A Unified Symbolic Analysis of WireGuard

Pascal Lafourcade[*][†], Dhekra Mahmoud[*][†] and Sylvain Ruhault[‡]

[*]Université Clermont Auvergne
[†]Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes
[‡]Agence Nationale de la Sécurité des Systèmes d'Information

Abstract—WireGuard [22], [21] is a Virtual Private Network (VPN), presented at NDSS 2017, recently integrated into the Linux Kernel [53] and paid commercial VPNs such as NordVPN, Mullvad and ProtonVPN [52]. It proposes a different approach from other classical VPN such as IPsec [28] or OpenVPN [33] because it does not let users configure cryptographic algorithms. The protocol inside WireGuard is a dedicated extension of `IKpsk2` protocol from Noise Framework [47]. Different analyses of WireGuard and `IKpsk2` protocols have been proposed, in both the symbolic and the computational model, with or without computer-aided proof assistants. These analyses however consider different adversarial models or refer to incomplete versions of the protocols. In this work, we propose a unified formal model of WireGuard protocol in the symbolic model. Our model uses the automatic cryptographic protocol verifiers SAPIC+, PROVERIF and TAMARIN. We consider a complete protocol execution, including cookie messages used for resistance against denial of service attacks. We model a precise adversary that can read or set static, ephemeral or pre-shared keys, read or set `ecdh` pre-computations, control key distribution. Eventually, we present our results in a unified and interpretable way, allowing comparisons with previous analyses. Finally thanks to our more model, we confirm a flaw on the anonymity of the communications and point an implementation choice which considerably weakens its security. We propose a remediation that we prove secure using our models.

I. INTRODUCTION

During the last decades several complex cryptographic protocols have been constructed to offer more security and more services to the users. In the same time, automated analysis has made its way into mainstream security practice and many tools for automatic formal verification of cryptographic protocols have been designed and are in continuous progress [4]. For instance, the design of TLS 1.3 took several years and many formal security analyses have been done on the subject [7]–[10], [17], [19], [27], [29], [40], [41], [46], [48]. Still some attacks exist on TLS 1.3 [13], [45], [46]. The same story also occurs with IKE [11], [14], [18], [26], [36]. These two examples show that security assessment of cryptographic protocols is a process, in which all works are used as basis for the following ones, which are more precise. Tools such as PROVERIF and TAMARIN, used to assess these protocols, are given an abstract description of the protocol and its security properties, and give a proof that no attack exists within their model or find an attack violating the security properties. Symbolic verification tools do not directly operate on the cryptographic definition of cryptographic primitives but on an approximation and consider abstract definitions of their behaviors. Those abstractions make some attacks impossible to capture for some tools, because sometimes the tool is not able to deal with some algebraic properties which let those attacks hidden. Likewise, the symbolic model of the protocol itself or the modeling of the security property may not be precise enough and therefore the tool cannot find attacks. All these successive works constitute important steps to have more secure communication protocols as it is shown in [46]. In this paper we focus on WireGuard, a recent VPN, largely deployed, used and formally analyzed. Our aim is to propose a new symbolic model that aggregates and enriches all existing models, in terms of messages, adversaries and properties modeling.

A. Our Contribution

Previous analyses assessment and new model proposal. We first review previous analyses of WireGuard and `IKpsk2` and we point disparities between assessed properties, modeled protocols and adversary models: symbolic analyses involve different models and different proof assistants (PROVERIF and TAMARIN), computational analyses are manual or use proof assistant (CRYPTOVERIF). This allows to identify the most complete analyses: although incomplete, model from [42] is the closest to WireGuard protocol and threat model from [30] captures the largest number of adversarial capabilities. We propose a new symbolic model that enriches these analyses with a more complete protocol execution and more precise adversaries. We use the tool SAPIC+: our model is fully based on processes and all properties are verified in PROVERIF and in TAMARIN. Our adversaries can read or set static, ephemeral or pre-shared keys, read or set `ecdh` pre-computations, and control key distribution.

Anonymity. Our models allow us to confirm a flaw on WireGuard related to anonymity. This flaw has previously been identified in a computational analysis of the protocol [42], our contribution is to prove it in a new symbolic analysis, based on observational equivalence, on a more precise protocol model. Importantly, this flaw allows an attacker to identify a VPN user even if this user hides behind an access point, because this flaw is cryptographic and does not rely on the underlying transport protocol.

