$$\Pr(break_0) \leq n_P n_S \cdot \Pr(break_1)$$

Game 2 In this game, we guess the party of the intended partner of the test session π_i^s, and abort if $\pi_i^s.pid \neq j$. Thus

$$\Pr(break_1) \leq n_P \cdot \Pr(break_2).$$

Game 3 In this game, we abort if the session π_i^s sets the status $\pi_i^s.\alpha \leftarrow \mathtt{reject}$. Note that by **Game 2** we abort if the Test query is ever issued to a session that is not π_i^s. If the session π_i^s ever reaches the status $\pi_i^s.\alpha \leftarrow \mathtt{reject}$, then the $\mathsf{Test}(i, s)$ query will be rejected by the challenger as specified in Figure 1. Note that the difference between the adversary's advantage in **Game 2** and **Game 3** is 0: The sampling of the test bit b by the challenger only affects the response to the $\mathsf{Test}(i, s)$ query, which is always rejected if $\pi_i^s.\alpha = \mathtt{reject}$. Thus

$$\Pr(break_2) = \Pr(break_3).$$

Game 4 In this game we define an abort event $abort_{\mathtt{accept}}$ that triggers if the status of the test session $\pi_i^s \leftarrow \mathtt{accept}$. It is clear then that

$$\Pr(break_3) = \Pr(abort_{\mathtt{accept}}) + 1/2.$$

In the following sequence of games, we show that the probability of the abort event triggering (i.e. $\Pr(abort_{\mathtt{accept}})$) is negligibly close to zero.

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 hybrid into three. This is necessary because of the more convoluted combination of the static keys with both the other parties 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 ct1 with a random value $\mathtt{k}^*$ (**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 hybrid instead of inserting new ones is that we want to stay consistent with the numbering of the hybrids 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 theses 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 match the ones from the original proof by Dowling and Paterson. (Again: This does not affect cases 2 and 3.)

In **Game 5a** we replace the value $\hat{r} := \mathsf{HKDF}(\sigma_i, r_i)$ 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 $\hat{r} \leftarrow \mathsf{HKDF}(\sigma_i, r_i)$ and we are in **Game 4**. If the test bit sampled by the $\mathsf{prf}^{\mathsf{swap}}$ challenger is 1, then $\hat{r} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ 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 $\mathsf{HKDF}(\sigma_i, r_i)$ 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 $\frac{n_S}{2^\lambda}$ (which when multiplied by the number of sessions results in the famous approximation of the birthday-bound $\frac{n_S^2}{2^\lambda}$).

Given that, we initialize a prf challenger and replace all computations of $\mathsf{HKDF}(\sigma_i, \cdot)$ 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 $\hat{r} \leftarrow \mathsf{HKDF}(\sigma_i, r_i)$ and we are in **Game 4**. If the test bit sampled by the prf challenger is 1, then $\hat{r} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ 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 HKDF, and we find:

$$\Pr(abort_{\mathtt{accept}}) \\ \leq \frac{n_S}{2^\lambda} + \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_{5a})$$

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 our version of WireGuard, the long-term private keys are used to compute the following:

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

(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 $\texttt{shk1}$. 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 $\texttt{k}^*$ is indeed the shared secret encapsulated in $\texttt{c}^*$ and we are in **Game 5a**.
- If the test bit sampled by the IND-CCA challenger is 1, then $\texttt{k}^*$ is not the shared secret encapsulated in $\texttt{c}^*$ 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:

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

In **Game 5c** we replace the values of C_3, κ_3 with uniformly random and independent values $\widetilde{C_3}, \widetilde{\kappa_3} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ (where $\{0,1\}^{|\mathsf{HKDF}|}$ is the output space of the HKDF) used in the protocol execution of the test session. Specifically, we initialize a $\mathsf{prf}^{\mathsf{swap}}$ challenger and query $\texttt{shk1}$, 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**, $\texttt{shk1}$ 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 $\widetilde{C_3}, \widetilde{\kappa_3} \leftarrow \mathsf{HKDF}(C_2, \texttt{shk1})$ and we are in **Game 5b**. If the test bit sampled by the $\mathsf{prf}^{\mathsf{swap}}$ challenger is 1, then $\widetilde{C_3}, \widetilde{\kappa_3} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ 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 HKDF, and we find:

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

Game 6 In this game we replace the values C_4, κ_4 with uniformly random and independent values $\widetilde{C_4}, \widetilde{\kappa_4} \xleftarrow{\$} \{0,1\}^{\mathsf{HKDF}}$ (where $\{0,1\}^{\mathsf{HKDF}}$ is the output space of the HKDF) used in the protocol execution of the test session. Specifically, we initialize a prf challenger and query psk, and use the output $\widetilde{C_4}, \widetilde{\kappa_4}$ from the prf challenger to replace the computation of C_4, κ_4. Since by **Game 5c**, $\widetilde{C_3}$ is a uniformly random and independent value, this replacement is sound. If the test bit sampled by the prf challenger is 0, then $\widetilde{C_4}, \widetilde{\kappa_4} \leftarrow \mathsf{HKDF}(C_3, psk)$ and we are in **Game 5c**. If the test bit sampled by the prf challenger is 1, then $\widetilde{C_4}, \widetilde{\kappa_4} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 6**. Thus any adversary $\mathcal{A}$ capable of distinguishing this change can be turned into a successful adversary against the prf security of HKDF, and we find:

$$\Pr(break_{5c}) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_{6})$$

Game 7 In this game we replace the value C_6 with a uniformly random and independent value $\widetilde{C_6} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ (where $\{0,1\}^{|\mathsf{HKDF}|}$ is the output space of HKDF) used in the protocol execution of the test session. Specifically, we initialize a prf challenger, query it with $\texttt{ct2}$, and use the output $\widetilde{C_6}$ from the prf challenger to replace the computation of C_6. Since by **Game 6**, $\widetilde{C_4}$ is a uniformly random and independent value, this replacement is sound. If the test bit sampled by the prf challenger is 0, then $\widetilde{C_6} \leftarrow \mathsf{prf}(C_4, \texttt{ct2})$ and we are in **Game 6**. If the test bit sampled by the prf challenger is 1, then $\widetilde{C_6} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 7**. Thus any adversary $\mathcal{A}$ capable of distinguishing this change can be turned into a successful adversary against the prf security of HKDF, and we find:

$$\Pr(break_{6}) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_{7})$$

Game 8 As in previous games, we replace the computation of C_7 with a uniformly random value $\widetilde{C_7}$ from the same distribution, in the challenger's execution of the test session π_i^s. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_7 \leftarrow \mathsf{HKDF}(\widetilde{C_6}, \texttt{shk2})$ we instead initialize a prf challenger and query it with $\texttt{shk2}$. We note that by **Game 7** that $\widetilde{C_6}$ is a uniformly random value and independent value, and thus this replacement is sound.If the random bit b sampled by the prf challenger is 0, then $\widetilde{C_7} \leftarrow \mathsf{HKDF}(\widetilde{C_6}, \texttt{shk2})$ and we are in **Game 7**. If the random bit b sampled by the prf challenger is 1, then $\widetilde{C_7} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 8** . Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF, and thus

$$\Pr(break_{7}) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_{8}).$$

Game 9 As in previous games, we replace the computation of C_8 with a uniformly random value $\widetilde{C_8}$ from the same distribution, in the challenger's execution of the test session π_i^s. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_8 \leftarrow \mathsf{HKDF}(\widetilde{C_7}, \texttt{shk3})$ we instead initialize a prf challenger and query it with $\texttt{shk3}$. We note that by **Game 8** that $\widetilde{C_7}$ is a uniformly random value and independent value, and thus this replacement is sound.If the random bit b sampled by the prf challenger is 0, then $C_8 \leftarrow \mathsf{HKDF}(\widetilde{C_7}, \texttt{shk3})$ and we are in **Game 8**. If the random bit b sampled by the prf challenger is 1, then $\widetilde{C_8} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 9**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF, and thus

$$\Pr(break_{8}) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_{9}).$$

Game 10 As in previous games, we replace the computation of C_9, tmp, κ_9 with uniformly random values $\widetilde{C_9}, \widetilde{tmp}, \widetilde{\kappa_9}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a HKDF challenger in the following way: When it is time to compute $C_9, tmp, \kappa_9 \leftarrow \mathsf{HKDF}(\widetilde{C_8}, psk)$ we instead initialize a prf challenger and query it with psk. We note that by **Game 9** that $\widetilde{C_8}$ is a uniformly random value and independent value, and thus this replacement is sound.If the random bit b sampled by the prf challenger is 0, then we are in **Game 9**. If the random bit b sampled by the prf challenger is 1, then we are in **Game 10**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF, and thus

