A Cryptographic Analysis of the WireGuard Protocol

Benjamin Dowling and Kenneth G. Paterson

Information Security Group
Royal Holloway, University of London
benjamin.dowling@rhul.ac.uk, kenny.paterson@rhul.ac.uk

Abstract. WireGuard (Donenfeld, NDSS 2017) is a recently proposed secure network tunnel operating at layer 3. WireGuard aims to replace existing tunnelling solutions like IPsec and OpenVPN, while requiring less code, being more secure, more performant, and easier to use. The cryptographic design of WireGuard is based on the Noise framework. It makes use of a key exchange component which combines long-term and ephemeral Diffie-Hellman values (along with optional preshared keys). This is followed by the use of the established keys in an AEAD construction to encapsulate IP packets in UDP. To date, WireGuard has received no rigorous security analysis. In this paper, we, rectify this. We first observe that, in order to prevent Key Compromise Impersonation (KCI) attacks, any analysis of WireGuard's key exchange component must take into account the first AEAD ciphertext from initiator to responder. This message effectively acts as a key confirmation and makes the key exchange component of WireGuard a 1.5 RTT protocol. However, the fact that this ciphertext is computed using the established session key rules out a proof of session key indistinguishability for WireGuard's key exchange component, limiting the degree of modularity that is achievable when analysing the protocol's security. To overcome this proof barrier, and as an alternative to performing a monolithic analysis of the entire WireGuard protocol, we add an extra message to the protocol. This is done in a minimally invasive way that does not increase the number of round trips needed by the overall WireGuard protocol. This change enables us to prove strong authentication and key indistinguishability properties for the key exchange component of WireGuard under standard cryptographic assumptions.

1 Introduction

WireGuard: WireGuard [11] was recently proposed by Donenfeld as a replacement for existing secure communications protocols like IPsec and OpenVPN. It has numerous benefits, not least its simplicity and ease of configuration, high performance in software, and small codebase. Indeed, the protocol is implemented in less than 4,000 lines of code, making it relatively easy to audit compared to large, complex and buggy code-bases typically encountered with IPsec and SSL/TLS (on which OpenVPN is based).

From a networking perspective, WireGuard encapsulates IP packets in UDP packets, which are then further encapsulated in IP packets. This is done carefully so as to avoid too much packet overhead. WireGuard also offers a highly simplified version of IPsec's approach to managing which security transforms get applied to which packets: essentially, WireGuard matches on IP address ranges and associates IP addresses with static Diffie-Hellman keys. This avoids much of the complexity associated with IPsec's Security Associations/Security Policy Database mechanisms.

From a cryptographic perspective, WireGuard presents an interesting design. It is highly modular, with a key exchange phase, called the handshake, that is presented as being clearly separated from the subsequent use of the keys in a data transport protocol. A key feature is the one-round (or 1-RTT) nature of the key exchange phase. The key exchange phase runs between an initiator and a responder. It combines long-term and ephemeral Diffie-Hellman values, exclusively using Curve25519 [3], and is built from the Noise protocol framework [22]. In fact, every possible pairwise combination of long-term and ephemeral values is involved in the key computations, presumably in an effort to strengthen security in the face of various combinations of long-term and ephemeral private key compromise. The long-term keys are not supported by a PKI, but are instead assumed to be pre-configured and known to the communicating parties (or trusted on first use, as per SSH). The protocol specification includes an option for using preshared keys between pairs of parties, to augment the DH-based exchange and as a hedge against quantum adversaries. The key exchange phase relies on the BLAKE2s hash function [2] for hashing parts of the transcript, to build HMAC (a hash-based MAC algorithm), and for HKDF (an HMAC-based key derivation function). The data transport protocol uses solely ChaCha20-Poly1305 as specified in RFC 7539 [21] as an AEAD scheme in a lightweight packet format. The AEAD processing incorporates explicit sequence numbers and the receiver uses a standard sliding window technique to deal with packet delays and reorderings.

Security of WireGuard: To the best of our knowledge, with the exception of an initial and high-level symbolic analysis,[1] WireGuard has received no rigorous security analysis. In particular, it has not benefitted from any computational (as opposed to symbolic) proofs. In this paper, we provide such an analysis.

We cannot prove the handshake protocol (as presented in [11]) secure because of an unfortunate reliance on the first message sent in the subsequent data transport protocol to provide entity authentication of the initiator to the responder. Without this extra message, there is a simple Key Compromise Impersonation (KCI) attack, violating a desirable authentication goal of the protocol. This attack was already pointed out by Donenfeld in [11]. Strictly speaking, it means that the key exchange phase is not 1-RTT (as the responder cannot safely send data to the initiator until it has received a verified data transport message from the initiator). We show that there is also an attack on the forward secrecy of the protocol in the same KCI setting, similar to observations made by Krawczyk in [17]. Such an attack recovers session keys rather than breaking an authentication property, and is arguably more serious. However, the attack requires a very particular set of compromise capabilities on the part of the attacker, so we regard it more as a barrier to obtaining strong security proofs than as a practical attack.

On the other hand, if we take the extra message required to prevent the KCI attack of [11] and our new attack into account, it becomes impossible to prove the usual key indistinguishability (KI) property desired of a key exchange protocol (and which, broadly speaking, guarantees that it can be securely composed with subsequent use of the keys [9]). This is because the data transport protocol uses the very keys that we would desire to prove indistinguishable from random to AEAD-protect potentially known plaintexts. Such issues are well-known in the analysis of real-world secure communications protocols – they are endemic, for example, in the analysis of SSL/TLS prior to version 1.3 [20, 15, 18].

There are two basic approaches to solving this problem: analyse the entire protocol (handshake and data transport) as a monolithic entity, or modify the protocol to provide a proper key separation between keys used in the handshake to provide authentication and keys used in the data transport layer. The former approach has been successfully applied (see for example the ACCE framework of [15]) but is complex, requires models highly tuned to the protocol, and results in quite unwieldy proofs. The latter approach makes for easier analysis and highlights better what needs to be considered to be part of the key exchange protocol in order to establish its security, but necessitates changes to the protocol.

Our contributions: In this paper, we adopt the latter approach, making minimally invasive changes to WireGuard to enable us to prove its security.

In more detail, we work with a security model for key exchange based on that of Cremers and Feltz [10] but extended to take into account WireGuard's preshared key option. The model allows us to handle a full range of security properties in one clean sweep, including authentication, regular key indistinguishability, forward security, and KCI attacks (including advanced forms in which key security is considered). The model considers a powerful adversary who is permitted to make every combination of ephemeral and honestly-generated long-term key compromise bar those allowing trivial attacks, and who is able to interact with multiple parties in arbitrary numbers of protocol runs.

We build a description of WireGuard's key exchange phase that takes into account all of its main cryptographic features, including the fine details of its many key derivation and (partial) transcript hashing steps. However, inline with our choice of how to handle the KI/modularity problem, we make a small modification to the handshake protocol, adding an extra flow from initiator to responder which explicitly authenticates one party to the other. This job is currently fulfilled by the first packet from initiator to responder in the data transport protocol. With this modification in place, we are then able to prove the security of WireGuard's key exchange protocol under fairly standard cryptographic assumptions, in the standard model. Specifically, our proof relies on a PRFODH assumption [15, 8] (alternatively, we could have chosen to work with gap-DH and the Random Oracle Model).

Roadmap: Section 2 provides preliminary definitions, mostly focussed on security notions for the base primitives used in WireGuard. Section 3 describes the WireGuard handshake protocol. Section 4 presents the security model for key exchange that we use in Section 5, where our main security result, Theorem 1, can be found. We wrap up with conclusion and future work in Section 6.

2 Preliminaries

Here we formalise the security assumptions that we will be using in our analysis of WireGuard, specifically restating the security assumptions for pseudo-random function (PRF) security, for Authenticated-Encryption with

[1] `https://www.wireguard.com/papers/wireguard-formal-verification.pdf`

Associated Data (AEAD) schemes. We use an asymptotic approach, relying on primitives that are parameterised with a security parameter λ; all our definitions and results can be made concrete at the expense of using extended notation. In later sections, we will suppress all dependence on λ in our naming of primitives to ease the notation.

We let $\mathbb{G} = \langle g \rangle$ denote a finite cyclic group of prime order q that is generated by g. We utilise different typefaces to represent distinct objects: algorithms (such as an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ in a security game), adversarial $\mathsf{Queries}$ (such as Test or Reveal), protocol and per-session *variables* (such as a public-key / secret-key pair (pk, sk)), definitions for security $\mathsf{notions}$ (such as coll or aead), and `constant` protocol values (such as `InitiatorHello` and `ResponderHello`).

Definition 1 (prf Security). *A pseudo-random function family is a collection of deterministic functions* $\mathsf{PRF} = \{\mathsf{PRF}_\lambda : \mathcal{K} \times \mathcal{I} \to \mathcal{O} : \lambda \in \mathbb{N}\}$*, one function for each value of* λ*. Here,* $\mathcal{K}$*,* $\mathcal{I}$*,* $\mathcal{O}$ *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 a PPT 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 a PPT algorithm* $\mathcal{A}$ *in breaking the* pseudo-random function security *of a PRF family* PRF *as* $\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{A}}(\lambda) = |2 \cdot \Pr(b' = b) - 1|$*. We say that* PRF *is secure if for all PPT algorithms* $\mathcal{A}$*,* $\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{A}}(\lambda)$ *is negligible in the security parameter* λ*.*

We define authentication security for AEAD, as introduced by [23], often referred to as integrity for AEAD.

