A Cryptographic Analysis of the WireGuard Protocol

Benjamin Dowling and Kenneth G. Paterson

Information Security Group, Royal Holloway, University of London
{benjamin.dowling, 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.

Keywords: Authenticated key exchange, Cryptographic protocols, Formal analysis, WireGuard.

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 [23]. 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 [22] 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

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

the protocol in the same KCI setting, similar to observations made by Krawczyk in [18]. Such an attack recovers session keys rather than breaking authentication properties, and is arguably more serious. However, the attack requires a 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 [21, 16, 19].

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 [16]) 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, in-line 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 [16, 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 the security assumptions for pseudo-random function (PRF) security, for Authenticated-Encryption with Associated Data (AEAD) schemes (due to space constraints, these can be found in the full version [14]). 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`).

We now introduce the PRFODH assumption that will be needed for our analysis of WireGuard. The first version of this assumption was introduced by [16] 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 [19], 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 in the full version [14]

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 1 (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,* $\mathsf{sym\text{-}lr\text{-}PRFODH}$ *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 "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 $\mathfrak{I}^{\mathsf{sym\text{-}lr\text{-}PRFODH}}_{\mathsf{PRF},\mathcal{A}}$ 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 $\mathcal{A}$.*
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:*
   - ODH_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 $y \leftarrow \mathsf{PRF}_\lambda(S^u, x)$.*
   - ODH_v*: on a query of the form (T, x), the challenger first checks if $T \notin \mathbb{G}$ or if $(T, x) = (g^u, x^*)$ 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 $\mathsf{sym\text{-}lr\text{-}PRFODH}$ *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 $\mathsf{sym\text{-}lr\text{-}PRFODH}$ *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:

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

- 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:

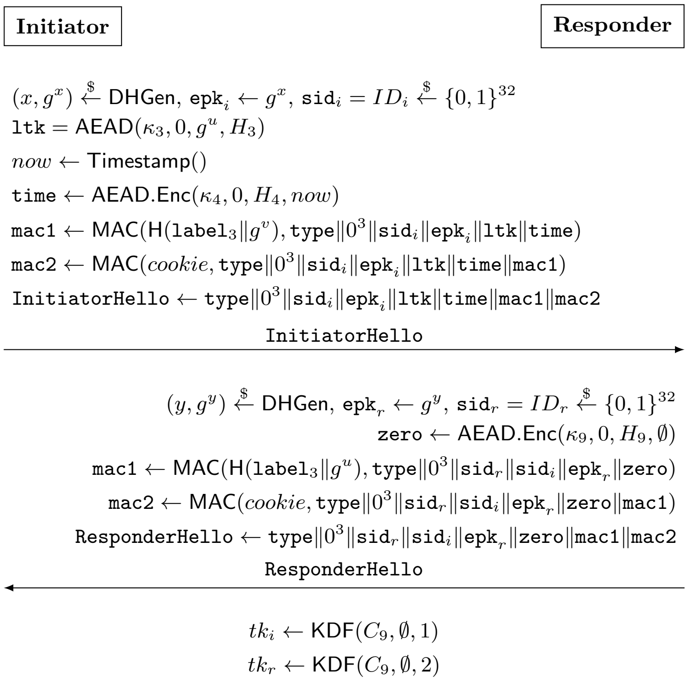
- A randomly-sampled session identifier ID_ρ for each user in the session (i.e we use ID_i to refer to the session identifier of the initiator and for the responder we refer to ID_r).
- An updating seed value C_k, is used to seed the key-derivation function at various points during the key-exchange.
- 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.