$$\Pr(break_9) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_{10}).$$

Game 11 In this game, the test session π_i^s will only set $\pi_i^s.\alpha \leftarrow \texttt{accept}$ if the adversary is able to produce a value $\texttt{zero} = \mathsf{AEAD}(\widetilde{\kappa_9}, 0, H_9, \emptyset)$ that decrypts correctly. In this game, we now initialize an aead challenger to decrypt $\texttt{RespHello.zero}$ ciphertexts in the test session π_i^s. By **Game 10** that $\widetilde{\kappa_9}$ is a uniformly random and independent value, and thus this change is undetectable. Since the $\widetilde{\kappa_9}$ is internal to the aead challenger, then it follows that the adversary capable of forging such a $\texttt{zero}$ ciphertext breaks the security of the AEAD scheme. We find that

$$\Pr(break_{10}) = \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda).$$

Thus

$$\begin{aligned}\Pr(abort_{\mathsf{accept}}) \leq & \Big(\frac{n_S}{2^\lambda} + \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda) \\ & + 6 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + 2 \cdot \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ & + \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)\Big)\end{aligned}$$

It follows then

$$\begin{aligned}\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_1}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda) \leq\ & n_P^2 n_S \Big(\frac{n_S}{2^\lambda} \\ & + \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda) \\ & + 6 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ & + 2 \cdot \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ & + \mathsf{Adv}^{\mathsf{aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)\Big).\end{aligned}$$

2) Case 2: Test $\texttt{resp}$ *session without contributive keyshare partner:* In this case we bound the probability that a session π_i^s such that $\pi_i^s.\rho = \texttt{resp}$ will accept when there exists no contributive keyshare partner. Recall that an contributive keyshare partner exists for a session π_i^s when for some session π_j^t, $\pi_j^t.kid$ is a substring of $\pi_i^s.m_r$. Informally, the test session π_i^s has not received the keyshares that were honestly generated by another session, having either been modified or injected wholesale by the adversary.

*a) **Proof sketch**:* We begin by guessing the index of the test session, and adding an abort event that occurs if a Test query is directed to a session that does not have the index of the guessed session, and similarly, guess the party index of the intended partner session. Afterwards, we add another abort event that occurs if the guessed test session π_i^s reaches the $\texttt{reject}$ status. Since we already abort if the guessed session is not the session indicated by the Test query, and if the session π_i^s has reached the reject status, the $\mathsf{Test}(i, s)$ query will always respond with $\perp$, there is no difference in the adversary's advantage in the two games - any further queries that the adversary makes is responded to identically regardless of the sampling of the random test bit b.

We define an abort event $abort_{\mathsf{accept}}$ that will occur if $\pi_i^s \leftarrow \texttt{accept}$. The following games then are designed to bound the probability of $abort_{\mathsf{accept}}$ occurring to be negligibly close to zero. Note that from this game onwards, the adversary is unable to make a $\mathsf{CorruptASK}(j)$ query, since we now abort the game when the session π_i^s reaches a status that is not $\texttt{active}$, and by the Case 1 definition (a test session without a contributive keyshare session) and the cleanness predicate $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$, the adversary can only win by not issuing a $\mathsf{CorruptASK}(j)$ query before the test session completes. We can now (cleverly) embed CCAKEM challenge values from the IND-CCA challenger into the long-term asymmetric keys of the party P_j without needing to address the adversary's ability to issue a $\mathsf{CorruptASK}(j)$ query.

We then replace the value C_8 with a uniformly random and independent value $\widetilde{C_8}$, and argue that any adversary capable of distinguishing this change would be able to break either the prf or the IND-CCA assumption. In the next game we replace the values C_9, tmp, κ_4 with uniformly random and independent values $\widetilde{C_9}, \widetilde{tmp}, \widetilde{\kappa_4}$, and argue that any adversary capable of distinguishing this change would be able to break the PRF assumption. In a similar fashion, we replace the values C_{10}, κ_{10} with uniformly random and independent values $\widetilde{C_{10}}, \widetilde{\kappa_{10}}$ and again argue that any distinguishing adversary can be turned into an adversary against the PRF assumption. Finally, we argue that the test session π_i^s will only reach an $\texttt{accept}$ state (and trigger the $abort_{\mathsf{accept}}$ event) if it receives a value $\texttt{conf} = \mathsf{AEAD.Enc}(\widetilde{\kappa_{10}}, 0, \emptyset, H_{10})$. We use the fact that $\widetilde{\kappa_{10}}$ is a uniformly random and independent value to embed $\widetilde{\kappa_{10}}$ within an $\texttt{aead-auth}$ challenger, and add an abort rule $abort_{dec}$ that triggers if the $\texttt{conf}$ ciphertext received in the $\texttt{SenderConf}$ message would decrypt without error. Logically then, since the $\widetilde{\kappa_{10}}$ value is internal to the $\texttt{aead}$ challenger, if $\texttt{conf}$ would decrypt correctly, then $\mathcal{A}$ has managed to produce a ciphertext $\mathsf{AEAD.Enc}(\widetilde{\kappa_{10}}, 0, \emptyset, H_{10})$ that has not been the result of an encryption oracle query on $(0, \emptyset, H_{10})$, and we can use $\texttt{zero}$ to break the $\texttt{aead-auth}$ security of the AEAD scheme. We note that since $\widetilde{\kappa_{10}}$ is already a uniformly random and independent value, that this change is sound, and that the probability of $abort_{dec}$ triggering is bound by the probability of adversary breaking the $\texttt{aead-auth}$ security of AEAD.

Since a session with role $\pi_i^s.\rho = \texttt{resp}$ will only accept if it receives a ciphertext $\texttt{conf}$ that decrypts correctly, and $abort_{dec}$ triggers if such a ciphertext decrypts correctly, then the probability of π_i^s reaching an $\texttt{accept}$ state is 0 in the final game, and the adversary cannot force a session π_i^s to accept without a contributive keyshare partner π_j^t. **Game 0** This is a standard eCK-PFS-PSK game. Thus we have:

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_2}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda) = \Pr(break_0)$$

Game 1 In this game, we guess the index (i, s) of the session π_i^s, and abort if during the execution of the experiment, a query $\mathsf{Test}(i^*, s^*)$ is received and $(i^*, s^*) \neq (i, s)$. Thus:

$$\Pr(break_0) \leq n_P n_S \cdot \Pr(break_1)$$

Game 2 In this game, we guess the party of the intended partner of the test session π_i^s, and abort if $\pi_i^s.pid \neq j$. Thus:

$$\Pr(break_1) \leq n_P \cdot \Pr(break_2)$$

Game 3 In this game, we abort if the session π_i^s sets the status $\pi_i^s.\alpha \leftarrow \texttt{reject}$. Note that by **Game 1** we abort if the Test query is ever issued to a session that is not π_i^s. If the session π_i^s ever reaches the status $\pi_i^s.\alpha \leftarrow \texttt{reject}$, then the $\mathsf{Test}(i, s)$ query will be rejected by the challenger as specified in Figure 1. Note that the difference between the adversary's advantage in **Game 2** and **Game 3** is 0 as the sampling of the test bit b by the challenger only affects the response to the $\mathsf{Test}(i, s)$ query, which is always rejected if $\pi_i^s.\alpha = \texttt{reject}$. Thus:

$$\Pr(break_2) = \Pr(break_3)$$

Game 4 In this game we define an abort event $abort_{\mathsf{accept}}$ that triggers if the status of the test session $\pi_i^s \leftarrow \texttt{accept}$. It is clear then that

$$\Pr(break_3) \leq \Pr(abort_{\mathsf{accept}}) + \Pr(break_4)$$

and additionally that $\Pr(break_4) = {}^1\!/_2$, since all responses to the adversary are identical regardless of the sampling of the test bit b. In the following sequence of games, we show that the probability of the abort event triggering (i.e. $\Pr(abort_{\mathsf{accept}})$) is negligibly close to zero.

Game 5 In this game we replace the computation of C_8 with uniformly random and independent values $\widetilde{C_8}$. This works very similar to **Game 5** of **Case 1** and mostly changes labels. For the same reason as back then, we also split this game into three subhybrids numbered 5a, 5b and 5c.