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

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

We now introduce the PRFODH assumption that will be needed for our analysis of WireGuard. The first version of this assumption was introduced by [15] in order to prove the TLS-DHE handshake secure in the standard model. This was subsequently modified in later works analysing real-world protocols, such as TLS-RSA [18], the in-development TLS 1.3 [12, 13], and the Extended Access Control Protocol [7]. This assumption was generalised in [8] in order to capture the different variants of PRFODH in a parameterised way. We give the formulation from [8] verbatim below:

Definition 3 (Generic PRFODH Assumption). *Let* $\mathbb{G}$ *be a cyclic group of order* q *with generator* g *(where* $\mathbb{G}$*,* q *and* g *all implicitly depend on* λ*). Let* $\mathsf{PRF}_\lambda : \mathbb{G} \times \mathcal{M} \to \mathcal{K}$ *be a function from a pseudo-random function family that takes a group element* $k \in \mathbb{G}$ *and a salt value* $m \in \mathcal{M}$ *as inputs, and outputs a value* $y \in \mathcal{K}$*. We define a security notion* $\mathsf{lr\text{-}PRFODH}$ *security which is parameterised by* $\mathsf{l}, \mathsf{r} \in \{\mathsf{n}, \mathsf{s}, \mathsf{m}\}$ *indicating how often the adversary is allowed to query certain "left" and "right" oracles (*ODH_u *and* ODH_v *respectively), where* n *indicates that no query is allowed,* s *that a single query is allowed, and* m *that multiple (polynomially many) queries are allowed to the respective oracle. Consider the following security game between a challenger* $\mathcal{C}$ *and a PPT adversary* $\mathcal{A}$*, both running on input* λ*.*

1. *The challenger $\mathcal{C}$ samples $u \xleftarrow{\$} \mathbb{Z}_q$ and provides $\mathbb{G}, g$ and g^u to the adversary $\mathcal{A}$.*
2. *If $\mathsf{l} = \mathsf{m}$, $\mathcal{A}$ can issue arbitrarily many queries to the oracle ODH_u. These queries are handled as follows: on a query of the form (S, x), the challenger first checks if $S \notin \mathbb{G}$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{PRF}_\lambda(S^u, x)$ and returns y.*
3. *Eventually, $\mathcal{A}$ issues a challenge query x^*. On receipt of this query, $\mathcal{C}$ samples $v \xleftarrow{\$} \mathbb{Z}_q$ and a bit $b \leftarrow \{0,1\}$ uniformly at random. It then computes $y_0 = \mathsf{PRF}_\lambda(g^{uv}, x^*)$ and samples $y_1 \xleftarrow{\$} \{0,1\}^\lambda$ uniformly at random. The challenger returns (g^v, y_b) to $\mathcal{A}$.*
4. *Next, $\mathcal{A}$ may issue (arbitrarily interleaved) queries to oracles ODH_u and ODH_v (depending on l and r). These queries are handled as follows:*
   - $\mathsf{ODH}_\mathbf{u}$ ***oracle.*** *The adversary $\mathcal{A}$ may ask zero (if $\mathsf{l} = \mathsf{n}$), a single (if $\mathsf{l} = \mathsf{s}$), or arbitrarily many (if $\mathsf{l} = \mathsf{m}$) queries to this oracle. On a query of the form (S, x), the challenger first checks if $S \notin \mathbb{G}$ or if $(S, x) = (g^v, x^*)$ and returns $\perp$ if either holds. Otherwise, it returns $y \leftarrow \mathsf{PRF}_\lambda(S^u, x)$.*
   - $\mathsf{ODH}_\mathbf{v}$ ***oracle.*** *The adversary $\mathcal{A}$ may ask zero (if $\mathsf{r} = \mathsf{n}$), a single (if $\mathsf{r} = \mathsf{s}$), or arbitrarily many (if $\mathsf{r} = \mathsf{m}$) queries to this oracle. On a query of the form (T, x), the challenger first checks if $T \notin \mathbb{G}$ or if $(S, x) = (g^u, x^*)$ and returns $\perp$ if either holds. Otherwise, it returns $y \leftarrow \mathsf{PRF}(T^v, x)$.*
5. *At some point, $\mathcal{A}$ stops and outputs $b' \in \{0,1\}$.*

We say that the adversary wins the lr-PRFODH *game if $b' = b$ and define the advantage function*

$$\mathsf{Adv}^{\mathsf{lr\text{-}PRFODH}}_{\mathsf{PRF},\mathbb{G},q,\mathcal{A}}(\lambda) = \Pr(b' = b).$$

We say that the lr-PRFODH *assumption holds if the advantage $\mathsf{Adv}^{\mathsf{lr\text{-}PRFODH}}_{\mathsf{PRF},\mathbb{G},q,\mathcal{A}}(\lambda)$ of any PPT adversary $\mathcal{A}$ is negligible.*

We extend the definition from [8] similarly to [12]: compared to [8] we allow the adversary access to ODH_u and ODH_v oracles *before* the adversary issues the challenge query x^*. This generalisation is necessary in our analysis of WireGuard, because public ephemeral DH values are used to compute a salt value that is used as an input to a PRF during the key computations. We refer to our extension as the *symmetric generic* PRFODH *assumption.*

Definition 4 (Symmetric generic PRFODH Assumption). *Let $\mathbb{G}$ be a cyclic group of order q with generator g (where $\mathbb{G}$, q and g all implicitly depend on λ). Let $\mathsf{PRF}_\lambda : \mathbb{G} \times \mathcal{M} \to \mathcal{K}$ be a function from a pseudo-random function family that takes a group element $k \in \mathbb{G}$ and a salt value $m \in \mathcal{M}$ as input, and outputs a value $y \in \mathcal{K}$. We define a security notion,* sym-lr-PRFODH *security, which is parameterised by: $\mathsf{l}, \mathsf{r} \in \{\mathsf{n}, \mathsf{s}, \mathsf{m}\}$ indicating how often the adversary is allowed to query "left" and "right" oracles (ODH_u and ODH_v), where n indicates that no query is allowed, s that a single query is allowed, and m that multiple (polynomially many) queries are allowed to the respective oracle. Consider the following security game $\mathfrak{D}^{\mathsf{sym\text{-}lr\text{-}PRFODH}}_{\mathsf{PRF},\mathcal{A}}$ between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$, both running on input λ.*

1. *The challenger $\mathcal{C}$ samples $u, v \xleftarrow{\$} \mathbb{Z}_q$ and provides $\mathbb{G}, g, g^u, g^v$ to the adversary $\mathcal{A}$.*
2. *If $\mathsf{l} = \mathsf{m}$, $\mathcal{A}$ can issue arbitrarily many queries to oracle ODH_u, and if $\mathsf{r} = \mathsf{m}$ and $\mathsf{sym} = \mathsf{Y}$ to the oracle ODH_v. These are implemented as follows:*
   - ODH_u*: on a query of the form (S, x), the challenger first checks if $S \notin \mathbb{G}$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{PRF}_\lambda(S^u, x)$ and returns y.*
   - ODH_v*: on a query of the form (T, x), the challenger first checks if $T \notin \mathbb{G}$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{PRF}_\lambda(T^v, x)$ and returns y.*
3. *Eventually, $\mathcal{A}$ issues a challenge query x^*. It is required that, for all queries (S, x) to ODH_u made previously, if $S = g^v$, then $x \neq x^*$. Likewise, it is required that, for all queries (T, x) to ODH_v made previously, if $T = g^u$, then $x \neq x^*$. This is to prevent trivial wins by $\mathcal{A}$. $\mathcal{C}$ samples a bit $b \xleftarrow{\$} \{0,1\}$ uniformly at random, computes $y_0 = \mathsf{PRF}_\lambda(g^{uv}, x^*)$, and samples $y_1 \xleftarrow{\$} \{0,1\}^\lambda$ uniformly at random. The challenger returns y_b to $\mathcal{A}$.*
4. *Next, $\mathcal{A}$ may issue (arbitrarily interleaved) queries to oracles ODH_u and ODH_v. These are handled as follows:*
   - $\mathsf{ODH}_\mathbf{u}$*: on a query of the form (S, x), the challenger first checks if $S \notin \mathbb{G}$ or if $(S, x) = (g^v, x^*)$ and returns $\perp$ if either holds. Otherwise, it returns $y \leftarrow \mathsf{PRF}_\lambda(S^u, x)$.*
   - $\mathsf{ODH}_\mathbf{v}$*: on a query of the form (T, x), the challenger first checks if $T \notin \mathbb{G}$ or if $(S, x) = (g^u, x^*)$ and returns $\perp$ if either holds. Otherwise, it returns $y \leftarrow \mathsf{PRF}_\lambda(T^v, x)$.*
5. *At some point, $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$.*

We say that the adversary wins the sym-lr-PRFODH *game if* $b' = b$ *and define the advantage function*

$$\mathsf{Adv}^{\mathsf{sym\text{-}lr\text{-}PRFODH}}_{\mathsf{PRF},\mathbb{G},q,\mathcal{A}}(\lambda) = |2 \cdot \Pr(b' = b) - 1|.$$

We say that the sym-lr-PRFODH *assumption holds if the advantage* $\mathsf{Adv}^{\mathsf{sym\text{-}lr\text{-}PRFODH}}_{\mathsf{PRF},\mathbb{G},q,\mathcal{A}}(\lambda)$ *of any PPT adversary* $\mathcal{A}$ *is negligible.*

3 The WireGuard Protocol

The WireGuard protocol is, as presented in [11][2], cleanly separated into two distinct phases:

- A *key exchange* or *handshake* phase, where users exchange ephemeral elliptic-curve Diffie-Hellman values, as well as encrypted long-term Diffie-Hellman values and compute AEAD keys; and
- A *data transport* phase, where users may send authenticated and confidential transport data under the previously computed AEAD keys.

The handshake phase is a 1-RTT protocol in which users maintain the following set of variables:

- A randomly-sampled session identifier ID_ρ for each user in the session (i.e we use ID_i to refer to the session identifier of the initiator and for the responder we refer to ID_r).
- An updating seed value C_k, is used to seed the key-derivation function at various points during the key-exchange.
- An updating hash value H_k, is used to hash subsets of the transcript together, to bind the computed AEAD keys to the initial key-exchange.
- A tuple of AEAD keys that are used for confidentiality of the long-term key of the initiator, and to authenticate hash values.
- Long-term elliptic-curve Diffie-Hellman keys g^u, g^v of initiator and responder, respectively.
- Ephemeral elliptic-curve Diffie-Hellman keys g^x, g^y of initiator and responder, respectively.
- Optional long-term preshared key psk.

In Figure 1 we describe the computations required to construct the key exchange messages, which we refer to as `InitiatorHello` and `ResponderHello`. For conciseness, we do not include the chaining steps required to compute the various C_k and H_k values throughout the protocol (we instead list them in Table 1). Nor do we make explicit the verification of the `mac1`, `mac2` MAC values nor the `time`, `zero` AEAD values, but assume that they are correctly verified before deriving the session keys tk_i and tk_r.

k	Seed value C_k	Key κ_k	Hash value H_k
1	$\mathsf{H}(\texttt{label}_1)$	$\emptyset$	$\mathsf{H}(C_1 \| \texttt{label}_2)$
2	$(C_1, g^x, 1)$	$\emptyset$	$\mathsf{H}(H_1 \| g^v)$
3	$(C_2, g^{xv}, 1)$	$(C_2, g^{xv}, 2)$	$\mathsf{H}(H_2 \| g^x)$
4	$(C_3, g^{uv}, 1)$	$(C_3, g^{uv}, 2)$	$\mathsf{H}(H_3 \| \texttt{ltk})$
5	$\emptyset$	$\emptyset$	$\mathsf{H}(H_4 \| \texttt{time})$
6	$(C_4, g^y, 1)$	$\emptyset$	$\mathsf{H}(H_5 \| g^y)$
7	$(C_6, g^{xy}, 1)$	$\emptyset$	$\emptyset$
8	$(C_7, g^{uy}, 1)$	$\emptyset$	$\emptyset$
9	$(C_8, psk, 1)$	$(C_8, psk, 3)$	$\mathsf{H}(H_6 \| \mathsf{KDF}(C_8, psk, 2))$
10	$\emptyset$	$\emptyset$	$\mathsf{H}(H_9 \| \texttt{zero})$

Table 1: A detailed look at the computation of the chaining seed (C_k) and hash (H_k) values, as well as the intermediate AEAD keys (κ_k) used in the WireGuard Key-Exchange protocol. Note that unless otherwise specified, the triples (X, Y, Z) in the table are used in that order as the inputs to a key-derivation function $\mathsf{KDF}(X, Y, Z)$ (so X is used as the keying material, Y is the salt value and Z the index of the output key) to compute the relevant values. Finally, we denote with $\emptyset$ values that are not used during protocol execution.