Pre-computation. When setting an interface, Wire-Guard implements a pre-computation of ecdh products between static keys to speed up message consumption. Peers have a specific field that contains pre-computed ecdh products (e.g., in Linux Kernel, this field is named `precomputed_static_static` and in user-land Go implementation, this field is named `precomputedStaticStatic`): at interface setting, all peers public keys are read and ecdh product between interface private key and public key is computed and field's value is set with computation's result. This implementation optimization increases attack surface as it allows an adversary to potentially get access to ecdh pre-computation and therefore requires a clear assessment.

(a) Safe environment. (b) Unsafe environment.

Fig. 1: Potential vulnerability against Wireguard.

To illustrate importance of pre-computation assessment, we describe in Fig.1 a potential vulnerability against one WireGuard implementation. Default storage for static private keys is in configuration file. Hence an attacker that accesses these files compromises static private keys. To mitigate the risk, static private keys can be generated and stored in smart card which can perform ecdh operations. An example of such card is OpenPGP embedded in YubiKey [44]. Security objective in this context is to protect against an attacker that has access to files and memory of the WireGuard process, but shall not be able to compromise static keys stored in smart card. In this implementation, OpenPGP on YubiKey is used to generate and store a static private key, because a support for ecdh is mandatory, as recent YubiKey has. Then Go version of WireGuard is used, which provides a full user-space implementation. Such an architecture aims at mitigating memory leakage, as static key is protected by the smart card. Once interface is mounted, an attacker could however access ecdh product in process memory, due to pre-computation. Precisely, we consider an Initiator of static private key u, embedded in a smart card 🔲, which uses a WireGuard client ⛨ and a network configuration 🔳 that contains Responder's public key g^v. When the smart card is plugged and WireGuard interface is mounted, ecdh product g^{uv} is pre-computed in memory 🔳. In a safe environment, depicted in Fig.1a, this pre-computation is not compromised, however, in an unsafe environment, depicted in Fig.1b, an attacker 🐞 can corrupt memory 🔳 and pre-computed ecdh product g^{uv}, while private key u in smart card 🔲 remains safe. Our contribution is to consider a symbolic adversary model that enhances model from [30] with pre-computations, adapted to a complete protocol model of WireGuard. Note that compromise of ecdh

pre-computation is *weaker* than compromise of private static keys: if adversary has access to a private key, adversary knows ecdh product, however, the opposite is false. We show that in contradiction with this, an adversary that has access to pre-computation is *as powerful* as an adversary that has access to static private keys.

Mitigation. We finally propose and prove recommendations that counter the attack against anonymity and enhance security of protocol implementation.

B. Related Work

WireGuard. Computer-aided cryptography has allowed the analysis of a large number of protocols, in both the symbolic and the computational models [4]. In line with this, assessment of WireGuard through rigorous security analysis is at the core of the product: WireGuard has itself been analyzed in the symbolic model with TAMARIN [23], in the computational model with CRYPTOVERIF [42] and a (non computer-aided) analysis has been proposed in [24]. WireGuard uses a dedicated protocol that relies on an ecdh key exchange from the Noise framework [47], named `IKpsk2`. Protocols from this framework have also been analyzed in the symbolic model: complete analyses have been proposed with PROVERIF [35] and with TAMARIN [51], which allowed to confirm claimed security properties from the framework. A complementary analysis is proposed in [30], also with TAMARIN. In line with the (non computer-aided) analysis done on WireGuard, an analysis of Noise protocols in the computational model is presented in [25]. In the context of Post-Quantum security, [2] proposes a slight modification of WireGuard and [32] presents a new protocol, based on WireGuard original specification, and revisits symbolic analysis of [23] and computational one of [24].

SAPIC$^+$. This is a new tool, unifying the use of PROVERIF and TAMARIN for symbolic analyses that has recently been used to assess EDHOC protocol [34]. Our contribution is to propose a new model of WireGuard protocol in SAPIC$^+$ that enriches all previous symbolic models and allows to verify security properties in both PROVERIF and TAMARIN. We propose a methodology which combines all tools and benefits from the strength of each.

C. Outline

Section II proposes an overview of symbolic model, including insights related to the applied Π-Calculus, SAPIC$^+$, PROVERIF and TAMARIN provers. Section III provides a high level overview of WireGuard. In Section IV we assess previous analyses and finally in Section V we describe our new model. In Section VI, we present our results.

II. SYMBOLIC MODEL

In the symbolic model, as explained in [12], [16], cryptographic primitives are considered as perfect, modeled by function symbols in an algebra of terms, possibly with equations. Messages are *terms* on these primitives and the adversary can symbolically compute new terms only using these primitives

(and equations that model them). In particular, an adversary can decipher an encrypted term only if it has access to the corresponding secret key. In this model, the attacker is considered to have complete control of the network: he can eavesdrop, remove, substitute, duplicate and delay messages that the parties are sending one another, and insert messages of his choice on the public channels (Dolev-Yao attacker [20]).