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

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

In the first case (r_r is unknown to the adversary), we initialize a $\mathsf{prf}^{\mathsf{swap}}$ challenger, query σ_r, 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_r 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 $\hat{r} \leftarrow \mathsf{HKDF}(\sigma_r, r_r)$ and we are in **Game 4**. If the test bit sampled by the $\mathsf{prf}^{\mathsf{swap}}$ challenger is 1, then $\hat{r} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ is a truly random value and we are in **Game 5a**.

For the second case we first establish that r_r, while being (potentially) known to the adversary is still fresh in the sense that $\mathsf{HKDF}(\sigma_r, r_r)$ has never been evaluated: Since r_r 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 $\frac{n_S}{2^\lambda}$ (which when multiplied by the number of sessions results in the famous approximation of the birthday-bound $\frac{n_S^2}{2^\lambda}$).

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

$$\begin{aligned}&\Pr(abort_{\mathsf{accept}})\\ &\leq \frac{n_S}{2^\lambda} + \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_{5a})\end{aligned}$$

In **Game 5b** we replace the computation of $\texttt{shk3}$ by sampling the value uniformly at random from the space of shared secrets of the KEM and ignoring the second output of $\mathsf{CCAKEM.Enc}(\texttt{spk}_r)$. 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 2**, 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 3**, 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 $\texttt{pk}^*$ 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 $\pi_i^s.\alpha \leftarrow \texttt{reject}$ and abort if $\pi_i^s.\alpha \leftarrow \texttt{accept}$. Thus we will not need to reveal the private key $\texttt{sk}^*$ 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 our version of WireGuard, the long-term private keys are used to compute the following:

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

(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 $\mathsf{CCAKEM.Dec}(\texttt{ctX})$, (where $\texttt{ctX}$ is the relevant encapsulation) which will output $\texttt{shkX}$ using the CCA challenger's internal knowledge of $\texttt{sk}^*$.

During session i we request a challenge consisting of a ciphertext and a candidate shared secret $(\texttt{c}^*, \texttt{k}^*)$ from the IND-CCA challenger and use those values in place of $\texttt{ct3}$

and $\texttt{shk3}$. 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 $\texttt{k}^*$ is indeed the shared secret encapsulated in $\texttt{c}^*$ and we are in **Game 5a**.
- If the test bit sampled by the IND-CCA challenger is 1, then $\texttt{k}^*$ is not the shared secret encapsulated in $\texttt{c}^*$ 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:

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

In **Game 5c** we replace the values of C_8 with uniformly random and independent values $\widetilde{C_8} \xleftarrow{\$} \{0,1\}^{|\mathsf{KDF}_1|}$ (where $\{0,1\}^{|\mathsf{KDF}_1|}$ is the output space of the KDF_1) used in the protocol execution of the test session. Specifically, we initialize a $\mathsf{prf}^{\mathsf{swap}}$ challenger and query $\texttt{shk3}$, and use the output $\widetilde{C_8}$ from the $\mathsf{prf}^{\mathsf{swap}}$ challenger to replace the computation of C_8. Since by **Game 5b**, $\texttt{shk3}$ is a uniformly random and independent value, this replacement is sound. If the test bit sampled by the prf challenger is 0, then $\widetilde{C_8} \leftarrow \mathsf{HKDF}(C_7, \texttt{shk3})$ and we are in **Game 5b**. If the test bit sampled by the prf challenger is 1, then $\widetilde{C_8} \xleftarrow{\$} \{0,1\}^{|\mathsf{KDF}_1|}$ 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 HKDF, and we find:

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

Game 6 In this game we replace the values C_9, tmp, κ_9 with uniformly random and independent values $\widetilde{C_9}, \widetilde{tmp}, \widetilde{\kappa_9} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ (where $\{0,1\}^{|\mathsf{HKDF}|}$ is the output space of HKDF) used in the protocol execution of the test session. Specifically, we initialize a PRF challenger and issue the challenge psk to it, and use the output $\widetilde{C_9}, \widetilde{tmp}, \widetilde{\kappa_9}$ from the PRF challenger to replace the computation of C_9, tmp, κ_9. Since by **Game 5c**, $\widetilde{C_8}$ is a uniformly random and independent value, this replacement is sound. If the test bit sampled by the prf challenger is 0, then $\widetilde{C_9}, \widetilde{tmp}, \widetilde{\kappa_4} \leftarrow \mathsf{HKDF}(C_8, psk)$ and we are in **Game 5c**. If the test bit sampled by the prf challenger is 1, then $\widetilde{C_9}, tmp, \widetilde{\kappa_9} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 6**. Thus any adversary $\mathcal{A}$ capable of distinguishing this change can be turned into a successful adversary against the PRF assumption, and we find:

$$\Pr(break_{5c}) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_6)$$

Game 7 In this game we replace the values $C_{10}, \kappa_{10} \leftarrow \mathsf{HKDF}(\widetilde{C_9}, \emptyset)$ with uniformly random and independent values $\widetilde{C_{10}}, \widetilde{\kappa_{10}} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ (where $\{0,1\}^{|\mathsf{HKDF}|}$ is the output space of the HKDF) used in the protocol execution of the test session. Specifically, we initialize a PRF challenger and issue the challenge query $\emptyset$ to it, and use the output $\widetilde{C_{10}}, \widetilde{\kappa_{10}}$ from the prf challenger to replace the computation of C_{10}, κ_{10}. Since by **Game 6**, $\widetilde{C_9}$ is a uniformly random and independent value, this replacement is sound. If the test bit sampled by the PRF challenger is 0, then $\widetilde{C_{10}}, \widetilde{\kappa_{10}} \leftarrow \mathsf{HKDF}(\widetilde{C_9}, \emptyset)$ and we are in **Game 6**. If the test bit sampled by the prf challenger is 1, then $\widetilde{C_{10}}, \widetilde{\kappa_{10}} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 7**. Thus any adversary $\mathcal{A}$ capable of distinguishing this change can be turned into a successful adversary against the prf assumption, and we find:

$$\Pr(break_6) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_7)$$

Game 8 In this game, we add an abort event $abort_{\mathsf{decrypt}}$ that triggers if the test session π_i^s receives a ciphertext $\texttt{conf}$ in the $\texttt{SenderConf}$ message that decrypts correctly. Since the test session π_i^s will only reach an accept status if $\texttt{conf}$ decrypts correctly, it follows that

$$\Pr(break_7) \leq \Pr(abort_{\mathsf{decrypt}}).$$

Now we show that the probability of $abort_{\mathsf{decrypt}}$ is negligibly close to zero. We do so by initializing an aead-auth challenger to decrypt $\texttt{SenderConf.conf}$ ciphertexts in the test session π_i^s. We note that by **Game 7** that $\widetilde{\kappa_9}$ is a uniformly random and independent value, and since the aead challenger samples the internal aead key from the same distribution thus this change is undetectable. If π_i^s receives a ciphertext $\texttt{conf}$ in the $\texttt{SenderConf}$ message that decrypts correctly and the aead encryption oracle has not been queried, then it follows that this ciphertext $\texttt{conf}$ is a forged ciphertext, breaking the auth security of the AEAD scheme. Thus, we find that:

$$\Pr(abort_{\mathsf{decrypt}}) \leq \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda).$$

Thus we find that the probability of $\mathcal{A}$ in causing a session π_i^s with $\rho = \texttt{resp}$ to reach $\pi_i^s.\alpha \leftarrow \texttt{accept}$ and triggering $break_{\mathsf{accept}}$ to be:

$$\begin{aligned}\Pr(abort_{\mathsf{accept}}) \leq \Big(&\frac{n_S}{2^\lambda} + \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda) \\ &+ \ 3 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ &+ \ 2 \cdot \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)\Big).\end{aligned}$$

We can finally show that

$$\begin{aligned}\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_2}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda) \leq n_P^2 n_S \Big(&\frac{n_S}{2^\lambda} + \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda) \\ &+ 3 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ &+ \ 2 \cdot \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)\Big).\end{aligned}$$

3) Case 3: Test session with contributive keyshare partner: By the case definition and the definition of the cleanness predicate $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ there are five ways that the cleanness predicate could potentially be upheld[5]: $\mathcal{A}$ has issued $\mathsf{Test}(i,s)$ where $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}(\pi_i^s)$ is upheld and has a contributive keyshare session π_j^t and either:

1) A pre-shared key exists between party P_i and the test session's intended partner, *and* $\mathcal{A}$ did not issue $\mathsf{CorruptPSK}(i,j)$, or $\mathsf{CorruptPSK}(j,i)$. We denote with $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.1}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda)$ the advantage of $\mathcal{A}$ in winning in this case and refer to this as the *pre-shared subcase*.
2) $\mathcal{A}$ did not issue $\mathsf{CorruptEPK}(i,s)$ or $\mathsf{CorruptEPK}(j,t)$. We denote with $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.2}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda)$ the advantage $\mathcal{A}$ and refer to this as the *ephemerals subcase*.
3) $\mathcal{A}$ did not issue $\mathsf{CorruptEPK}(i,s)$ or $\mathsf{CorruptASK}(j)$. We denote with $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.3}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda)$ the advantage of $\mathcal{A}$ and refer to this as the *ephemeral/long-term subcase*.
4) $\mathcal{A}$ did not issue $\mathsf{CorruptASK}(i)$ or $\mathsf{CorruptEPK}(j,t)$. We denote with $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.4}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda)$ the advantage of $\mathcal{A}$ and refer to this as the *long-term/ephemeral subcase*.
5) $\mathcal{A}$ did not issue $\mathsf{CorruptASK}(i)$ or $\mathsf{CorruptASK}(j)$. We denote with $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.5}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda)$ the advantage of $\mathcal{A}$ and refer to this as the *long-terms subcase*.

Since at least one of these subcases must apply, then:

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_3}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda) = \max\left\{\begin{array}{l}\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.1}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda),\\ \mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.2}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda),\\ \mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.3}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda),\\ \mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.4}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda),\\ \mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.5}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda)\end{array}\right\}$$

We now turn to bounding the advantage of the adversary $\mathcal{A}$ in each of the subcases, and show that if the advantage of $\mathcal{A}$ in each subcase is negligible, then so too is the advantage of $\mathcal{A}$ in Case 3.

Case 3.1: The Preshared Subcase: In this subcase we assume that the cleanness predicate is upheld such that a pre-shared secret between the test session and its honest contributive keyshare session exists, and has not been corrupted. Due to the definition of Case 3, we know that such an honest contributive keyshare session exists. In what follows, we show that the probability of $\mathcal{A}$ in winning the key-indistinguishability game is negligible.

a) ***Proof sketch****:* We begin by guessing the index of the test session, and add an abort event that occurs if a Test query is directed to a session that does not have the index of the guessed session, and similarly, guess the index of the contributive keyshare partner. We then replace the value of C_9, tmp, κ_9 with uniformly random values $\widetilde{C_9}, \widetilde{tmp}, \widetilde{\kappa_9}$, and note that by the subcase definition and the $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$, that the adversary cannot issue either a $\mathsf{CorruptPSK}(i,j)$ or $\mathsf{CorruptPSK}(j,i)$ query. Since the psk shared between the two parties is a uniformly random and independent value, we argue that any adversary capable of distinguishing this replacement would be able to break the PRF assumption. In a similar fashion, we replace the values C_{10}, κ_{10} with uniformly random and independent values $\widetilde{C_{10}}, \widetilde{\kappa_{10}}$, and argue that since $\widetilde{C_9}$ was already independent from the protocol execution that this replacement was sound and that any adversary capable of distinguishing this change would be able to be turned into a adversary against PRF security. In the final game and with a similar argument, we replace tk_i, tk_r with uniformly random and independent values, based on the PRF security of HKDF. Since the session keys are now uniformly random and independent of the test bit b sampled by the challenger, the advantage of $\mathcal{A}$ against the eCK-PFS-PSK-security of the modified WireGuard protocol in the pre-shared key subcase is negligible. **Game 0** This is a standard eCK-PFS-PSK with cleanness predicate $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ upheld as in Definition 3. Thus

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.1}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda) = \Pr(break_0).$$

Game 1 In this game, we guess the index (i,s) of the Test session π_i^s and abort if, during the experiment, a query $\mathsf{Test}(i^*,s^*)$ is issued such that $(i^*,s^*) \neq (i,s)$. Thus

$$\Pr(break_0) \le n_P n_S \cdot \Pr(break_1).$$

Game 2 In this game, we guess the index (j,t) of the contributive keyshare session π_j^t (which exists by the Case 3 definition) and abort if during the experiment, a query $\mathsf{Test}(i,s)$ is issued when the contributive keyshare session $\pi_{t^*}^{j^*}$ exists such that $(j^*,t^*) \neq (j,t)$. Thus

$$\Pr(break_1) \le n_P n_S \cdot \Pr(break_2).$$

Game 3 In this game, we replace the computation of C_9, tmp, κ_9 with uniformly random values $\widetilde{C_9}, \widetilde{tmp}, \widetilde{\kappa_9}$ in the execution of session π_i^s and its partner session π_j^t. We do so by interacting with a $\mathsf{prf}^{\mathsf{swap}}$ challenger in the following way: When it is time to compute $C_9, tmp, \kappa_9 \leftarrow \mathsf{HKDF}(C_8, psk)$ we instead initialize a $\mathsf{prf}^{\mathsf{swap}}$ challenger and query C_8. We note that by the cleanness predicate and the preconditions of this subcase that psk is a uniformly random value that will not be revealed by $\mathcal{A}$ through a $\mathsf{CorruptPSK}(i,j)$ query, and thus this replacement is sound. If the random bit b sampled by the prf challenger is 0, then we are in **Game 2**. If the random bit b sampled by the $\mathsf{prf}^{\mathsf{swap}}$ challenger is 1, then we are in **Game 3**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the $\mathsf{prf}^{\mathsf{swap}}$ security of HKDF and thus

$$\Pr(break_2) \le \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_3).$$

Game 4 Similarly to the previous game, we replace the computation of C_{10} with a uniformly random value $\widetilde{C_{10}}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_{10} \leftarrow \mathsf{HKDF}(C_9, \emptyset)$ we instead initialize a prf challenger and query it with the empty string $\emptyset$. We note that by **Game 3** that C_9 is a uniformly random value independent from the protocol execution, and as such the replacement is sound.If the random bit b sampled by the prf challenger is 0,

[5]Note that we do not make explicit in each condition that $\mathcal{A}$ has not issued either a $\mathsf{Reveal}(i,s)$ or $\mathsf{Reveal}(j,t)$ query

then we are in **Game 3**. If the random bit b sampled by the prf challenger is 1, then we are in **Game 4**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF, and thus

$$\Pr(break_3) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{prf},\mathcal{A}}(\lambda) + \Pr(break_4).$$

Game 5 As in previous games, we replace the values $tk_i, tk_r \leftarrow \mathsf{HKDF}(\widetilde{C_{10}}, \emptyset)$ computed by the challenger in the execution of the test session and its honest contributive keyshare session partner π_j^t with uniformly random values $\widetilde{tk_i}, \widetilde{tk_r}$. We do so by interacting with a prf challenger in the following way: When it is time to compute tk_i, tk_r in the appropriate sessions, we instead initialize a prf challenger and query it with the empty string $\emptyset$. We note that by **Game 4** that $\widetilde{C_{10}}$ is a uniformly random value independent from the protocol execution, and as such the replacement is sound.If the random bit sampled by the prf challenger if 0, then we are in **Game 4**, but otherwise the output of the prf challenger $\widetilde{tk_i}, \widetilde{tk_r}$ is uniformly random and independent and we are in **Game 5**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF, and thus

$$\Pr(break_4) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_5).$$

Since the response to the $\mathsf{Test}(i,s)$ query is (in **Game 5**) uniformly random and independent regardless of the value of the test bit b, then the adversary's success in winning the key-indistinguishability game is reduced to simply guessing and thus:

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.1}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda) \leq n_P^2 n_S^2 \Big(\quad 2 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{prf^{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \Big).$$

Case 3.2: The Ephemerals Subcase: In this subcase we know that (by the definition of $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ and the subcase preconditions) that the session π_i^s such that the $\mathsf{Test}(i,s)$ session will be queried has an honest contributive keyshare session π_j^t and that $\mathsf{CorruptEPK}(i,s)$ and $\mathsf{CorruptEPK}(j,t)$ queries have not been issued during the execution of the experiment. We now show that in this subcase, the adversary's probability in winning the key-indistinguishability game is negligible.