[2] And in the updated version at `https://www.wireguard.com/papers/wireguard.pdf` that we rely on hereafter.

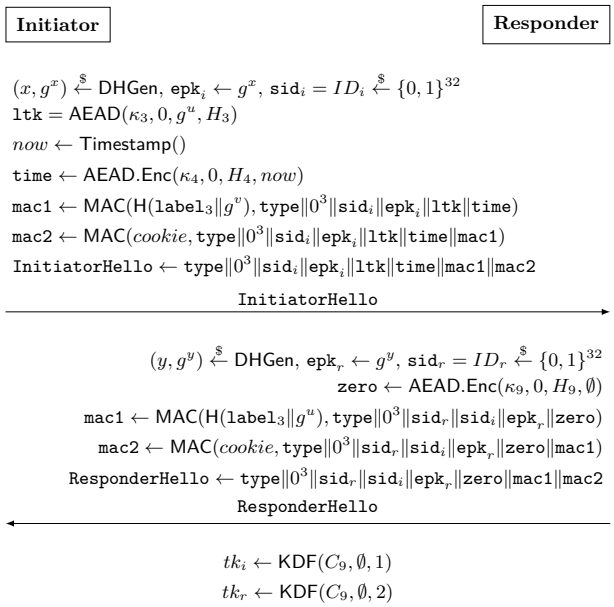

Fig. 1: A brief overview of the WireGuard Key-Exchange Protocol. For more details on the computation of the chaining seed (C_k), hash (H_k) and intermediate key (κ_k) values, refer to Table 1. Note that all verifications of MAC and AEAD values are left implicit, but are obviously crucial to security.

3.1 Remarks on the Protocol

As noted in the introduction (and noted by Donenfeld [11]), it is clear that WireGuard's 1-RTT handshake taken in isolation is not secure in the KCI setting. This is because an attacker in possession of the responder's long-term private DH value v can construct the first protocol message and thence impersonate the initiator *to* the responder. Our attack in Section 5.1 extends this authentication attack to a session key recovery attack. WireGuard protects against this kind of KCI attack by requiring the first data transport message to be sent by the initiator and the responder to check the integrity of this message. Strictly speaking, then, the first data transport message should be regarded as part of the handshake, making it no longer 1-RTT.

An attractive aspect of WireGuard (from a provable security standpoint) is that it is "cryptographically opinionated", meaning that the protocol has no algorithm negotiation functionality — all WireGuard sessions will use Curve25519 for ECDH key exchange, BLAKE2 as the underlying hash function that builds both HMAC and HKDF, and ChaCha20-Poly1305 as the AEAD encryption scheme. As is known from the analysis of SSL/TLS,[1, 4, 5, 14] and more generally [16], such negotiation mechanisms can lead to downgrade attacks that can fatally undermine security especially if a protocol supports both weak and strong cryptographic options. This decision to avoid ciphersuite negotiation simplifies the analysis of WireGuard.

Surprisingly, the full key exchange transcript is not authenticated by either party — the `mac1` and `mac2` values are keyed with public values $H(\mathtt{label}_3 \| g^v)$ and *cookie* and thus can be computed by an adversary. While the hash values H_3, H_4 and H_9 are headers in AEAD ciphertexts, these H values do not contain all of the transcript information — the session identifiers $\mathtt{sid}_i$ and $\mathtt{sid}_r$ are not involved in either the seed or hash chains. This then limits the options for analysing WireGuard, as we cannot hope to show full transcript authentication properties. It would be a straightforward modification to include the session identifiers in the derivation of the session keys and thus bind the session identifiers to the session keys themselves. One could argue that the lack of binding between transcripts and output session keys has facilitated attacks on SSL/TLS, such as the Triple Handshake attack [6], and so a small modification to the inputs of the chaining values C and hash values H would strengthen the security of the protocol.

4 Security Model

We propose a modification to the eCK-PFS security model introduced by Cremers and Feltz [10] that incorporates preshared keys and strengthens the security definitions accordingly. We explain the framework and give an algorithmic

description of the security model in Section 4.1, and describe the corruption abilities of the adversary in Section 4.2. 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 4.3. We then give our modified cleanness definitions in Section 4.4. Given that WireGuard uses a mix of long-term identity keys, ephemeral keys and preshared secrets in its key exchange protocol, it is appropriate to use an extended-Canetti-Krawcyzk model (as introduced in [19]), wherein the adversary is allowed to reveal subsets of these secrets. It is claimed in [11] that WireGuard "achieves the requirements of authenticated key exchange (AKE) security, avoids key-compromise impersonation, avoids replay attacks, provides perfect forward secrecy," [11]. These are all notions captured by our extended eCK-PFS model, so our subsequent security proof will formally establish that WireGuard meets its goals.

4.1 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.

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

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

$\mathsf{KE.EPKeyGen}(\lambda) \xrightarrow{\$} (ek, epk)$ 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 4.2. 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 4.4. We give an algorithmic description of this experiment in Figure 2.

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.
- $m_s \in \{0,1\}^* \cup \{\perp\}$ – the concatenation of messages sent by the session, initialised by $\perp$.
- $m_r \in \{0,1\}^* \cup \{\perp\}$ – the concatenation of messages received by the session, initialised by $\perp$.
- $kid \in \{0,1\}^* \cup \{\perp\}$ – the concatenation of public keyshare information received by the session, initialised by $\perp$.
- $\alpha \in \{\texttt{active}, \texttt{accept}, \texttt{reject}, \perp\}$ – the current status of the session, initialised with $\perp$.
- $k \in \{0,1\}^* \cup \{\perp\}$ – the computed session key, or $\perp$ if no session key has yet been computed.
- $ek \in \{0,1\}^* \times \{0,1\}^* \cup \{\perp\}$ – the ephemeral key pair used by the session during protocol execution, initialised as $\perp$.
- $psk \in \{0,1\}^* \times \{0,1\}^* \cup \{\perp\}$ – the preshared secret and identifier used by the session during protocol execution, initialised as $\perp$.
- $st \in \{0,1\}^*$ – any additional state used by the session during protocol execution.

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

$b \xleftarrow{\$} \{0,1\}$
tested $\leftarrow$ `false`
for $i = 1$ **to** n_P **do**
  $(pk_i, sk_i) \xleftarrow{\$} \mathsf{ASKeyGen}(\lambda)$
  $\mathsf{ASKflag}_i \leftarrow$ `clean`
  $\mathbf{PSK_i}[1], \ldots, \mathbf{PSK_i}[n_P] \leftarrow \perp$
  $\mathbf{PSKflag_i}[1], \ldots, \mathbf{PSKflag_i}[n_P] \leftarrow \perp$
  $\mathbf{EPKflag_i}[1], \ldots, \mathbf{EPKflag_i}[n_S] \leftarrow \perp$
  $\mathbf{RSKflag_i}[1], \ldots, \mathbf{RSKflag_i}[n_S] \leftarrow \perp$
  $ctr_i \leftarrow 0$
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
  return 0
end if

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

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

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

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

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

if $(i = j) \vee (\mathbf{PSKflag_i}[j] \neq \perp)$ **then**
  return $\perp$
end if
$(psk, pskid) \leftarrow \mathsf{KE.PSKeyGen}(\lambda)$
$\mathbf{PSK_i}[j] \leftarrow (psk, pskid)$
$\mathbf{PSK_j}[i] \leftarrow (psk, pskid)$
$\mathbf{PSKflag_i}[j], \mathbf{PSKflag_j}[i] \leftarrow$ `clean`
if $pskid \neq \emptyset$ **then**
  return $pskid$
else
  return $\top$
end if

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

if $\pi_i^s.\alpha \neq$ `accept` **then**
  return $\perp$
else
  $\mathbf{RSKflag_i}[s] \leftarrow$ `corrupt`
  return $\pi_i^s.k$
end if

$\mathsf{CorruptASK}(i)$:

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

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

$\mathbf{EKflag_i}[s] \leftarrow$ `corrupt`
return $\pi_i^s.ek$

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

if $\mathbf{PSK_i}[j] = \perp$ **then**
  return $\perp$
end if
if $\mathbf{PSKflag_i}[j] \neq$ `clean` **then**
  return $\perp$
else
  $\mathbf{PSKflag_i}[j] \leftarrow$ `corrupt`
  $\mathbf{PSKflag_j}[i] \leftarrow$ `corrupt`
  return $\mathbf{PSK_i}[j]$
end if

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

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

Fig. 2: eCK-PFS-PSK experiment for adversary $\mathcal{A}$ against the key-indistinguishability security of protocol KE.

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

- preshared keys $\{\mathbf{PSKflag_i}, \mathbf{PSKflag_2}, \ldots, \mathbf{PSKflag_{n_P}}\}$ where for each element $\mathbf{PSKflag_i}[j] \in \mathbf{PSKflag_i}$, $\mathbf{PSKflag_i}[j] \in \{\texttt{corrupt}, \texttt{clean}, \perp\}\ \forall\ i, j \in [n_P]$ and $\mathbf{PSKflag_i}[j] = \perp$ for $i = j$
- long-term keys $\{\mathsf{SKflag}_1, \ldots, \mathsf{SKflag}_{n_P}\}$, where $\mathsf{SKflag}_i \in \{\texttt{corrupt}, \texttt{clean}, \perp\}\ \forall\ i \in [n_P]$
- ephemeral keys $\{\mathbf{EKflag_1}, \ldots, \mathbf{EKflag_{n_P}}\}$, where $\mathbf{EKflag_i}[s] \in \{\texttt{corrupt}, \texttt{clean}, \perp\}\ \forall\ i \in [n_P]$ and $s \in [n_S]$.

We formalise the advantage of a PPT algorithm $\mathcal{A}$ in winning the eCK-PFS-PSK key indistinguishability experiment in the following way:

Definition 5 (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 PPT algorithm $\mathcal{A}$, we define the advantage of $\mathcal{A}$ in the* eCK-PFS-PSK *key-indistinguishability game to be:*

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

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

4.2 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 preshared keys. This would allow our model to be used in analysing (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 preshared 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, modelling the misuse or corruption of long-term secrets in other sessions, and additionally allowing our model to capture forward-secrecy notions.
- ephemeral private keys, modelling the use of bad randomness generators.
- preshared symmetric keys, modelling the leakage of shared secrets, potentially due to the misuse of the preshared secret by the partner, or the forced later revelation of these keys.
- session keys, modelling the leakage of keys by their use in bad cryptographic algorithms.