Several tools exist for verifying security protocols in the symbolic model [4]. The applied Π-Calculus, an extension of the Π-Calculus, is among the most used languages for modeling security protocols and constitutes the input language of many tools. For the full details about its syntax and semantics, see [1]. The protocol verifier SAPIC$^+$ [15] takes as input a protocol description in (a variant of) the applied Π-Calculus similar to PROVERIF and security properties specifications expressed exactly as in TAMARIN. Through its exports, SAPIC$^+$ supports the union of the theories supported by TAMARIN and PROVERIF.

---

query k; event(Accept(k)) $\implies$ event(Honest(k))

lemma A: "**All** $k \# i.$Accept(k)@$i \implies$ **Ex** $\#j.$Honest(k)@$j \& j < i$"

Fig. 2: PROVERIF query and TAMARIN lemma.

In PROVERIF protocols are described in the applied Π-Calculus. The grammar supports *events* which are annotations that do not change processes' behavior, but are inserted at precise locations to allow reasoning about protocols' executions. For instance, to express that, whenever the process accepts a key k, then k must have been honestly generated, we write **query** as in Fig.2. It requires that each occurrence of the event *Accept* is preceded by an earlier occurrence of the event *Honest* on every execution trace. The tool can handle many cryptographic primitives (encryption, signature, hash function, Diffie-Hellman Key agreement) specified as rewrite rules or equations. PROVERIF can also handle an unbounded number of sessions and an unbounded message space. It is able to prove *Traces* properties: *Secrecy* (as reachability properties: the adversary cannot obtain the secret), *Agreement* (as correspondence properties: if an event has been executed, then other events have been executed as well) and *Equivalences* between processes (as observational equivalence: the adversary cannot distinguish two processes).

In TAMARIN protocols are described using multiset rewrite (MSR) rules of the form: $[lhs] - [actions] \rightarrow [rhs]$. They describe the manipulation of the protocol state represented as a multiset of facts. Intuitively, for such a rule to fire, we require the (multiset of) facts on the left-hand side lhs. Then these facts are deleted, and the facts on the right-hand side rhs are added. The actions are facts that annotate rules, and will be used to specify properties. TAMARIN uses a temporal logic to express security properties about the protocol executions, modeled as traces, which are the sequence of actions triggered by MSR rules. For instance, to express the same property stated in the previous paragraph, we write **lemma** as in Fig.2. It requires that if an *Accept* action was raised for k at any

time point i of the trace, then the *Honest* action must have been raised for the same value k at a previous time point j. Trace properties are expressed as lemmas that must be valid on all traces. Equivalences between processes are expressed as observational equivalence as well.

III. WIREGUARD DESCRIPTION

We present WireGuard [21] and also assess the link between WireGuard and Noise protocols [47].

A. WireGuard Protocol

Notations. Bistrings are delimited with square brackets $[\cdots]$, $\varnothing$ is the empty bitstring, of length 0, if A and B are bitstrings, $[A\|B]$ is the concatenation of A and B, $|A|$ is the length of A and $\{A\}$ is an encryption of A; $0, 1, 2, 3, 4$ denote bitstrings that correspond to bytes 0, 1, 2, 3 and 4, respectively; if n is an integer, A^n is the n-bytes concatenation $[A\|\cdots\|A]$.

Overview. WireGuard involves two actors: *Initiator* and *Responder*, also referred to as *peers*. The protocol is composed of two phases: a key exchange phase and a transport phase. Key exchange phase involves two messages, `InitHello` and `RecHello`, transport phase involves one message, `TransData`. WireGuard involves a fourth message `CookieRep`, for protection against denial of service attacks.

WireGuard does not define the notions of *client* and *server*, peers can indifferently play the role of Initiator or Responder, a peer that starts a session is considered as an Initiator, the other peer the Responder. Key exchange is considered complete *after* the first message from the transport phase (which *must* be a message from Initiator), hence the protocol involves a **1.5-RTT** key exchange. Also explained in [21], protocol requires an out of band data share, not considered as part of the protocol. A *session* between two peers refers to a successful key exchange and the use of this key for data transport, a new exchange means a new session.