Game 0 This is a standard eCK-PFS-PSK game with cleanness predicate $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ upheld. Thus we have

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda) = \Pr(break_0).$$

Game 1 In this game, we guess the index (i,s) of the Test session π_i^s and abort if, during the experiment, a query $\mathsf{Test}(i^*, s^*)$ is issued such that $(i^*, s^*) \neq (i,s)$. Thus

$$\Pr(break_0) \leq n_P n_S \cdot \Pr(break_1).$$

Game 2 In this game, we guess the index (j,t) of the honest partner session π_j^t (which we know exists by the Case 3 definition) and abort if, during the experiment, a query $\mathsf{Test}(i,s)$ is issued if the contributive keyshare session $\pi_{t^*}^{j^*}$ exists such that $(j^*, t^*) \neq (i,s)$. Thus

$$\Pr(break_1) \leq n_P n_S \cdot \Pr(break_2).$$

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 `ct2` 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 $\mathtt{epk}_i$ 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):

- $C_7 \leftarrow \mathsf{HKDF}(c_2, \mathtt{shk2})$

Here we can replace `shk2` with the supposed shared key $\mathtt{k}^*$ from the IND-CPA-challenger. When the test bit sampled by the IND-CPA challenger is 0, then $\mathtt{k}^*$ 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 $\mathtt{k}^* \xleftarrow{\$} \mathcal{K}_{\mathsf{CPAKEM}}$ 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

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

Game 4 In this game, we replace the computation of C_7 with a uniformly random value $\widetilde{C_7}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a $\mathsf{prf^{swap}}$ challenger in the following way: When it is time to compute $C_7 \leftarrow \mathsf{HKDF}(C_6, \mathtt{shk2})$ we instead initialize a HKDF challenger and query it with C_6. We note that by **Game 3** that `shk2` is a uniformly random value and independent value, and thus this replacement is sound. If the random bit b sampled by the $\mathsf{prf^{swap}}$ challenger is 0, then $\widetilde{C_7} \leftarrow \mathsf{HKDF}(C_6, \mathtt{shk2})$ and we are in **Game 3**. If the random bit b sampled by the prf challenger is 1, then $\widetilde{C_7} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 4**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus

$$\Pr(break_3) \leq \mathsf{Adv}^{\mathsf{prf^{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_4).$$

Game 5 Similarly to the previous game, we replace the computation of C_8 with a uniformly random value $\widetilde{C_8}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_8 \leftarrow \mathsf{HKDF}(\widetilde{C_7}, \texttt{shk3})$ we instead initialize a prf challenger and query it with $\texttt{shk3}$. We note that by **Game 4** that $\widetilde{C_7}$ is a uniformly random value and independent value, and thus this replacement is sound. If the random bit b sampled by the prf challenger is 0, then $C_8 \leftarrow \mathsf{PRF}(\widetilde{C_7}, \texttt{shk3})$ and we are in **Game 4**. If the random bit b sampled by the prf challenger is 1, then $\widetilde{C_8} \xleftarrow{\$} \{0,1\}^{|\mathsf{PRF}|}$ and we are in **Game 5**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF, and thus

$$\Pr(break_4) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_5).$$

Game 6 As in previous games, we replace the computation of C_9, tmp, κ_9 with uniformly random values $\widetilde{C_9}, \widetilde{tmp}, \widetilde{\kappa_9}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_9, tmp, \kappa_9 \leftarrow \mathsf{HKDF}(\widetilde{C_8}, psk)$ we instead initialize a prf challenger and query it with psk. We note that by **Game 5** that $\widetilde{C_8}$ is a uniformly random value and independent value, and thus this replacement is sound. If the random bit b sampled by the prf challenger is 0, then we are in **Game 5**. If the random bit b sampled by the prf challenger is 1, then we are in **Game 6**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF, and thus

$$\Pr(break_5) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_6).$$

Game 7 As in previous games, we replace the computation of C_{10} with a uniformly random value $\widetilde{C_{10}}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_{10} \leftarrow \mathsf{HKDF}(C_9, \emptyset)$ we instead initialize a prf challenger and query it with the empty string $\emptyset$. We note that by **Game 6** that $\widetilde{C_9}$ is a uniformly random value independent from the protocol execution, and as such the replacement is sound. If the random bit b sampled by the prf challenger is 0, then we are in **Game 6**. If the random bit b sampled by the prf challenger is 1, then we are in **Game 7**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF, and thus

$$\Pr(break_6) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_7)$$

Game 8 As in previous games, we replace the values $tk_i, tk_r \leftarrow \mathsf{HKDF}(\widetilde{C_{10}}, \emptyset)$ computed by the challenger in the execution of the test session and its honest contributive keyshare session partner π_j^t with uniformly random values $\widetilde{tk_i}, \widetilde{tk_r}$. We do so by interacting with a prf challenger in the following way: When it is time to compute tk_i, tk_r in the appropriate sessions, we instead initialize a prf challenger and query it with the empty string $\emptyset$. We note that by **Game 4** that $\widetilde{C_{10}}$ is a uniformly random value independent from the protocol execution, and as such the replacement is sound. If the random bit sampled by the prf challenger if 0, then we are in **Game 7**, but otherwise the output of the prf challenger $\widetilde{tk_i}, \widetilde{tk_r}$ is uniformly random and independent and we are in **Game 8**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF, and thus

$$\Pr(break_7) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_8)$$

Since the response to the $\mathsf{Test}(i,s)$ query issued by the adversary is, in **Game 8**, uniformly random and independent of the test bit b sampled by the challenger, then the adversary's success in winning the key-indistinguishability game is reduced to simply guessing and thus:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK}, C_{3.2}}_{\mathsf{pqWG}, n_P, n_S, \mathcal{A}}(\lambda) \leq \; n_P^2 n_S^2 \Big(&\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{CPAKEM},\mathcal{A}}(\lambda) \\ &+ 4 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \Big)\end{aligned}$$

Case 3.3: The Ephemeral/Long-term Subcase: In this subcase we know that (by the definition of $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ and the subcase preconditions) that the session π_i^s such that the $\mathsf{Test}(i,s)$ session will be queried has an honest contributive keyshare session π_j^t and that $\mathsf{CorruptEPK}(i,s)$ and $\mathsf{CorruptASK}(j)$ queries have not been issued during the execution of the experiment. Note that in our proof we set that the test session has role `init` and the partner session has role `resp`, but the case where the test session has role `resp` and the partner session has role `init` follows analogously. In what follows, we show that in this subcase, the adversary's probability in winning the key-indistinguishability game is negligible under certain security assumptions.**Game 0** This is a standard eCK-PFS-PSK game with cleanness predicate $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ upheld. Thus we have:

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK}}_{\mathsf{pqWG}, n_P, n_S, \mathcal{A}}(\lambda) = \Pr(break_0)$$

Game 1 In this game, we guess the index (i,s) of the Test session π_i^s and abort if, during the experiment, a query $\mathsf{Test}(i^*, s^*)$ is issued such that $(i^*, s^*) \neq (i,s)$. Thus:

$$\Pr(break_0) \leq n_P n_S \cdot (\Pr(break_1))$$

Game 2 In this game, we guess the index (j,t) of the honest partner session π_j^t (which we know exists by the Case 3 definition) and abort if, during the experiment, a query $\mathsf{Test}(i,s)$ is issued if the contributive keyshare session $\pi_{t^*}^{j^*}$ exists such that $(j^*, t^*) \neq (i,s)$. Thus:

$$\Pr(break_1) \leq n_P n_S \cdot (\Pr(break_2))$$

Game 3 In this game we replace the computation of C_3, κ_3 with uniformly random and independent values $\widetilde{C_3}, \widetilde{\kappa_3}$. This is practically identical to the **Game 5** of case1, including the subhybrids.

In **Game 3a** we replace the value $\hat{r} := \mathsf{HKDF}(\sigma_i, r_i)$ passed to CCAKEM.Enc for the computation of `ct1` and `shk1` with a random bitstring $\hat{r}'$.

To show that this replacement is sound, we replace the value of $\hat{r}$ with a uniformly random and independent value $\hat{r}' \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ used in the protocol execution of the test session. Specifically, we initialize a $\mathsf{prf}^{\mathsf{swap}}$ challenger and 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 $\hat{r} \leftarrow \mathsf{HKDF}(\sigma_i, r_i)$ and we are in **Game 2**. If the test bit sampled by the $\mathsf{prf}^{\mathsf{swap}}$ challenger is 1, then $\hat{r} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ is a truly random value and we are in **Game 3a**.