The adversary interacts with the challenger via the following adversarial queries. An algorithmic descriptions of how the challenger responds is in Figure 2.

- $\mathsf{Create}(i, j, role) \rightarrow \{(i, s), \perp\}$: allows the adversary to begin new sessions. The challenger $\mathcal{C}$ creates a new session π^s_i with $\pi^s_i.pid \leftarrow j$, $\pi^s_i.\rho \leftarrow role$, $\pi^s_i.\alpha \leftarrow \texttt{active}$, $\pi^s_i.T \leftarrow \perp$, $\pi^s_i.sid \leftarrow \perp$, $\pi^s_i.k \leftarrow \perp$. $\mathcal{C}$ also computes $\mathsf{KE.EKeyGen}(\lambda) \xrightarrow{\$} (ek, epk)$ and sets $\pi^s_i.ek \leftarrow ek$. If a session π^s_i has already been created, $\mathcal{C}$ returns $\perp$. Otherwise, $\mathcal{C}$ returns (i, s) to $\mathcal{A}$.
- $\mathsf{CreatePSK}(i, j) \rightarrow \{pskid, \top, \perp\}$: allows the adversary to direct parties to generate a preshared key for use in future protocol executions. The challenger $\mathcal{C}$ checks that $i \neq j$ and that $\mathbf{PSK_i}[j] = \mathbf{PSK_j}[i] = \perp$. $\mathcal{C}$ then computes $\mathsf{KE.PSKeyGen}(\lambda) \xrightarrow{\$} psk$ and sets $\mathbf{PSK_i}[j] = \mathbf{PSK_j}[i] \leftarrow psk$, and the PSK register $\mathbf{PSKflag_i}[j] = \mathbf{PSKflag_j}[i] \leftarrow \texttt{clean}$. 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 $\mathbf{PSK_i}[j] \neq \perp$ or $\mathbf{PSK_j}[i] \neq \perp$ (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^s_i has been upheld and $\pi^s_i.\alpha = \texttt{accept}$ and if so, returns $\pi^s_i.k$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.

– $\mathsf{CorruptPSK}(i) \rightarrow \{psk, \perp\}$: allows the adversary access to the secret preshared key jointly shared by parties prior to protocol execution. The challenger $\mathcal{C}$ checks that $\mathbf{PSK_i}[j] = \mathbf{PSK_j}[i] \neq \perp$, and that $\mathbf{PSKflag_i}[j] = \mathbf{PSKflag_j}[i] = \texttt{clean}$. If so, $\mathcal{C}$ returns $PSK \leftarrow \mathbf{PSK_i}[j]$ to $\mathcal{A}$ and sets $\mathbf{PSKflag_i}[j] = \mathbf{PSKflag_j}[i] \leftarrow \texttt{corrupt}$. If $\mathbf{PSK_i}[j] = \mathbf{PSK_j}[i] = \perp$ or $\mathbf{PSKflag_i}[j] = \mathbf{PSKflag_j}[i] \neq \texttt{clean}$, (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 $\mathsf{ASKflag}_i \neq \texttt{corrupt}$. If so, $\mathcal{C}$ returns sk_i to $\mathcal{A}$. If $\mathsf{ASKflag}_i = \texttt{corrupt}$ (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 $\mathbf{EPKflag_{i,s}} = \texttt{clean}$. If so, $\mathcal{C}$ returns $\pi_i^s.ek$ to $\mathcal{A}$, and sets $\mathbf{EPKflag_{i,s}} \leftarrow \texttt{corrupt}$. If $\mathbf{EPKflag_{i,s}} = \texttt{corrupt}$, (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 $\pi_i^s.\alpha \neq \texttt{active}$, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ computes $\mathsf{KE}.f(\lambda, m, \pi_i^s) \rightarrow (m', \pi_i^s)$, sets $\pi_i^s \leftarrow \pi_i^{s\prime}$, and returns m' to $\mathcal{A}$.
– $\mathsf{Test}(i,s) \rightarrow \{k, \perp\}$: 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 $\mathcal{A}$.

4.3 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 [10] used the relaxed notion of *origin sessions*. However, both of these are still too restrictive for analysing 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 6 (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 because $\pi_i^s.kid$ is: 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 preshared key identifiers provided by the client and selected by the server.

4.4 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 [10] 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 preshared secret setting.

Definition 7 ($\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$). *A session π_i^s such that $\pi_i^s.\alpha = \texttt{accept}$ in the security experiment defined in Figure 2 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* $\mathbf{PSKflag_i}[\pi_i^s.pid] = \texttt{corrupt}$ *or* $\pi_i^s.psk = \perp$*, the queries* $\mathsf{CorruptASK}(i)$ *and* $\mathsf{CorruptEPK}(i, s)$ *have not* both *been issued.*
4. *If* $\mathbf{PSKflag_i}[\pi_i^s.pid] = \texttt{corrupt}$ *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 preshared secret between the test session and its intended partner has already been revealed. Since preshared keys are optional in our framework, we also must consider the scenario where a preshared 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 preshared 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 [10]. 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 preshared secrets in conditions 3 and 4, where the adversary falls back to eCK-PFS leakage paradigm if the preshared secret between the test session and its peer either does not already exist, or has been already revealed.

5 Security Analysis

In this section we examine the security implications of modelling the WireGuard handshake as a 1-RTT key exchange protocol. We have already noted that this results in a KCI attack on the protocol, also observed in [11]. However, we are able to demonstrate an arguably more serious attack on session key security in our eCK-PFS-PSK security model that results from this modelling. We show the attack in Figure 3 and discuss it in Section 5.1. Making minor modifications to the WireGuard handshake protocol will allow us to prove key-indistinguishability security in the strong eCK-PFS-PSK model. Specifically, we will add a key-confirmation message generated by the initiator. We describe the modified WireGuard handshake protocol in Section 5.2 and prove it secure in Section 5.3.

5.1 Attack on Forward-Secrecy Notions

We give a description of an attack on WireGuard as a 1-RTT protocol that is allowable within the eCK-PFS-PSK security model. It uses the ability of the adversary to target perfect forward secrecy combined with key-compromise-impersonation and results in full session key recovery. Specifically, it allows the adversary to corrupt the long-term key of a responder session, and thus impersonate any party initiating a session to the corrupted party. Since we model WireGuard as a 1-RTT key exchange protocol, we do not include the data transport message that would otherwise authenticate the initiator to a responder session, and thus the responder has to accept the session as soon as the responder has sent the `ResponderHello` message (this being the last message in the 1-RTT version of the protocol). Afterwards, the adversary is permitted to corrupt the long-term key of the party that it is impersonating. This enables it to compute the session key, and thus distinguish real session keys from random ones, breaking eCK-PFS-PSK key indistinguishability. The exact details of this attack within the eCK-PFS-PSK security model can be found in Figure 3.

Readers may argue that this attack is implausible in a real-world setting, and is entirely artificial, allowable only because of the severe key compromises permitted in the security model. We tend to agree, and present the attack here only as a means of illustrating that the WireGuard handshake protocol, as originally presented in its 1-RTT form, is not only vulnerable to standard KCI attacks, but also to key recovery attacks, and therefore not directly amenable to strong security proofs without incorporating additional messages as part of the handshake.

5.2 The Modified WireGuard Handshake

We note that in [11], the protection for a responder against KCI attacks is to wait for authenticated data transport messages to arrive from the initiator. Incorporating this into the WireGuard handshake would make it impossible

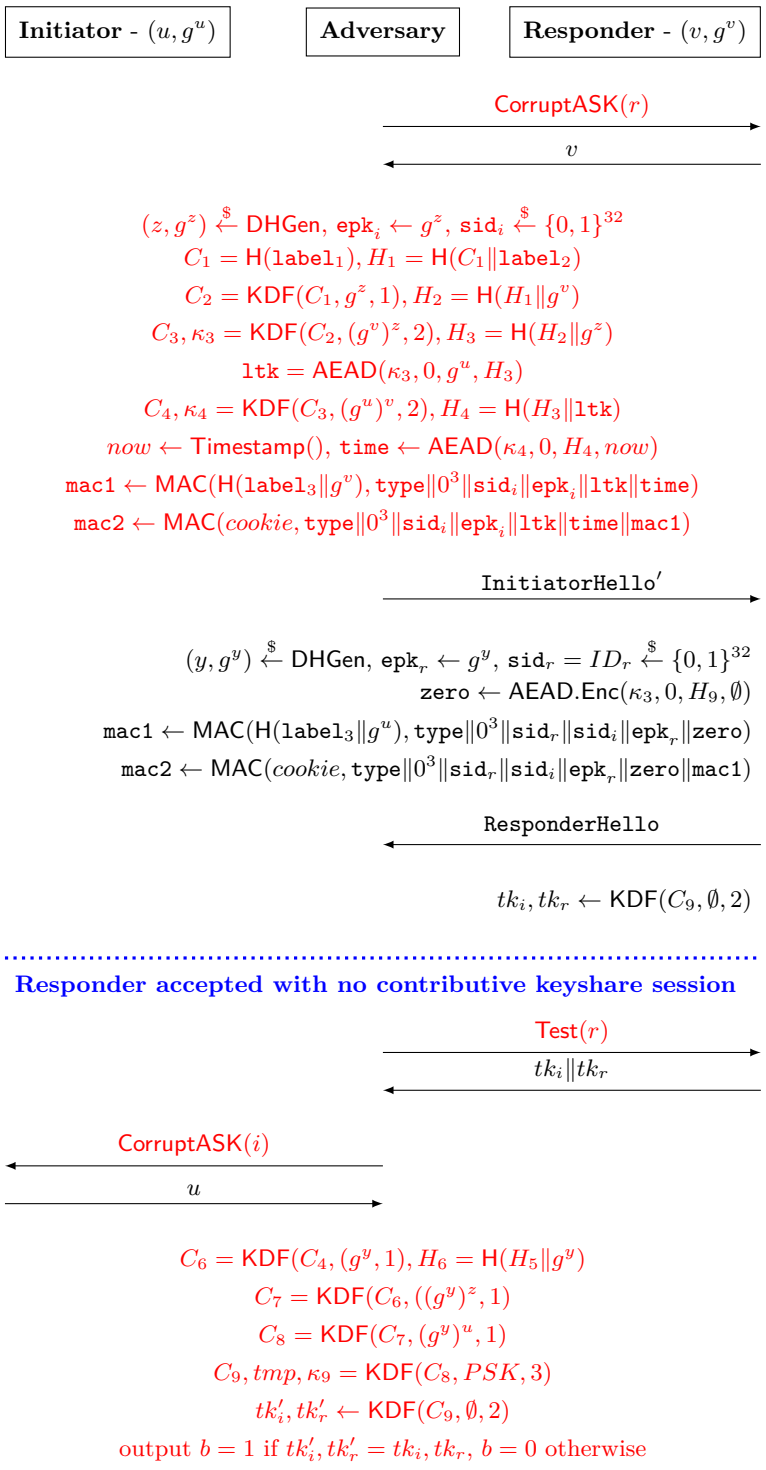

Fig. 3: A description of our attack on WireGuard in the eCK-PFS-PSK security model.