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



Initiator **Responder**

$(x, g^x) \overset{\$}{\leftarrow} \mathsf{DHGen}$, $\mathtt{epk}_i \leftarrow g^x$, $\mathtt{sid}_i = ID_i \overset{\$}{\leftarrow} \{0,1\}^{32}$

$\mathtt{ltk} = \mathsf{AEAD}(\kappa_3, 0, g^u, H_3)$

$now \leftarrow \mathsf{Timestamp}()$

$\mathtt{time} \leftarrow \mathsf{AEAD.Enc}(\kappa_4, 0, H_4, now)$

$\mathtt{mac1} \leftarrow \mathsf{MAC}(\mathsf{H}(\mathtt{label}_3 \| g^v), \mathtt{type}\|0^3\|\mathtt{sid}_i\|\mathtt{epk}_i\|\mathtt{ltk}\|\mathtt{time})$

$\mathtt{mac2} \leftarrow \mathsf{MAC}(cookie, \mathtt{type}\|0^3\|\mathtt{sid}_i\|\mathtt{epk}_i\|\mathtt{ltk}\|\mathtt{time}\|\mathtt{mac1})$

$\mathtt{InitiatorHello} \leftarrow \mathtt{type}\|0^3\|\mathtt{sid}_i\|\mathtt{epk}_i\|\mathtt{ltk}\|\mathtt{time}\|\mathtt{mac1}\|\mathtt{mac2}$

$$\xrightarrow{\mathtt{InitiatorHello}}$$

$(y, g^y) \overset{\$}{\leftarrow} \mathsf{DHGen}$, $\mathtt{epk}_r \leftarrow g^y$, $\mathtt{sid}_r = ID_r \overset{\$}{\leftarrow} \{0,1\}^{32}$

$\mathtt{zero} \leftarrow \mathsf{AEAD.Enc}(\kappa_9, 0, H_9, \emptyset)$

$\mathtt{mac1} \leftarrow \mathsf{MAC}(\mathsf{H}(\mathtt{label}_3 \| g^u), \mathtt{type}\|0^3\|\mathtt{sid}_r\|\mathtt{sid}_i\|\mathtt{epk}_r\|\mathtt{zero})$

$\mathtt{mac2} \leftarrow \mathsf{MAC}(cookie, \mathtt{type}\|0^3\|\mathtt{sid}_r\|\mathtt{sid}_i\|\mathtt{epk}_r\|\mathtt{zero}\|\mathtt{mac1})$

$\mathtt{ResponderHello} \leftarrow \mathtt{type}\|0^3\|\mathtt{sid}_r\|\mathtt{sid}_i\|\mathtt{epk}_r\|\mathtt{zero}\|\mathtt{mac1}\|\mathtt{mac2}$

$$\xleftarrow{\mathtt{ResponderHello}}$$

$tk_i \leftarrow \mathsf{KDF}(C_9, \emptyset, 1)$

$tk_r \leftarrow \mathsf{KDF}(C_9, \emptyset, 2)$

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.

k	Seed value C_k	Key κ_k	Hash value H_k
1	$\mathsf{H}(\mathtt{label}_1)$	$\emptyset$	$\mathsf{H}(C_1\|\mathtt{label}_2)$
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	$(C_3, g^{uv}, 1)$	$(C_3, g^{uv}, 2)$	$\mathsf{H}(H_3\|\mathtt{ltk})$
5	$\emptyset$	$\emptyset$	$\mathsf{H}(H_4\|\mathtt{time})$
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	$\emptyset$	$\emptyset$	$\mathsf{H}(H_9\|\mathtt{zero})$

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.

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, 15] and more generally [17], 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 $H(\texttt{label}_3 \| g^v)$ 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 $\texttt{sid}_i$ and $\texttt{sid}_r$ 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 $\mathsf{eCK\text{-}PFS}$ 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 [20]), 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 $\mathsf{eCK\text{-}PFS}$ 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:

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

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

$\mathsf{KE}.\mathsf{PSKeyGen}(\lambda) \overset{\$}{\to} (psk, pskid)$ 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 $pskid$.

$\mathsf{KE}.\mathsf{EPKeyGen}(\lambda) \overset{\$}{\to} (ek, epk)$ 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 $\mathsf{KE}.\mathsf{ASKeyGen}(\lambda)$ n_P times to generate a public-key/secret-key pair (pk_i, sk_i) for each party $P_i \in \{P_1, \dots, P_{n_P}\}$ and delivers all public-keys pk_i for $i \in \{1, \dots, n_P\}$ to $\mathcal{A}$. The challenger $\mathcal{C}$ then randomly samples a bit $b \overset{\$}{\leftarrow} \{0,1\}$ 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 $\mathsf{eCK\text{-}PFS\text{-}PSK}$ 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:

- $\rho \in \{\mathtt{init}, \mathtt{resp}\}$ – 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.
- $pid \in \{1, \dots, n_P, \star\}$ – the intended communication partner, represented with $\star$ if unspecified. Note that the identity of the partner session may be set during the protocol execution, in which case pid can be updated once.
- $m_s \in \{0,1\}^* \cup \{\bot\}$ – the concatenation of messages sent by the session, initialised by $\bot$.
- $m_r \in \{0,1\}^* \cup \{\bot\}$ – the concatenation of messages received by the session, initialised by $\bot$.
- $kid \in \{0,1\}^* \cup \{\bot\}$ – the concatenation of public keyshare information received by the session, initialised by $\bot$.

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

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

```
$\mathsf{Reveal}(i, s)$:
if $(\pi_i^s.\alpha \neq \mathtt{accept})$ then
    return $\bot$
else
    $\mathbf{RSKflag}_i[s] \leftarrow$ corrupt
    return $\pi_i^s.k$
end if
```

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

if $(i = j) \vee (\mathbf{PSKflag}_i[j] \neq \bot)$ then
    return $\bot$
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
```

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

if $\mathbf{PSK}_i[j] = \bot$ then
    return $\bot$
end if
if $\mathbf{PSKflag}_i[j] \neq$ clean then
    return $\bot$
else
    $\mathbf{PSKflag}_i[j] \leftarrow$ corrupt
    $\mathbf{PSKflag}_j[i] \leftarrow$ corrupt
    return $\mathbf{PSK}_i[j]$
end if
```

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

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

```
$\mathsf{CorruptASK}(i)$:

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