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 HKDF, and we find:

$$\Pr(break_2) \leq \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr\left(break_{3a}\right)$$

In **Game 3b** 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 2**, 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 1**, 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 `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 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 `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 our version of WireGuard, the long-term private keys are used to compute the following:

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

(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 CCAKEM.Dec(`ctX`), (where `ctX` is the relevant encapsulation) which will output `shkX` using the CCA challenger's internal knowledge of `sk`*.

During session i we request a challenge consisting of a ciphertext and a candidate shared secret (`c`*, `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 `k`* is indeed the shared secret encapsulated in `c`* and we are in **Game 3a**.
- If the test bit sampled by the IND-CCA challenger is 1, then `k`* is not the shared secret encapsulated in `c`* but sampled uniformly at random from the space of shared secrets and we are in **Game 3b**.

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:

$$\Pr(break_{3a}) \leq \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda) + \Pr(break_{3b})$$

In **Game 3c** we replace the values of C_3, κ_3 with uniformly random and independent values $\widetilde{C_3}, \widetilde{\kappa_3} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ (where $\{0,1\}^{|\mathsf{HKDF}|}$ is the output space of the HKDF) 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 3b**, `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 $\widetilde{C_3}, \widetilde{\kappa_3} \leftarrow \mathsf{HKDF}(C_2, \mathtt{shk1})$ and we are in **Game 3b**. If the test bit sampled by the $\mathsf{prf}^{\mathsf{swap}}$ challenger is 1, then $\widetilde{C_3}, \widetilde{\kappa_3} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 3c**.

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 HKDF, and we find:

$$\Pr(break_{3b}) \leq \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_{3c})$$

Game 4 In this game, we replace the computation of C_4, κ_4 with uniformly random values $\widetilde{C_4}, \widetilde{\kappa_4}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_4, \kappa_4 \leftarrow \mathsf{HKDF}(\widetilde{C_3}, psk)$ we instead initialize a prf challenger and query it with psk. We note that by **Game 3c** that $\widetilde{C_3}$ is a uniformly random value and independent value, and thus this replacement is sound. If the random bit b sampled by the prf challenger is 0, then we are in **Game 3c**. If the random bit b sampled by the prf challenger is 1, then we are in **Game 4**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_{3c}) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_4)$$

Game 5 In this game, we replace the computation of C_6 with a uniformly random value $\widetilde{C_6}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_6 \leftarrow \mathsf{HKDF}(\widetilde{C_4}, \texttt{ct2})$ we instead initialize a prf challenger and query it with $\texttt{ct2}$. We note that by **Game 4** that $\widetilde{C_4}$ is a uniformly random value and independent value, and thus this replacement is sound. If the random bit b sampled by the prf challenger is 0, then we are in **Game 4**. If the random bit b sampled by the prf challenger is 1, then we are in **Game 5**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_4) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_5)$$

Game 6 In this game, we replace the computation of C_7 with a uniformly random value $\widetilde{C_7}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_7 \leftarrow \mathsf{HKDF}(\widetilde{C_6}, \texttt{shk2})$ we instead initialize a prf challenger and query it with $\texttt{shk2}$. We note that by **Game 5** that $\widetilde{C_6}$ is a uniformly random value and independent value, and thus this replacement is sound. If the random bit b sampled by the prf challenger is 0, then $\widetilde{C_7} \leftarrow \mathsf{HKDF}(\widetilde{C_6}, \texttt{shk2})$ and we are in **Game 5**. If the random bit b sampled by the prf challenger is 1, then $\widetilde{C_7} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 6**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_5) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_6)$$

Game 7 Similarly to the previous game, we replace the computation of C_8 with a uniformly random value $\widetilde{C_8}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_8 \leftarrow \mathsf{HKDF}(\widetilde{C_7}, \texttt{shk3})$ we instead initialize a prf challenger and query it with $\texttt{shk3}$. We note that by **Game 6** that $\widetilde{C_7}$ is a uniformly random value and independent value, and thus this replacement is sound. If the random bit b sampled by the prf challenger is 0, then $C_8 \leftarrow \mathsf{HKDF}(\widetilde{C_7}, \texttt{shk3})$ and we are in **Game 6**. If the random bit b sampled by the prf challenger is 1, then $\widetilde{C_8} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 7**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_6) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_7)$$

Game 8 In this game, we replace the computation of C_9, tmp, κ_9 with uniformly random values $\widetilde{C_9}, \widetilde{tmp}, \widetilde{\kappa_9}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_9, tmp, \kappa_9 \leftarrow \mathsf{HKDF}(\widetilde{C_8}, psk)$ we instead initialize a prf challenger and query it with psk. We note that by **Game 7** that $\widetilde{C_8}$ is a uniformly random value and independent value, and thus this replacement is sound. If the random bit b sampled by the prf challenger is 0, then we are in **Game 7**. If the random bit b sampled by the prf challenger is 1, then we are in **Game 8**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_7) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_8)$$

Game 9 Similarly to the previous game, we replace the computation of C_{10} with a uniformly random value $\widetilde{C_{10}}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_{10} \leftarrow \mathsf{HKDF}(C_9, \emptyset)$ we instead initialize a prf challenger and query it with the empty string $\emptyset$. We note that by **Game 8** that $\widetilde{C_9}$ is a uniformly random value independent from the protocol execution, and as such the replacement is sound If the random bit b sampled by the prf challenger is 0, then we are in **Game 8**. If the random bit b sampled by the prf challenger is 1, then we are in **Game 9**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_8) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_9)$$

Game 10 Similarly to the previous games, we replace the values $tk_i, tk_r \leftarrow \mathsf{HKDF}(\widetilde{C_{10}}, \emptyset)$ computed by the challenger in the execution of the test session and its honest contributive keyshare session partner π_j^t with uniformly random values $\widetilde{tk_i}, \widetilde{tk_r}$. We do so by interacting with a prf challenger in the following way: When it is time to compute tk_i, tk_r in the appropriate sessions, we instead initialize a prf challenger and query it with the empty string $\emptyset$. We note that by **Game 9** that $\widetilde{C_{10}}$ is a uniformly random value independent from the protocol execution, and as such the replacement is sound. If the random bit sampled by the prf challenger if 0, then we are in **Game 9**, but otherwise the output of the prf challenger $\widetilde{tk_i}, \widetilde{tk_r}$ is uniformly random and independent and we are in **Game 10**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_9) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_{10})$$

Since the response to the $\mathsf{Test}(i, s)$ query issued by the adversary is, in **Game 10**, uniformly random and independent of the test bit b sampled by the challenger, then the adversary's success in winning the key-indistinguishability game is reduced to simply guessing and thus:

$$\Pr(break_{10}) = 1/2$$

$$\mathsf{Adv}^{\text{eCK-PFS-PSK},C_{3.3}}_{\text{pqWG},n_P,n_S,\mathcal{A}}(\lambda) \leq n_P^2 n_S^2 \Big(\mathsf{Adv}^{\text{IND-CCA}}_{\text{CCAKEM},\mathcal{A}}(\lambda) + 7 \cdot \mathsf{Adv}^{\text{prf}}_{\text{HKDF},\mathcal{A}}(\lambda) + 2 \cdot \mathsf{Adv}^{\text{prf}^{\text{swap}}}_{\text{HKDF},\mathcal{A}}(\lambda) \Big)$$

Case 3.4: The Long-term/Ephemeral Subcase: In this subcase we know that (by the definition of the cleanness predicate $\mathsf{clean}_{\text{eCK-PFS-PSK}}$ and the subcase preconditions) that the session π_i^s such that the $\mathsf{Test}(i,s)$ session will be queried has an honest contributive keyshare session π_j^t and that $\mathsf{CorruptASK}(i)$ and $\mathsf{CorruptEPK}(j,t)$ queries have not been issued during the execution of the experiment. Note that in our proof we set that the test session has role `init` and the partner session has role `resp`, but the case where the test session has role `resp` and the partner session has role `init` follows analogously. In what follows, we show that in this subcase, the adversary's probability in winning the key-indistinguishability game is negligible under certain security assumptions.**Game 0** This is a standard eCK-PFS-PSK game with cleanness predicate $\mathsf{clean}_{\text{eCK-PFS-PSK}}$ upheld. Thus we have:

$$\mathsf{Adv}^{\text{eCK-PFS-PSK}}_{\text{pqWG},n_P,n_S,\mathcal{A}}(\lambda) = \Pr(break_0)$$

Game 1 In this game, we guess the index (i,s) of the Test session π_i^s and abort if, during the experiment, a query $\mathsf{Test}(i^*,s^*)$ is issued such that $(i^*,s^*) \neq (i,s)$. Thus:

$$\Pr(break_0) \leq n_P n_S \cdot (\Pr(break_1))$$

Game 2 In this game, we guess the index (j,t) of the honest partner session π_j^t (which we know exists by the Case 3 definition) and abort if, during the experiment, a query $\mathsf{Test}(i,s)$ is issued if the contributive keyshare session $\pi_{t^*}^{j^*}$ exists such that $(j^*,t^*) \neq (i,s)$. Thus:

$$\Pr(break_1) \leq n_P n_S \cdot (\Pr(break_2))$$

Game 3 In this game we replace the computation of C_8 with uniformly random and independent values $\widetilde{C_8}$. This works almost identical to **Game 5** of **Case 2** and mostly changes labels.