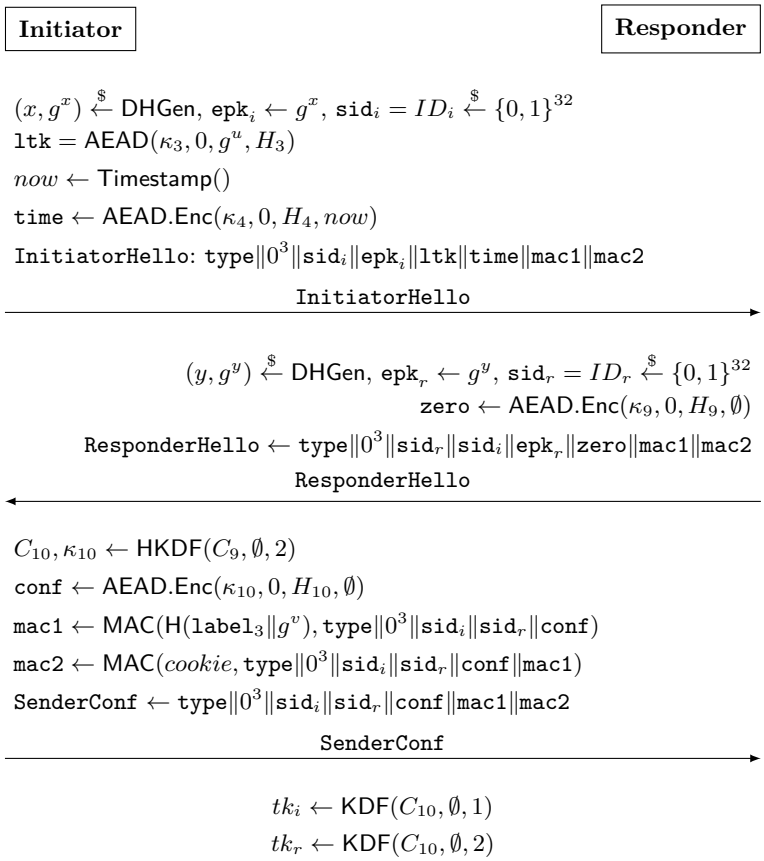

Fig. 4: The modification to the WireGuard handshake that allows eCK-PFS-PSK security. The change is limited to an additional `SenderConf` message that contains the value $\mathtt{conf} \leftarrow \mathsf{AEAD}(\kappa_{10}, 0, \emptyset, H_{10})$. Except for the computation of the new C_{10}, κ_{10} values, all values are computed as in the original WireGuard handshake protocol, and can be found in Table 1.

to prove it secure with respect to a key indistinguishability security notion, however, because the session keys, being used in the data transport protocol, would no longer remain indistinguishable from random when the subject of a Test query.

As explained in the introduction, there are two basic ways of surmounting this obstacle: consider the protocol (handshake and data transport) as a monolithic whole, or modify the protocol. We adopt the latter approach.

We present a modification to the WireGuard handshake protocol that allows us to prove notions of perfect forward secrecy and defence against key-compromise impersonation attacks. Figure 4 shows the modified protocol, denoted mWG. It adds a key-confirmation message sent from the initiator to the responder, computed using an extra derived key κ_{10} used solely for this purpose.

Our modifications are minor (involving at most 5 extra symmetric key operations) and do not require an additional round trip before either party can begin sending transport data, as the responder was already required to wait for initiator-sent data before it was able to begin safely sending its own.

5.3 Security of the Modified WireGuard Handshake

This section is dedicated to proving our main result:

Theorem 1. *The modified WireGuard handshake protocol* mWG *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, for any PPT algorithm* $\mathcal{A}$ *against the* eCK-PFS-PSK *key-indistinguishability game (defined in Figure 2)* $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK}}_{\mathsf{mWG},\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}},n_P,n_S,\mathcal{A}}(\lambda)$ *is negligible under the* prf*,* auth-aead*,* sym-ms-PRFODH*,* sym-mm-PRFODH *and* ddh *assumptions. More precisely:*

$$
\begin{aligned}
\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda) \leq\ & n_P^2 n_S \Big(\mathsf{Adv}^{\mathsf{sym\text{-}ms\text{-}PRFODH}}_{\mathbb{G},q,\mathsf{HKDF},\mathcal{A}}(\lambda) + 4\cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)\Big) \\
&+\ n_P^2 n_S \Big(\mathsf{Adv}^{\mathsf{sym\text{-}ms\text{-}PRFODH}}_{\mathbb{G},q,\mathsf{HKDF},\mathcal{A}}(\lambda) + 2\cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)\Big) \\
&+ \max\Big\{ \Big(n_P^2 n_S^2 \big(3\cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda)\big)\Big), \Big(n_P^2 n_S^2 \Big(\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},q,\mathcal{A}}(\lambda) + 5\cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda)\Big), \\
&\Big(\ n_P^2 n_S^2 \big(\mathsf{Adv}^{\mathsf{sym\text{-}ms\text{-}PRFODH}}_{\mathsf{HKDF},\mathbb{G},q,\mathcal{A}}(\lambda) + 3\cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda)\big)\Big), \\
&\Big(\ n_P^2 n_S^2 \big(\mathsf{Adv}^{\mathsf{sym\text{-}mm\text{-}PRFODH}}_{\mathsf{HKDF},\mathbb{G},q,\mathcal{A}}(\lambda) + 6\cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda)\big)\Big)\Big\}
\end{aligned}
$$

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 $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_l}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda)$ 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 $\pi_i^s.\rho = \texttt{init}$) has no contributive keyshare session.
2. The session π_i^s (where $\pi_i^s.\rho = \texttt{resp}$) has no contributive keyshare session.
3. The session π_i^s has a contributive keyshare session.

It follows then that $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}} \leq \big(\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_1}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_2}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_3}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda)\big)$. 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 negligble.

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.

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.

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 $abort_{\texttt{accept}}$ that will occur if $\pi_i^s \leftarrow \texttt{accept}$. The following games then are designed to bound the probability of $abort_{\texttt{accept}}$ occuring 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 DH challenge values from the sym-ms-PRFODH 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 the sym-ms-PRFODH 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 `ResponderHello` 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 $\pi_i^s.\rho = \texttt{init}$ will only accept if it recieves 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

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

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 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 $\pi_i^s.\alpha \leftarrow \texttt{reject}$, then the $\mathsf{Test}(i, s)$ query will be rejected by the challenger as specified in Figure 2. 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 $\pi_i^s.\alpha = \texttt{reject}$. Thus

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

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

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

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

Game 5 In this game, we replace the computation of the C_3, κ_3 values with uniformly random and independent values $\widetilde{C_3}, \widetilde{\kappa_3}$. We do so by interacting with a sym-ms-PRFODH 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 initialise a sym-ms-PRFODH challenger, and embed the DH challenge keyshare g^u into the long-term public-key of party P_j and give g^u to the adversary with all other (honestly generated) public keys. 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 $\pi_i^s.\alpha \leftarrow \texttt{accept}$, and thus will not need to reveal the private key u of the challenge DH keyshare to $\mathcal{A}$. However, we must account for all sessions t such that π_j^t must use the private key for computations. In WireGuard, the long-term private keys are used to compute the following:

- In sessions where P_j acts as the initiator: $C_4, \kappa_4 \leftarrow \mathsf{HKDF}(C_3, g^{uv})$, $C_8 \leftarrow \mathsf{HKDF}(C_6, g^{uy})$
- In sessions where P_j acts as the responder: $C_3, \kappa_3 \leftarrow \mathsf{HKDF}(C_2, g^{xu})$, $C_4, \kappa_4 \leftarrow \mathsf{HKDF}(C_3, g^{uv})$

Dealing with the challenger's computation of these values will be done in two ways:

- The other Diffie-Hellman private key (be it v, x or y) is a value that has been generated by another honest session. The challenger can then use its own internal knowledge of v, x or y to complete the computations.
- The other Diffie-Hellman private key is a value that is unknown to the challenger, as it has been generated instead by the adversary

In the second case, the challenger must instead use the ODH_u oracle provided by the sym-ms-PRFODH challenger, specifically querying $\mathsf{ODH}_u(C_k, X)$, (where X is the Diffie-Hellman public keyshare such that the private key is unknown to the challenger) which will output $\mathsf{HKDF}(C_k, X^u)$ using the sym-ms-PRFODH challenger's internal knowledge of u. In a similar fashion we embed the other DH challenge value g^v into the ephemeral public-key g^x of the test session π_i^s. Since the key will only be used in this session, the private key v (for consistency with the WireGuard protocol notation introduced in Figure 4, we will refer to the private key as x) will be used in the two following ways:

- $C_3, \kappa_3 \leftarrow \mathsf{HKDF}(C_2, g^{xv})$
- $C_7 \leftarrow \mathsf{HKDF}(C_6, g^{xy})$

Taking the second case first, we compute C_7 by querying the $\mathsf{ODH}_v(C_6, g^y)$ oracle provided by the sym-ms-PRFODH challenger, which will compute $\mathsf{HKDF}(C_6, g^{xy})$ concretely, and can continue without further disruption. In the first case, we query the sym-ms-PRFODH with the challenge salt value C_2, and will return $\widetilde{C_3}, \widetilde{\kappa_3}$. If the test bit sampled by the sym-ms-PRFODH challenger is 0, then $\widetilde{C_3}, \widetilde{\kappa_3} \leftarrow \mathsf{HKDF}(C_2, g^{xv})$ and we are in **Game 4**. If the test bit sampled by the sym-ms-PRFODH challenger is 1, then $\widetilde{C_3}, \widetilde{\kappa_3} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 5**. Thus any adversary $\mathcal{A}$ capable of distinguishing this change can be turned into a successful adversary against the sym-ms-PRFODH assumption, and we find:

$$\Pr(abort_{\texttt{accept}}) \leq \mathsf{Adv}^{\text{sym-ms-PRFODH}}_{\mathbb{G},q,\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_4).$$

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 initialise a prf challenger and query $((g^u)^v)$, 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 5**, $\widetilde{C_3}$ is a uniformly random and independent value, this replacement is sound. If the test bit sampled by the prf challenger is 0, then $\widetilde{C_4}, \widetilde{\kappa_4} \leftarrow \mathsf{HKDF}(C_3, g^{uv})$ and we are in **Game 5**. 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:

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

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 initialise a prf challenger, query it with (g^y), 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 $\widetilde{C_6} \leftarrow \mathsf{prf}(C_4, g^y)$ 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:

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

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 $C_7 \leftarrow \mathsf{HKDF}(\widetilde{C_6}, g^{xy})$ we instead initialise a prf challenger and query it with g^{xy}. 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 $\widetilde{C_7} \leftarrow \mathsf{HKDF}(\widetilde{C_6}, g^{xy})$ 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

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

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 $C_8 \leftarrow \mathsf{HKDF}(\widetilde{C_7}, g^{uy})$ we instead initialise a prf challenger and query it with g^{uy}. 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 $C_8 \leftarrow \mathsf{HKDF}(\widetilde{C_7}, g^{uy})$ 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

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

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 initialise 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

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

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 initalise an aead challenger to decrypt 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 zero ciphertext breaks the security of the AEAD scheme. We find that

$$\Pr(break_9) = \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda).$$

Thus

$$\Pr(abort_{\texttt{accept}}) \leq \big(\mathsf{Adv}^{\mathsf{sym\text{-}ms\text{-}PRFODH}}_{\mathbb{G},q,\mathsf{HKDF},\mathcal{A}}(\lambda) + 4 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)\big)$$

It follows then

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_1}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda) \leq \; n_P^2 n_S \Big(\mathsf{Adv}^{\mathsf{sym\text{-}ms\text{-}PRFODH}}_{\mathbb{G},q,\mathsf{HKDF},\mathcal{A}}(\lambda) + 4 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)\Big).$$

Case 2: Test 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.