```
$\mathsf{Test}(i, s)$:
if $(\mathsf{tested} = \mathtt{true}) \vee (\pi_i^s.\alpha \neq \mathtt{accept}) \vee (\pi_i^s = \bot)$ then
    return $\bot$
end if
tested $\leftarrow$ true
if $b = 0$ then
    return $\pi_i^s.k$
else
    return $k \xleftarrow{\$} \mathcal{K}$
end if
```

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

- $\alpha \in \{\mathtt{active}, \mathtt{accept}, \mathtt{reject}, \perp\}$ – 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.

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

- preshared keys $\{\boldsymbol{\mathsf{PSKflag}_1}, \boldsymbol{\mathsf{PSKflag}_2}, \ldots, \boldsymbol{\mathsf{PSKflag}_{n_P}}\}$ where for each element $\boldsymbol{\mathsf{PSKflag}_i}[j] \in \boldsymbol{\mathsf{PSKflag}_i}$, $\boldsymbol{\mathsf{PSKflag}_i}[j] \in \{\mathtt{corrupt}, \mathtt{clean}, \perp\}$ $\forall\ i,j \in [n_P]$ and $\boldsymbol{\mathsf{PSKflag}_i}[j] = \perp$ for $i = j$
- long-term keys $\{\mathsf{ASKflag}_1, \ldots, \mathsf{ASKflag}_{n_P}\}$, where $\mathsf{ASKflag}_i \in \{\mathtt{corrupt}$, $\mathtt{clean}$, $\perp\}$ $\forall\ i \in [n_P]$
- ephemeral keys $\{\boldsymbol{\mathsf{EPKflag}_1}, \ldots, \boldsymbol{\mathsf{EPKflag}_{n_P}}\}$, where $\boldsymbol{\mathsf{EPKflag}_i}[s] \in \{\mathtt{corrupt}$, $\mathtt{clean}$, $\perp\}$ $\forall\ i \in [n_P]$ and $s \in [n_S]$.
- session keys $\{\boldsymbol{\mathsf{RSKflag}_1}, \ldots, \boldsymbol{\mathsf{RSKflag}_{n_P}}\}$, where $\boldsymbol{\mathsf{RSKflag}_i}[s] \in \{\mathtt{corrupt}$, $\mathtt{clean}$, $\perp\}$ $\forall\ i \in [n_P]$ 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 2 (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:*

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

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

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 queries below. An algorithmic description 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.
- $\mathsf{CreatePSK}(i, j) \rightarrow \{pskid, \top, \perp\}$: allows the adversary to direct parties to generate a preshared key for use in future protocol executions.
- $\mathsf{Reveal}(i, s)$: allows the adversary access to the secret session key computed by a session during protocol execution.
- $\mathsf{CorruptPSK}(i) \rightarrow \{psk, \perp\}$: allows the adversary access to the secret preshared key jointly shared by parties prior to protocol execution.
- $\mathsf{CorruptASK}(i) \rightarrow \{sk_i, \perp\}$: allows the adversary access to the secret long-term key generated by a party prior to protocol execution.
- $\mathsf{CorruptEPK}(i, s) \rightarrow \{ek, \perp\}$: allows the adversary access to the secret ephemeral key generated by a session during protocol execution.
- $\mathsf{Send}(i, s, m) \rightarrow \{m', \perp\}$: allows the adversary to send messages to sessions for protocol execution and receive their output.
- $\mathsf{Test}(i, s) \rightarrow \{k, \perp\}$: sends the adversary a real-or-random session key used in determining the success of $\mathcal{A}$ in the key-indistinguishability game.

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 3 (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 4 ($\mathsf{clean}_{\mathsf{eCK\text{-}PFS\text{-}PSK}}$). *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* is a contributive keyshare session for *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 $\pi_i^s.psk = \bot$, 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 $\pi_i^s.psk = \bot$, 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

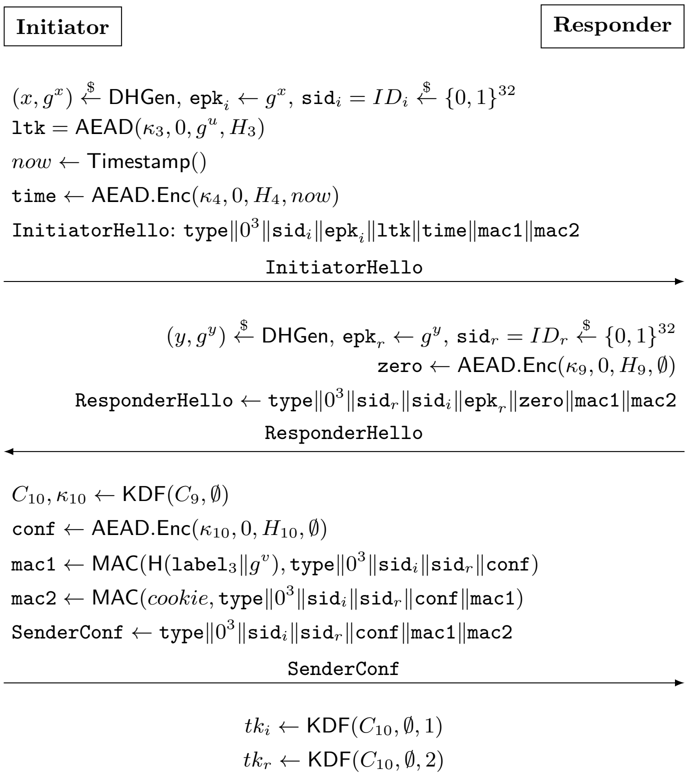
In this section we examine the security implications of modelling the WireGuard handshake as a 1-RTT key exchange protocol. We have already noted that this results in a KCI attack on the protocol, also observed in [11]. However, we note an arguably more serious attack on session key security in our eCK-PFS-PSK security model that results from this modelling. We discuss the implications of this attack 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 briefly describe 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 the full version [14].

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



Initiator **Responder**