In **Game 3a** we replace the value $\hat{r} := \mathsf{HKDF}(\sigma_r, r_r)$ passed to CCAKEM.Enc for the computation of `ct1` and `shk1` with a random bitstring $\hat{r}'$.

To show that this replacement is sound, we replace the value of $\hat{r}$ with a uniformly random and independent value $\hat{r}' \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ used in the protocol execution of the test session. Specifically, we initialize a $\mathsf{prf}^{\mathsf{swap}}$ challenger and 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_r 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 $\hat{r} \leftarrow \mathsf{HKDF}(\sigma_r, r_r)$ and we are in **Game 2**. If the test bit sampled by the $\mathsf{prf}^{\mathsf{swap}}$ challenger is 1, then $\hat{r} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ is a truly random value and we are in **Game 3a**.

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 HKDF, and we find:

$$\Pr(break_2) \leq \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\text{HKDF},\mathcal{A}}(\lambda) + \Pr(break_{3a})$$

In **Game 3b** we replace the computation of `shk3` 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 2**, 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 1**, 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 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 our version of WireGuard, the long-term private keys are used to compute the following:

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

(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 `ct3` and `shk3`. 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 3a**.
- 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 3b**.

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:

$$\Pr(break_{3a}) \leq \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda) + \Pr(break_{3b})$$

In **Game 3c** we replace the values of C_83 with uniformly random and independent values $\widetilde{C_8} \xleftarrow{\$} \{0,1\}^{|\mathsf{KDF}_1|}$ (where $\{0,1\}^{|\mathsf{KDF}_1|}$ is the output space of the KDF_1) used in the protocol execution of the test session. Specifically, we initialize a $\mathsf{prf}^{\mathsf{swap}}$ challenger and query $\mathtt{shk3}$, and use the output $\widetilde{C_8}$ from the $\mathsf{prf}^{\mathsf{swap}}$ challenger to replace the computation of C_8. Since by **Game 3b**, $\mathtt{shk3}$ 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 $\widetilde{C_8} \leftarrow \mathsf{HKDF}(C_7, \mathtt{shk3})$ and we are in **Game 3b**. If the test bit sampled by the $\mathsf{prf}^{\mathsf{swap}}$ challenger is 1, then $\widetilde{C_8} \xleftarrow{\$} \{0,1\}^{|\mathsf{KDF}_1|}$ and we are in **Game 3c**.

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 HKDF, and we find:

$$\Pr(break_{3b}) \leq \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_{3c})$$

Game 4 In this game, we replace the computation of C_9, tmp, κ_9 with uniformly random values $\widetilde{C_9}, \widetilde{tmp}, \widetilde{\kappa_9}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_9, tmp, \kappa_9 \leftarrow \mathsf{HKDF}(\widetilde{C_8}, psk)$ we instead initialize a prf challenger and query it with psk. We note that by **Game 3c** that $\widetilde{C_8}$ is a uniformly random value and independent value, and thus this replacement is sound. If the random bit b sampled by the prf challenger is 0, then we are in **Game 3c**. If the random bit b sampled by the prf challenger is 1, then we are in **Game 4**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_{3c}) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_4)$$

Game 5 Similarly to the previous game, we replace the computation of C_{10} with a uniformly random value $\widetilde{C_{10}}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_{10} \leftarrow \mathsf{HKDF}(C_9, \emptyset)$ we instead initialize a prf challenger and query it with the empty string $\emptyset$. We note that by **Game 4** that $\widetilde{C_9}$ is a uniformly random value independent from the protocol execution, and as such the replacement is sound. If the random bit b sampled by the prf challenger is 0, then we are in **Game 4**. If the random bit b sampled by the prf challenger is 1, then we are in **Game 5**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_4) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_5)$$

Game 6 Similarly to the previous games, we replace the values $tk_i, tk_r \leftarrow \mathsf{HKDF}(\widetilde{C_{10}}, \emptyset)$ computed by the challenger in the execution of the test session and its honest contributive keyshare session partner π_j^t with uniformly random values $\widetilde{tk_i}, \widetilde{tk_r}$. We do so by interacting with a prf challenger in the following way: When it is time to compute tk_i, tk_r in the appropriate sessions, we instead initialize a prf challenger and query it with the empty string $\emptyset$. We note that by **Game 5** that $\widetilde{C_{10}}$ is a uniformly random value independent from the protocol execution, and as such the replacement is sound. If the random bit sampled by the prf challenger if 0, then we are in **Game 5**, but otherwise the output of the prf challenger $\widetilde{tk_i}, \widetilde{tk_r}$ is uniformly random and independent and we are in **Game 6**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_5) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_6)$$

Since the response to the $\mathsf{Test}(i, s)$ query issued by the adversary is, in **Game 6**, uniformly random and independent regardless of the test bit b sampled by the challenger, then the adversary's success in winning the key-indistinguishability game is reduced to simply guessing and thus:

$$\Pr(break_6) = 1/2$$

$$\begin{aligned}\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.4}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda) \leq\ & n_P^2 n_S^2 \Big(\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda) \\ & + 3 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ & + 2 \cdot \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \Big)\end{aligned}$$

Case 3.5: The Long-terms Subcase: In this subcase we know that (by the definition of $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ and the subcase preconditions) that the session π_i^s such that the $\mathsf{Test}(i, s)$ session will be queried has an honest contributive keyshare session π_j^t and that $\mathsf{CorruptASK}(i)$ and $\mathsf{CorruptASK}(j)$ queries have not been issued during the execution of the experiment. In what follows, we show that in this subcase, the adversary's probability in winning the key-indistinguishability game is negligible under certain security assumptions.**Game 0** This is a standard eCK-PFS-PSK game with cleanness predicate $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ upheld. Thus we have:

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda) = \Pr(break_0)$$

Game 1 In this game, we guess the index (i, s) of the Test session π_i^s and abort if, during the experiment, a query $\mathsf{Test}(i^*, s^*)$ is issued such that $(i^*, s^*) \neq (i, s)$. Thus:

$$\Pr(break_0) \leq n_P n_S \cdot (\Pr(break_1))$$

Game 2 In this game, we guess the index (j, t) of the honest partner session π_j^t (which we know exists by the Case

3 definition) and abort if, during the experiment, a query $\mathsf{Test}(i, s)$ is issued if the contributive keyshare session $\pi_{t^*}^{j^*}$ exists such that $(j^*, t^*) \neq (i, s)$. Thus:

$$\Pr(break_1) \leq n_P n_S \cdot (\Pr(break_2))$$

Game 3 In this game we replace the computation of C_3, κ_3 with uniformly random and independent values $\widetilde{C_3}, \widetilde{\kappa_3}$. This is mostly identical to **Game 5** of case1 and **Game 3** of **Case 3.3**, except that the first subhybrid **Game 3a** differs slightly because we have to assume that σ_i is uncorrupted instead of r_i.

The case is also special because there exists an alternative way to proof it secure: Instead of basing the security on the security of `shk1`, it would also be possible to base it on `shk3`, in which case the proof would resemble those of **Case 2** and **Case 3.4**. For brevity and because there is little to be gained from describing them here in detail as well, we will refrain from doing so.

In **Game 3a** we replace the values $\hat{r} := \mathsf{HKDF}(\sigma_i, r_i)$ passed to $\mathsf{CCAKEM.Enc}$ for the computation of `ct1` and `shk1` with random bitstrings $\hat{r}'$ in all games of the responder.