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 reject status. Since we alrady 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 $abort_{\texttt{accept}}$ that will occur if $\pi_i^s \leftarrow \texttt{accept}$. The following games then are designed to bound the probability of $abort_{\texttt{accept}}$ occuring 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 DH challenge values from the sym-ms-PRFODH 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 the sym-ms-PRFODH 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 accept state (and trigger the $abort_{\texttt{accept}}$ 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 aead-auth challenger, and add an abort rule $abort_{dec}$ that triggers if the conf ciphertext received in the SenderConf message would decrypt without error. Logically then, since the $\widetilde{\kappa_{10}}$ value is internal to the aead challenger, if 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 zero to break the 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 aead-auth security of AEAD.

Since a session with role $\pi_i^s.\rho = \texttt{resp}$ will only accept if it recieves a ciphertext conf 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 a contributive keyshare partner π_j^t.

Game 0 This is a standard eCK-PFS-PSK game. Thus we have:

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

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 2. 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 $abort_{\texttt{accept}}$ that triggers if the status of the test session $\pi_i^s \leftarrow \texttt{accept}$. It is clear then that

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

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. $\Pr(abort_{\texttt{accept}})$) is negligibly close to zero.

Game 5 In this game, we replace the computation of the C_8 value with a uniformly random and independent value $\widetilde{C_8}$. We do so by interacting with a $\mathsf{sym\text{-}ms\text{-}PRFODH}$ 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 intialise a $\mathsf{sym\text{-}ms\text{-}PRFODH}$ challenger, and embed the DH challenge keyshare g^u into the long-term public-key of party P_j and give g^u to the adversary with all other (honestly generated) public keys. 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 $\pi_i^s.\alpha \leftarrow \texttt{accept}$, and thus will not need to reveal the private key u of the challenge DH keyshare to $\mathcal{A}$. However, we must account for all sessions t such that π_j^t must use the private key for copmutations. In WireGuard, the long-term private keys are used in the following computations:

- In sessions where P_j acts as the initiator: $C_4, \kappa_4 \leftarrow \mathsf{HKDF}(C_3, g^{uv})$, $C_8 \leftarrow \mathsf{HKDF}(C_6, g^{uy})$
- In sessions where P_j acts as the responder: $C_3, \kappa_3 \leftarrow \mathsf{HKDF}(C_2, g^{xu})$, $C_4, \kappa_4 \leftarrow \mathsf{HKDF}(C_3, g^{uv})$

Dealing with the challenger's computation of these values will be done in two ways:

- The other Diffie-Hellman private key (be it v, x or y) is a value that has been generated by another honest session. The challenger can then use its own internal knowledge of v, x or y to complete the computations.
- The other Diffie-Hellman private key is a value that is unknown to the challenger, as it has been generated instead by the adversary

In these cases, the challenger must instead use the ODH_u oracle provided by the $\mathsf{sym\text{-}ms\text{-}PRFODH}$ challenger, specifically querying $\mathsf{ODH}_u(C_k, X)$, (where X is the Diffie-Hellman public keyshare such that the private key is unknown to the challenger) which will output $\mathsf{HKDF}(C_k, X^u)$ using the $\mathsf{sym\text{-}ms\text{-}PRFODH}$ challenger's internal knowledge of u.

In a similar fashion we embed the other DH challenge value g^v into the ephemeral public-key g^y of the test session π_i^s. Since the key will only be used in this session, the private key v (for consistency with the WireGuard protocol notation introduced in Figure 4, we will refer to the private key as y) will be used in the two following ways:

– $C_7 \leftarrow \mathsf{HKDF}(C_6, g^{xy})$
– $C_8 \leftarrow \mathsf{HKDF}(C_7, g^{uy})$

Taking the first case first, we compute C_7 by querying the $\mathsf{ODH}_v(C_6, g^x)$ oracle provided by the sym-ms-PRFODH challenger, which will compute $\mathsf{HKDF}(C_6, g^{xy})$ concretely, and can continue without further disruption. In the second case, we query the sym-ms-PRFODH with the challenge salt value C_7, and will return $\widetilde{C_8}$. If the test bit sampled by the sym-ms-PRFODH challenger is 0, then $\widetilde{C_8} \leftarrow \mathsf{HKDF}(C_7, g^{uy})$ and we are in **Game 4**. If the test bit sampled by the sym-ms-PRFODH challenger is 1, then $\widetilde{C_8} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 5**. Thus any adversary $\mathcal{A}$ capable of distinguishing this change can be turned into a successful adversary against the sym-ms-PRFODH assumption, and we find:

$$\Pr(abort_{\mathtt{accept}}) \leq \mathsf{Adv}^{\mathsf{sym\text{-}ms\text{-}PRFODH}}_{\mathbb{G},q,\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_5)$$

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 initialise 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 5**, $\widetilde{C_8}$ 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 5**. 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:

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

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 initialise 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 $abort_{\mathtt{decrypt}}$ that triggers if the test session π_i^s receives a ciphertext `conf` in the `SenderConf` message that decrypts correctly. Since the test session π_i^s will only reach an accept status if `conf` decrypts correctly, it follows that

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

Now we show that the probability of $abort_{\mathtt{decrypt}}$ is negligibly close to zero. We do so by initialising an aead-auth challenger to decrypt `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 `conf` in the `SenderConf` message that decrypts correctly and the aead encryption oracle has not been queried, then it follows that this ciphertext `conf` is a forged ciphertext, breaking the auth security of the AEAD scheme. Thus, we find that:

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

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

$$\Pr(abort_{\mathtt{accept}}) \leq \left(\mathsf{Adv}^{\mathsf{sym\text{-}ms\text{-}PRFODH}}_{\mathbb{G},q,\mathsf{HKDF},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda)\right).$$

We can finally show that

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_2}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda) \leq n_P^2 n_S \Big(\mathsf{Adv}^{\mathsf{sym\text{-}ms\text{-}PRFODH}}_{\mathbb{G},q,\mathsf{HKDF},\mathcal{A}}(\lambda) + 2 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{auth\text{-}aead}}_{\mathsf{AEAD},\mathcal{A}}(\lambda) \Big).$$

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[3]: $\mathcal{A}$ 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 preshared 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 $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.1}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda)$ the advantage of $\mathcal{A}$ in winning in this case and refer to this as the *preshared subcase*.
2. $\mathcal{A}$ did not issue $\mathsf{CorruptEPK}(i,s)$ or $\mathsf{CorruptEPK}(j,t)$. We denote with $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.2}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda)$ 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 $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.3}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda)$ 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 $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.4}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda)$ 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 $\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.5}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda)$ the advantage of $\mathcal{A}$ and refer to this as the *long-terms subcase*.

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

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_3}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda) = \max\Big\{ \mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.1}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda), \mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.2}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda), \mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.3}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda), \mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.4}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda), \mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.5}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda) \Big\}$$

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 preshared 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.

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 $\mathsf{eCK\text{-}PFS\text{-}PSK}$-security of the modified WireGuard protocol in the preshared key subcase is negligible.

Game 0 This is a standard $\mathsf{eCK\text{-}PFS\text{-}PSK}$ with cleanness predicate $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ upheld as in Definition 7. Thus

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

[3] 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

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 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

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

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 prf challenger in the following way: When it is time to compute $C_9, tmp, \kappa_9 \leftarrow \mathsf{HKDF}(C_8, psk)$ we instead initialise a prf 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 prf 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 prf security of HKDF and thus

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

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 initialise 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, 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 initialise 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:

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.1}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda) \leq n_P^2 n_S^2 \Big(3 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda)\Big).$$

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

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

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 value g^{xy} computed in the test session π_i^s and its honest contributive keyshare session partner with a random element from the same group. Note that since the initiator session and the responder session both get key confirmation messages that include derivations based on the Diffie-Hellman keyshares, and thus know that the ephemeral keyshare values were recieved by the other session without modification. We explicitly interact with a ddh challenger, and replace the ephemeral $\mathtt{epk}_i$ and $\mathtt{epk}_r$ values sent in the `InitiatorHello` and `ResponderHello` messages with the challenge DH keyshares from the ddh challenger. We require the private keys of the DH values in three computations in the initiator and responder sessions:

- $C_3, \kappa_3 \leftarrow \mathsf{HKDF}(C_2, g^{vx})$
- $C_7 \leftarrow \mathsf{HKDF}(C_6, g^{xy})$
- $C_8 \leftarrow \mathsf{HKDF}(C_7, g^{uy})$

In the first and third cases, the challenger uses its internal knowledge of the long-term keys of the parties to simulate knowledge of the ephemeral private keys, computing instead $C_3, \kappa_3 \leftarrow \mathsf{HKDF}(C_2, (g^x)^v)$ and $C_8 \leftarrow \mathsf{HKDF}(C_7, (g^y)^u)$. In the second case, we replace the g^{xy} value with the output g^z value from the ddh challenger. When the test bit sampled by the ddh challenger is 0, then $z = xy$ and we are in **Game 2**. When the test bit sampled by the ddh challenger is 1, then $z \xleftarrow{\$} \mathbb{Z}_q$ and we are in **Game 3**. Any adversary that can detect this change can be turned into an adversary against the ddh problem and thus

$$\Pr(break_2) \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},q,\mathcal{A}}(\lambda) + \Pr(break_3).$$

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 HKDF challenger in the following way: When it is time to compute $C_7 \leftarrow \mathsf{HKDF}(C_6, g^z)$ we instead initialise a HKDF challenger and query it with C_6. We note that by **Game 3** that g^z 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 $\widetilde{C_7} \leftarrow \mathsf{HKDF}(C_6, g^z)$ 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

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

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 $C_8 \leftarrow \mathsf{HKDF}(\widetilde{C_7}, g^{uy})$ we instead initialise a prf challenger and query it with g^{uy}. 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 $C_8 \leftarrow \mathsf{PRF}(\widetilde{C_7}, g^{uy})$ 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 initialise 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 initialise 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 initialise 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:

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.2}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda) \leq \; n_P^2 n_S^2 \Big(\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},q,\mathcal{A}}(\lambda) + 5 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \Big)$$

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:

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

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 the C_3, κ_3 values with uniformly random and independent values $\widetilde{C_3}, \widetilde{\kappa_3}$. We do so by interacting with a sym-ms-PRFODH 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 intialise a sym-ms-PRFODH challenger, and embed the DH challenge keyshare g^v into the long-term public-key of party P_j and give g^v to the adversary with all other (honestly generated) public keys. Note that by the definition of this case, $\mathcal{A}$ is not able to issue a $\mathsf{CorruptASK}(j)$ query, and thus will not need to reveal the private key v of the challenge DH keyshare to $\mathcal{A}$. However, we must account for all sessions t such that P_j must use the private key for computations. In WireGuard, the long-term private keys are always used in the following ways (note that for consistency we use v to indicate the long-term private key of P_j independently of its role):

- If the party P_j is acting as the responder:
  - $C_3, \kappa_3 \leftarrow \mathsf{HKDF}(C_2, g^{x'v})$
  - $C_4, \kappa_4 \leftarrow \mathsf{HKDF}(C_3, g^{u'v})$
- If the party P_j is acting as the initiator:
  - $C_4, \kappa_4 \leftarrow \mathsf{HKDF}(C_3, g^{u'v})$
  - $C_8, \kappa_8 \leftarrow \mathsf{HKDF}(C_7, g^{y'v})$

Dealing with the challenger's computation of these values will be done in two ways:

- x', u', or y' are values that has been generated by another honest session. The challenger can then use its own internal knowledge of x', u', and y' to compute C_3, κ_3, C_4, κ_4 and C_8, κ_8 respectively.
- x', u', or y' are values that are unknown to the challenger, as they have been generated instead by the adversary.

In these cases, the challenger must instead use the ODH_v oracle provided by the sym-ms-PRFODH challenger, specifically querying $\mathsf{ODH}_v(C_2, g^{x'})$, $\mathsf{ODH}_v(C_3, g^{u'})$, or $\mathsf{ODH}_v(C_7, g^{y'})$ which will output $\mathsf{ODH}_v(C_2, g^{x'v})$, $\mathsf{ODH}_v(C_3, g^{u'v})$, or $\mathsf{ODH}_v(C_7, g^{y'v})$ using the sym-ms-PRFODH challenger's internal knowledge of v.

In a similar fashion we embed the other DH challenge value g^x into the ephemeral public-key g^x of the test session π_i^s. Since the key will only be used in this session, the private key x will be used in the two following ways:

– $C_3, \kappa_3 \leftarrow \mathsf{HKDF}(C_2, g^{xv})$
– $C_7 \leftarrow \mathsf{HKDF}(C_6, g^{xy})$

Taking the second case first, we compute C_7 by querying the $\mathsf{ODH}_x(C_6, g^y)$ oracle provided by the sym-ms-PRFODH challenger, which will compute $\mathsf{HKDF}(C_6, g^{xy})$ concretely, and can continue without further disruption. In the first case, we query the sym-ms-PRFODH with the challenge salt value C_2, and will return $\widetilde{C_3}, \widetilde{\kappa_3}$. If the test bit sampled by the sym-ms-PRFODH challenger is 0, then $\widetilde{C_3}, \widetilde{\kappa_3} \leftarrow \mathsf{HKDF}(C_2, g^{xv})$ and we are in **Game 2**. If the test bit sampled by the sym-ms-PRFODH challenger is 1, then $\widetilde{C_3}, \widetilde{\kappa_3} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 3**. Thus any adversary $\mathcal{A}$ capable of distinguishing this change can be turned into a successful adversary against the sym-ms-PRFODH assumption, and we find:

$$\Pr(break_2) \leq \mathsf{Adv}^{\mathsf{sym\text{-}ms\text{-}PRFODH}}_{\mathbb{G},q,\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_3)$$

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 $C_4, \kappa_4 \leftarrow \mathsf{HKDF}(\widetilde{C_3}, g^{uv})$ we instead initialise a prf challenger and query it with g^{uv}. We note that by **Game 3** 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 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:

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

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 $C_6 \leftarrow \mathsf{HKDF}(\widetilde{C_4}, g^y)$ we instead initialise a prf challenger and query it with g^y. 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 $C_7 \leftarrow \mathsf{HKDF}(\widetilde{C_6}, g^{xy})$ we instead initialise a prf challenger and query it with g^{xy}. 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 $\widetilde{C_7} \leftarrow \mathsf{HKDF}(\widetilde{C_6}, g^{xy})$ 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 $C_8 \leftarrow \mathsf{HKDF}(\widetilde{C_7}, g^{uy})$ we instead initialise a prf challenger and query it with g^{uy}. 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 $C_8 \leftarrow \mathsf{HKDF}(\widetilde{C_7}, g^{uy})$ 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 initialise 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 initialise 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 initialise 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:

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

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.3}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda) \leq \; n_P^2 n_S^2 \Big(\mathsf{Adv}^{\mathsf{sym\text{-}ms\text{-}PRFODH}}_{\mathsf{HKDF},\mathbb{G},q,\mathcal{A}}(\lambda) + 7 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \Big)$$

Case 3.4: The Long-term/Ephemeral Subcase In this subcase we know that (by the definition of the cleanness predicate $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ 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 $\mathsf{CorruptEPK}(j, t)$ 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 $\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$ upheld. Thus we have:

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

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 the C_8 values with a uniformly random and independent value $\widetilde{C_8}$. We do so by interacting with a sym-ms-PRFODH challenger in the following way:

Note that by **Game 2**, we know at the beginning of the experiment the index of session π_j^t such that $\mathsf{Test}(i,s)$ is issued by the adversary and π_j^t is the honest partner of π_i^s. Similarly, by **Game 1**, we know at the beginning of the experiment the index of the party P_i of the test session π_i^s. Thus, we intialise a sym-ms-PRFODH challenger, and embed the DH challenge keyshare g^u into the long-term public-key of party P_i and give g^u to the adversary with all other (honestly generated) public keys. Note that by the definition of this case, $\mathcal{A}$ is not able to issue a $\mathsf{CorruptASK}(i)$ query, and thus will not need to reveal the private key u of the challenge DH keyshare to $\mathcal{A}$. However, we must account for all sessions such that P_i must use the private key for computations. In WireGuard, the long-term private keys are always used in the following ways (note that for consistency we use u to indicate the long-term private key of P_i independently of its role):

- If the party P_i is acting as the initiator:
  - $C_4, \kappa_4 \leftarrow \mathsf{HKDF}(C_3, g^{uv'})$
  - $C_8, \kappa_8 \leftarrow \mathsf{HKDF}(C_7, g^{uy'})$
- If the party P_i is acting as the responder:
  - $C_3, \kappa_3 \leftarrow \mathsf{HKDF}(C_2, g^{x'u})$
  - $C_4, \kappa_4 \leftarrow \mathsf{HKDF}(C_3, g^{uv'})$

Dealing with the challenger's computation of these values will be done in two ways:

- x', v', or y' are values that have been generated by another honest session. The challenger can then use its own internal knowledge of x', v', and y' to compute C_3, κ_3, C_4, κ_4 and C_8, κ_8 respectively.
- x', v', or y' are values that are unknown to the challenger, as they have been generated instead by the adversary.

In these cases, the challenger must instead use the ODH_v oracle provided by the sym-ms-PRFODH challenger, specifically querying $\mathsf{ODH}_u(C_2, g^{x'})$, $\mathsf{ODH}_u(C_3, g^{v'})$, or $\mathsf{ODH}_u(C_7, g^{y'})$ which will output $\mathsf{ODH}_u(C_2, g^{x'u})$, $\mathsf{ODH}_u(C_3, g^{uv'})$, or $\mathsf{ODH}_u(C_7, g^{uy'})$ using the sym-ms-PRFODH challenger's internal knowledge of u.

In a similar fashion we embed the other DH challenge value g^y into the ephemeral public-key of the partner session π_j^t. The private key y will be used in the two following ways:

- $C_8 \leftarrow \mathsf{HKDF}(C_7, g^{uy})$
- $C_7 \leftarrow \mathsf{HKDF}(C_6, g^{xy})$

Taking the second case first, we compute C_7 by querying the $\mathsf{ODH}_y(C_6, g^x)$ oracle provided by the sym-ms-PRFODH challenger, which will compute $\mathsf{HKDF}(C_6, g^{xy})$ concretely, and can continue without further disruption. In the first case, we query the sym-ms-PRFODH with the challenge salt value C_7, and will return $\widetilde{C_8}$. If the test bit sampled by the sym-ms-PRFODH challenger is 0, then $\widetilde{C_8} \leftarrow \mathsf{HKDF}(C_7, g^{uy})$ and we are in **Game 2**. If the test bit sampled by the sym-ms-PRFODH challenger is 1, then $\widetilde{C_8}, \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 3**. Thus any adversary $\mathcal{A}$ capable of distinguishing this change can be turned into a successful adversary against the sym-ms-PRFODH assumption, and we find:

$$\Pr(break_2) \leq \mathsf{Adv}^{\mathsf{sym\text{-}ms\text{-}PRFODH}}_{\mathbb{G},q,\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_3)$$

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 initialise a prf challenger and query it with psk. We note that by **Game 3** 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 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:

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

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 initialise 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 initialise 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:

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

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.4}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda) \leq \; n_P^2 n_S^2 \Big(\mathsf{Adv}^{\mathsf{sym\text{-}ms\text{-}PRFODH}}_{\mathsf{HKDF},\mathbb{G},q,\mathcal{A}}(\lambda) + 3 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \Big)$$

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:

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

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 the C_4, κ_4 values with uniformly random and independent values $\widetilde{C_4}, \widetilde{\kappa_4}$. We do so by interacting with a sym-mm-PRFODH challenger in the following way:

Note that by **Game 2**, we know at the beginning of the experiment the index of the party owner P_i 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 intialise a sym-mm-PRFODH challenger, and embed the DH challenge keyshares $g\tilde{A}\dot{z}$, g^v into the long-term public-keys of party P_i and P_j and give g^u, g^v to the adversary with all other (honestly generated) public keys. Note that by the definition of this case, $\mathcal{A}$ is not able to issue either a $\mathsf{CorruptASK}(i)$ or a $\mathsf{CorruptASK}(j)$ query, and thus will not need to reveal the private keys u, v of the challenge DH keyshare to $\mathcal{A}$. However, we must account for all sessions such that P_i and P_j must use the private key for computations. In WireGuard, the long-term private keys are always used in the following ways (note that for consistency we use u to indicate the long-term private key of party P independently of its role and identity):

- If the party P is acting as the initiator:
  - $C_4, \kappa_4 \leftarrow \mathsf{HKDF}(C_3, g^{uv'})$
  - $C_8, \kappa_8 \leftarrow \mathsf{HKDF}(C_7, g^{uy'})$
- If the party P is acting as the responder:
  - $C_3, \kappa_3 \leftarrow \mathsf{HKDF}(C_2, g^{x'u})$
  - $C_4, \kappa_4 \leftarrow \mathsf{HKDF}(C_3, g^{uv'})$

Dealing with the challenger's computation of these values will be done in two ways:

- x', v', or y' are values that have been generated by another honest session. The challenger can then use its own internal knowledge of x', v', and y' to compute C_3, κ_3, C_4, κ_4 and C_8, κ_8 respectively.
- x', v', or y' are values that are unknown to the challenger, as they have been generated instead by the adversary.