$(x, g^x) \xleftarrow{\$} \mathsf{DHGen}$, $\mathtt{epk}_i \leftarrow g^x$, $\mathtt{sid}_i = ID_i \xleftarrow{\$} \{0,1\}^{32}$

$\mathtt{ltk} = \mathsf{AEAD}(\kappa_3, 0, g^u, H_3)$

$now \leftarrow \mathsf{Timestamp}()$

$\mathtt{time} \leftarrow \mathsf{AEAD.Enc}(\kappa_4, 0, H_4, now)$

$\mathtt{InitiatorHello}$: $\mathtt{type} \| 0^3 \| \mathtt{sid}_i \| \mathtt{epk}_i \| \mathtt{ltk} \| \mathtt{time} \| \mathtt{mac1} \| \mathtt{mac2}$

$\xrightarrow{\mathtt{InitiatorHello}}$

$(y, g^y) \xleftarrow{\$} \mathsf{DHGen}$, $\mathtt{epk}_r \leftarrow g^y$, $\mathtt{sid}_r = ID_r \xleftarrow{\$} \{0,1\}^{32}$

$\mathtt{zero} \leftarrow \mathsf{AEAD.Enc}(\kappa_9, 0, H_9, \emptyset)$

$\mathtt{ResponderHello} \leftarrow \mathtt{type} \| 0^3 \| \mathtt{sid}_r \| \mathtt{sid}_i \| \mathtt{epk}_r \| \mathtt{zero} \| \mathtt{mac1} \| \mathtt{mac2}$

$\xleftarrow{\mathtt{ResponderHello}}$

$C_{10}, \kappa_{10} \leftarrow \mathsf{KDF}(C_9, \emptyset)$

$\mathtt{conf} \leftarrow \mathsf{AEAD.Enc}(\kappa_{10}, 0, H_{10}, \emptyset)$

$\mathtt{mac1} \leftarrow \mathsf{MAC}(\mathsf{H}(\mathtt{label}_3 \| g^v), \mathtt{type} \| 0^3 \| \mathtt{sid}_i \| \mathtt{sid}_r \| \mathtt{conf})$

$\mathtt{mac2} \leftarrow \mathsf{MAC}(cookie, \mathtt{type} \| 0^3 \| \mathtt{sid}_i \| \mathtt{sid}_r \| \mathtt{conf} \| \mathtt{mac1})$

$\mathtt{SenderConf} \leftarrow \mathtt{type} \| 0^3 \| \mathtt{sid}_i \| \mathtt{sid}_r \| \mathtt{conf} \| \mathtt{mac1} \| \mathtt{mac2}$

$\xrightarrow{\mathtt{SenderConf}}$

$tk_i \leftarrow \mathsf{KDF}(C_{10}, \emptyset, 1)$

$tk_r \leftarrow \mathsf{KDF}(C_{10}, \emptyset, 2)$

Fig. 3: The modification to the WireGuard handshake that allows $\mathsf{eCK\text{-}PFS\text{-}PSK}$ security. The change is limited to an additional `SenderConf` message that contains the value $\mathtt{conf} \leftarrow \mathsf{AEAD}(\kappa_{10}, 0, H_{10}, \emptyset)$. 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.

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 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, and 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 3 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* $\mathsf{eCK\text{-}PFS\text{-}PSK}$-*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* $\mathsf{eCK\text{-}PFS\text{-}PSK}$ *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, $\mathsf{auth\text{-}aead}$, $\mathsf{sym\text{-}ms\text{-}PRFODH}$, $\mathsf{sym\text{-}mm\text{-}PRFODH}$ *and* ddh *assumptions.*

Due to space constraints, we point readers to the full version of this work [14] for a more detailed security statement, as well as full details of the proof.

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 $\mathsf{eCK\text{-}PFS\text{-}PSK}$ security model. This amends the previous $\mathsf{eCK\text{-}PFS}$ 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) $\mathsf{eCK\text{-}PFS\text{-}PSK}$ 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 [16]. However, this would require a different "record layer" modelling from that used for TLS in [16] to allow for packet loss and packet reordering. One could also implement our modification and measure its effect on the performance of WireGuard, but 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. We are grateful to Håkon Jacobsen and Benjamin Lipp as well as the anonymous reviewers for feedback on our work.

References

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 K. G. Paterson. A Cryptographic Analysis of the WireGuard Protocol. Cryptology ePrint Archive, Report 2018/080, January 2018. https://eprint.iacr.org/2018/080.
15. 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.
16. 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.
17. 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.
18. 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.
19. 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.
20. 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.
21. 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.
22. Y. Nir and A. Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 7539 (Informational), May 2015.
23. T. Perrin. The Noise Protocol Framework. October 2017. http://noiseprotocol.org/noise.html.