We first establish that r_i, while being (potentially) known to the adversary is still fresh in the sense that $\mathsf{HKDF}(\sigma_i, r_i)$ 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 $\frac{n_S}{2^\lambda}$ (which when multiplied by the number of sessions results in the famous approximation of the birthday-bound $\frac{n_S^2}{2^\lambda}$).

We do so by interacting with a prf-challenger in the following way: Whenever it is time to compute to compute $\mathsf{HKDF}(\sigma_r, X)$ for some value X, we instead query the prf-challenger with X and use the output $\widetilde{r}$ from the prf-challenger to replace the computation of $\hat{r}$. 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 $\hat{r} \leftarrow \mathsf{HKDF}(\sigma_i, r_i)$ and we are in **Game 2**. If the test bit sampled by the prf challenger is 1, then $\hat{r} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ 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 3a**.

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

$$\Pr(break_2) \leq \frac{n_S}{2^\lambda} + \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_{3a})$$

Note that this case slightly differs from the previous ones in the same place in that we replace more than just one value with randomness. This is because unlike r_i and r_r, σ_i is used in multiple interactions and thus it becomes necessary to deal with all of them.

In **Game 3b** 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 $\mathsf{IND\text{-}CCA}$-assumption of the used KEM, we interact with an $\mathsf{IND\text{-}CCA}$ challenger in the following way: Note that by **Game 2**, 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 1**, 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 $\mathsf{IND\text{-}CCA}$ 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 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 our version of $\mathsf{WireGuard}$, the long-term private keys are used to compute the following:

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

(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 $\mathsf{IND\text{-}CCA}$ challenger and use those values in place of `ct1` and `shk1`. Given the definition of the $\mathsf{IND\text{-}CCA}$ game, there are two cases:

- If the test bit sampled by the $\mathsf{IND\text{-}CCA}$ challenger is 0, then $\mathtt{k}^*$ is indeed the shared secret encapsulated in $\mathtt{c}^*$ and we are in **Game 3a**.
- If the test bit sampled by the $\mathsf{IND\text{-}CCA}$ 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 3b**.

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

$$\Pr(break_{3a}) \leq \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda) + \Pr(break_{3b})$$

In **Game 3c** we replace the values of C_3, κ_3 with uniformly random and independent values $\widetilde{C_3}, \widetilde{\kappa_3} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ (where $\{0,1\}^{|\mathsf{HKDF}|}$ is the output space of the HKDF) used in the protocol execution of the test session. Specifically, we initialize a $\mathsf{prf}^{\mathsf{swap}}$ challenger and query $\texttt{shk1}$, 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 3b**, $\texttt{shk1}$ 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 $\widetilde{C_3}, \widetilde{\kappa_3} \leftarrow \mathsf{HKDF}(C_2, \texttt{shk1})$ and we are in **Game 3b**. If the test bit sampled by the $\mathsf{prf}^{\mathsf{swap}}$ challenger is 1, then $\widetilde{C_3}, \widetilde{\kappa_3} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 3c**.

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 HKDF, and we find:

$$\Pr(break_{3b}) \leq \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_{3c})$$

Game 4 In this game, we replace the computation of C_6 with a uniformly random value $\widetilde{C_6}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_6 \leftarrow \mathsf{HKDF}(\widetilde{C_4}, \texttt{ct2})$ we instead initialize a prf challenger and query it with $\texttt{ct2}$. We note that by **Game 3c** that $\widetilde{C_4}$ is a uniformly random value and independent value, and thus this replacement is sound. If the random bit b sampled by the prf challenger is 0, then we are in **Game 4**. If the random bit b sampled by the prf challenger is 1, then we are in **Game 4**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_{3c}) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_4)$$

Game 5 In this game, we replace the computation of C_7 with a uniformly random value $\widetilde{C_7}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_7 \leftarrow \mathsf{HKDF}(\widetilde{C_6}, \texttt{shk2})$ we instead initialize a prf challenger and query it with $\texttt{shk2}$. We note that by **Game 4** that $\widetilde{C_6}$ is a uniformly random value and independent value, and thus this replacement is sound. If the random bit b sampled by the prf challenger is 0, then $\widetilde{C_7} \leftarrow \mathsf{HKDF}(\widetilde{C_6}, \texttt{shk2})$ and we are in **Game 4**. If the random bit b sampled by the prf challenger is 1, then $\widetilde{C_7} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 5**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_4) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_5)$$

Game 6 Similarly to the previous game, we replace the computation of C_8 with a uniformly random value $\widetilde{C_8}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_8 \leftarrow \mathsf{HKDF}(\widetilde{C_7}, \texttt{shk3})$ we instead initialize a prf challenger and query it with $\texttt{shk3}$. We note that by **Game 5** that $\widetilde{C_7}$ is a uniformly random value and independent value, and thus this replacement is sound. If the random bit b sampled by the prf challenger is 0, then $C_8 \leftarrow \mathsf{HKDF}(\widetilde{C_7}, \texttt{shk3})$ and we are in **Game 5**. If the random bit b sampled by the prf challenger is 1, then $\widetilde{C_8} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 6**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_5) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_6)$$

Game 7 In this game, we replace the computation of C_9, tmp, κ_9 with uniformly random values $\widetilde{C_9}, \widetilde{tmp}, \widetilde{\kappa_9}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_9, tmp, \kappa_9 \leftarrow \mathsf{HKDF}(\widetilde{C_8}, psk)$ we instead initialize a prf challenger and query it with psk. We note that by **Game 6** that $\widetilde{C_8}$ is a uniformly random value and independent value, and thus this replacement is sound. If the random bit b sampled by the prf challenger is 0, then we are in **Game 6**. If the random bit b sampled by the prf challenger is 1, then we are in **Game 7**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_6) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_7)$$

Game 8 Similarly to the previous game, we replace the computation of C_{10} with a uniformly random value $\widetilde{C_{10}}$ from the same distribution, in the challenger's execution of the test session π_i^s and its partner session π_j^t. We do so by interacting with a prf challenger in the following way: When it is time to compute $C_{10} \leftarrow \mathsf{HKDF}(C_9, \emptyset)$ we instead initialize a prf challenger and query it with the empty string $\emptyset$. We note that by **Game 8** that $\widetilde{C_9}$ is a uniformly random value independent from the protocol execution, and as such the replacement is sound If the random bit b sampled by the prf challenger is 0, then we are in **Game 7**. If the random bit b sampled by the prf challenger is 1, then we are in **Game 8**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_7) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_8)$$

Game 9 Similarly to the previous games, we replace the values $tk_i, tk_r \leftarrow \mathsf{HKDF}(\widetilde{C_{10}}, \emptyset)$ computed by the challenger in the execution of the test session and its honest contributive keyshare session partner π_j^t with uniformly random values $\widetilde{tk_i}, \widetilde{tk_r}$. We do so by interacting with a prf challenger in the following way: When it is time to compute tk_i, tk_r in the appropriate sessions, we instead initialize a prf challenger and query it with the empty string $\emptyset$. We note that by **Game 8** that $\widetilde{C_{10}}$ is a uniformly random value independent from the protocol execution, and as such the replacement is sound. If the random bit sampled by the prf challenger if 0, then we are in **Game 8**, but otherwise the output of the prf challenger $\widetilde{tk_i}, \widetilde{tk_r}$ is uniformly random and independent and we are in **Game 9**. Any adversary $\mathcal{A}$ capable of distinguishing this change in the experiment can be turned into an algorithm against the prf security of HKDF and thus:

$$\Pr(break_8) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_9)$$

Since the response to the $\mathsf{Test}(i, s)$ query issued by the adversary is, in **Game 9**, uniformly random and independent of the test bit b sampled by the challenger, then the adversary's success in winning the key-indistinguishability game is reduced to simply guessing and thus:

$$\Pr(break_9) = 1/2$$

$$\begin{aligned}\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.5}}_{\mathsf{pqWG},n_P,n_S,\mathcal{A}}(\lambda) \leq\ & n_P^2 n_S^2 \Big(\frac{n_S}{2^\lambda} \\ & + \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{CCAKEM},\mathcal{A}}(\lambda) \\ & + 7 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \\ & + \mathsf{Adv}^{\mathsf{prf}^{\mathsf{swap}}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \Big)\end{aligned}$$