In these cases, the challenger must instead use the ODH_u (respectively ODH_v) oracles provided by the sym-mm-PRFODH challenger, specifically querying $\mathsf{ODH}_u(C_2, g^{x'})$, $\mathsf{ODH}_u(C_3, g^{v'})$, or $\mathsf{ODH}_u(C_7, g^{y'})$ which will output $\mathsf{ODH}_u(C_2, g^{x'u})$, $\mathsf{ODH}_u(C_3, g^{uv'})$, or $\mathsf{ODH}_u(C_7, g^{uy'})$ using the sym-mm-PRFODH challenger's internal knowledge of u (and similarly for the use of the long-term private key v of party P_j).

Now we must deal with the cases where P_i and P_j interact, which will come in two following ways:

- $C_4, \kappa_4 \leftarrow \mathsf{HKDF}(C_3', g^{uv})$, where $C_3' = C_3$ and C_3 is computed in the test session and (potentially) its honest partner session.
- $C_4, \kappa_4 \leftarrow \mathsf{HKDF}(C_3', g^{uv})$, where $C_3' \neq C_3$.

Dealing (unusually) with the first case first, the first time that the challenger must compute this value we query the sym-ms-PRFODH with the challenge salt value C_3, and will return $\widetilde{C_4}, \widetilde{\kappa_4}$. Any following time the challenger must compute the values C_4, κ_4 we simple replace the computation of C_4, κ_4 with the challenge values $\widetilde{C_4}, \widetilde{\kappa_4}$, ensuring consistency of computations through the experiment execution. Taking the second case second, we compute C_4, κ_4 by querying the $\mathsf{ODH}_u(C_3', g^v)$ oracle provided by the sym-mm-PRFODH challenger, which will compute $\mathsf{HKDF}(C_3', g^{uv})$ as the salt value C_3' is distinct from the challenged salt value C_3, and can continue without further disruption. Note that in the test session (and its honest partner session, if one exists) that if the test bit sampled by the sym-mm-PRFODH challenger is 0, then $\widetilde{C_4}, \widetilde{\kappa_4} \leftarrow \mathsf{HKDF}(C_3, g^{uv})$ and we are in **Game 2**. If the test bit sampled by the sym-mm-PRFODH challenger is 1, then $\widetilde{C_4}, \widetilde{\kappa_4} \xleftarrow{\$} \{0,1\}^{|\mathsf{HKDF}|}$ and we are in **Game 3**. Thus any adversary $\mathcal{A}$ capable of distinguishing this change can be turned into a successful adversary against the sym-mm-PRFODH assumption, and we find:

$$\Pr(break_2) \leq \mathsf{Adv}^{\text{sym-mm-PRFODH}}_{\mathbb{G},q,\mathsf{HKDF},\mathcal{A}}(\lambda) + \Pr(break_3)$$

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 $C_6 \leftarrow \mathsf{HKDF}(\widetilde{C_4}, g^y)$ we instead initialise a prf challenger and query it with g^y. We note that by **Game 3** 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:

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

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 $C_7 \leftarrow \mathsf{HKDF}(\widetilde{C_6}, g^{xy})$ we instead initialise a prf challenger and query it with g^{xy}. 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 $\widetilde{C_7} \leftarrow \mathsf{HKDF}(\widetilde{C_6}, g^{xy})$ 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 $C_8 \leftarrow \mathsf{HKDF}(\widetilde{C_7}, g^{uy})$ we instead initialise a prf challenger and query it with g^{uy}. 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 $C_8 \leftarrow \mathsf{HKDF}(\widetilde{C_7}, g^{uy})$ 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 initialise 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 initialise 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 initialise 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:

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

$$\mathsf{Adv}^{\mathsf{eCK\text{-}PFS\text{-}PSK},C_{3.5}}_{\mathsf{mWG},n_P,n_S,\mathcal{A}}(\lambda) \leq \; n_P^2 n_S^2 \Big(\mathsf{Adv}^{\mathsf{sym\text{-}mm\text{-}PRFODH}}_{\mathsf{HKDF},\mathbb{G},q,\mathcal{A}}(\lambda) + 6 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{HKDF},\mathcal{A}}(\lambda) \Big)$$

6 Conclusions and Future Work

We gave a description of the WireGuard protocol, and demonstrated that it has an implicit entanglement of its data transport phase and its key exchange (or handshake) phase. This is needed to ensure protection against KCI attacks. In turn this means that WireGuard either cannot be proven secure as a key exchange protocol using standard key-indistinguishability notions, or it is vulnerable to key-recovery attacks in the KCI setting. Despite this issue, we believe that the design of WireGuard protocol is an interesting one, and our attack is intended more to make a subtle point about the need to cleanly separate a key exchange protocol and the usage of its session keys in subsequent protocols.

We presented the eCK-PFS-PSK security model. This amends the previous eCK-PFS model of [10] to cover key exchange protocols such as WireGuard that combine preshared keys with long-term and ephemeral keys. We then made a minimal set of modifications to the WireGuard handshake protocol, and proved that the modified WireGuard protocol achieves key-indistinguishability security in our new (and strong) eCK-PFS-PSK model.

Other approaches to analysing WireGuard may also be rewarding. Instead of separately establishing the security of the handshake and assuming it securely composes with the data transport phase, one could imagine making a monolithic analysis similar to the ACCE approach introduced in [15]. However, this would require a different "record layer" modelling from that used for TLS in [15] to allow for packet loss and packet reordering. One could also implement our modification and measure its effect on the performance of WireGuard. (We expect it to be very small.)

Finally, we made certain simplifications to simplify our analysis of WireGuard. For instance we did not model the Cookie Reply messages that are designed to protect peers that are under load, nor did we analyse WireGuard's key rotation mechanisms.Given its several attractive properties, WireGuard is certainly deserving of further formal security analysis.

Acknowledgements

Dowling was supported by EPSRC grant EP/L018543/1. Paterson was supported in part by a research programme funded by Huawei Technologies and delivered through the Institute for Cyber Security Innovation at Royal Holloway, University of London, and in part by EPSRC grants EP/M013472/1 and EP/L018543/1.

Bibliography

[1] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Z. Béguelin, and P. Zimmermann. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. In *22nd ACM SIGSAC Conference on Computer and Communications Security, CCS 2015 Denver, Colorado, USA*, pages 5–17, 2015.

[2] J. Aumasson, W. Meier, R. C. Phan, and L. Henzen. *The Hash Function BLAKE*. Information Security and Cryptography. Springer, 2014.

[3] D. J. Bernstein. Curve25519: New Diffie-Hellman speed records. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, Apr. 2006.

[4] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *2015 IEEE Symposium on Security and Privacy*, pages 535–552. IEEE Computer Society Press, May 2015.

[5] K. Bhargavan, C. Brzuska, C. Fournet, M. Green, M. Kohlweiss, and S. Z. Béguelin. Downgrade resilience in key-exchange protocols. In *2016 IEEE Symposium on Security and Privacy*, pages 506–525. IEEE Computer Society Press, May 2016.

[6] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti, and P.-Y. Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *2014 IEEE Symposium on Security and Privacy*, pages 98–113. IEEE Computer Society Press, May 2014.

[7] J. Brendel and M. Fischlin. Zero Round-Trip Time for the Extended Access Control Protocol. In S. N. Foley, D. Gollmann, and E. Snekkenes, editors, *Computer Security – ESORICS 2017*, pages 297–314, Cham, 2017. Springer International Publishing.

[8] J. Brendel, M. Fischlin, F. Günther, and C. Janson. PRF-ODH: Relations, Instantiations, and Impossibility Results. In J. Katz and H. Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 651–681, Cham, 2017. Springer International Publishing.

[9] C. Brzuska, M. Fischlin, B. Warinschi, and S. C. Williams. Composability of Bellare-Rogaway key exchange protocols. In Y. Chen, G. Danezis, and V. Shmatikov, editors, *ACM CCS 11*, pages 51–62. ACM Press, Oct. 2011.

[10] C. J. F. Cremers and M. Feltz. Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In S. Foresti, M. Yung, and F. Martinelli, editors, *ESORICS 2012*, volume 7459 of *LNCS*, pages 734–751. Springer, Heidelberg, Sept. 2012.

[11] J. Donenfeld. WireGuard: Next generation kernel network tunnel. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA*, 2017.

[12] B. Dowling, M. Fischlin, F. Günther, and D. Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In I. Ray, N. Li, and C. Kruegel:, editors, *ACM CCS 15*, pages 1197–1210. ACM Press, Oct. 2015.

[13] B. Dowling, M. Fischlin, F. Günther, and D. Stebila. A cryptographic analysis of the TLS 1.3 draft-10 full and pre-shared key handshake protocol. Cryptology ePrint Archive, Report 2016/081, 2016. `http://eprint.iacr.org/2016/081`.

[14] B. Dowling and D. Stebila. Modelling ciphersuite and version negotiation in the TLS protocol. In E. Foo and D. Stebila, editors, *ACISP 15*, volume 9144 of *LNCS*, pages 270–288. Springer, Heidelberg, June / July 2015.

[15] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 273–293. Springer, Heidelberg, Aug. 2012.

[16] T. Jager, K. G. Paterson, and J. Somorovsky. One bad apple: Backwards compatibility attacks on state-of-the-art cryptography. In *NDSS 2013*. The Internet Society, Feb. 2013.

[17] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566. Springer, Heidelberg, Aug. 2005.

[18] H. Krawczyk, K. G. Paterson, and H. Wee. On the security of the TLS protocol: A systematic analysis. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 429–448. Springer, Heidelberg, Aug. 2013.

[19] B. A. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In W. Susilo, J. K. Liu, and Y. Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 1–16. Springer, Heidelberg, Nov. 2007.

[20] P. Morrissey, N. P. Smart, and B. Warinschi. A modular security analysis of the TLS handshake protocol. In J. Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 55–73. Springer, Heidelberg, Dec. 2008.

[21] Y. Nir and A. Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 7539 (Informational), May 2015.

[22] T. Perrin. The Noise Protocol Framework. October 2017. `http://noiseprotocol.org/noise.html`.

[23] P. Rogaway. Authenticated-encryption with associated-data. In V. Atluri, editor, *ACM CCS 02*, pages 98–107. ACM Press, Nov. 2002.