Cryptographic Primitives. WireGuard uses a a cyclic group $\mathbb{G}$, of generator g and a closed set of cryptographic primitives: a hash function h, two message authentication codes hmac and mac, three key derivations $\mathsf{kdf}_1, \mathsf{kdf}_2, \mathsf{kdf}_3$, two authenticated encryption algorithms aead and xaead and a padding scheme pad. Initiator and Responder use the following material: $u, U = g^u$ is Initiator static key pair, $x, X = g^x$ is Initiator ephemeral key pair, $v, V = g^v$ is Responder static key pair, $y, Y = g^y$ is Responder ephemeral key pair, ts is a timestamp, psk is an optional pre-shared key and C, l and M are public constants. In WireGuard, these are instantiated as follows:

- $\mathbb{G}$ is the group of points of curve Curve25519 [6], [38], of base point g.
- $h \leftarrow$ h(I) is the computation of a 32-byte fingerprint h from input I with hash function Blake2s [3].
- $M \leftarrow$ hmac(K, I) is the computation of a 32-byte message authentication code M from key K and input I with hash function Blake2s as described in [31].
- $M \leftarrow$ mac(K, I) is the computation of a 16-byte message authentication code M from input I and key K with Blake2s hash function, as described in [50].

- $\tau_1 \leftarrow \mathsf{kdf}_1(K, I)$, $(\tau_1, \tau_2) \leftarrow \mathsf{kdf}_2(K, I)$ and $(\tau_1, \tau_2, \tau_3) \leftarrow \mathsf{kdf}_3(K, I)$ are key derivations from [37]. They compute 32-byte t-uples from key K and input I:
  - $\tau_0 = \mathsf{hmac}(K, I)$; $\tau_1 = \mathsf{hmac}(\tau_0, 1)$, $\mathsf{kdf}_1(K, I) = \tau_1$;
  - $\tau_2 := \mathsf{hmac}(\tau_0 \| \tau_1, 2)$, $\mathsf{kdf}_2(K, I) := (\tau_1, \tau_2)$;
  - $\tau_3 := \mathsf{hmac}(\tau_0 \| \tau_1 \| \tau_2, 3)$, $\mathsf{kdf}_3(K, I) := (\tau_1, \tau_2, \tau_3)$.
- $(C, T) \leftarrow \mathsf{aead}(K, N, P, A)$ is the encryption algorithm `AEAD_CHACHA20_POLY1305` from [39], which itself combines ChaCha20 and Poly1305 algorithms. From a key K, a 12-byte nonce N, a plaintext P and an authentication data A, it computes a ciphertext C of length $|P|$ bytes and a 16-bytes authentication tag T, hence its total byte-length is $|P| + 16$. The 12-bytes nonce N is 0^4 followed by a 8-byte counter.
- $(C, T) \leftarrow \mathsf{xaead}(K, N, P, A)$ is the encryption algorithm `AEAD_XCHACHA20_POLY1305` which is a variant of `AEAD_CHACHA20_POLY1305` where the nonce N is a random 24-bytes string.
- $P \| 0^{16 \lceil |P|/16 \rceil - |P|} \leftarrow \mathsf{pad}(P)$ is the padding algorithm.

Messages content. Messages of WireGuard protocol are depicted in Fig.3 and Fig.4. `InitHello` message is $[1 \| 0^3 \| s_i \| X \| \{U\} \| \{ts\} \| \mathsf{mac}_1^i \| \mathsf{mac}_2^i]$, where 1 and 0^3 are constant bitstrings, s_i is a random session identifier, X is Initiator ephemeral key, $\{U\}$ is encrypted Initiator's static public key, $\{ts\}$ is an encrypted timestamp, mac_1^i is a first message authentication code. Depending on `CookieRep` message, mac_2^i takes two values: either 0^{16} or a second message authentication code. Message `RecHello` is $[2 \| 0^3 \| s_r \| s_i \| Y \| \{\varnothing\} \| \mathsf{mac}_1^r \| \mathsf{mac}_2^r]$, where 2 and 0^3 are constant bitstrings, s_r, s_i are session identifiers, Y is Responder ephemeral key, $\{\varnothing\}$ is an encryption of empty string, mac_1^r and mac_2^r are similar as for `InitHello` message. Message `TransData` from Initiator to Responder is $[3 \| 0^3 \| s_r \| i_k \| \{\mathsf{pad}(P_{i_k})\}]$, from Responder to Initiator is $[3 \| 0^3 \| s_i \| r_k \| \{\mathsf{pad}(P_{r_k})\}]$, where 3 and 0^3 are constant bitstrings, s_i and s_r are session identifiers, $\{\mathsf{pad}(P_{i_k})\}$ and $\{\mathsf{pad}(P_{r_k})\}$ are padded and encrypted payloads. Finally, message `CookieRep` is $[4 \| 0^3 \| s_i \| \rho \| \{\tau\}]$, where 4 and 0^3 are constant bitstrings, s_i is a session identifier, ρ is a random nonce and $\{\tau\}$ is an encrypted cookie.
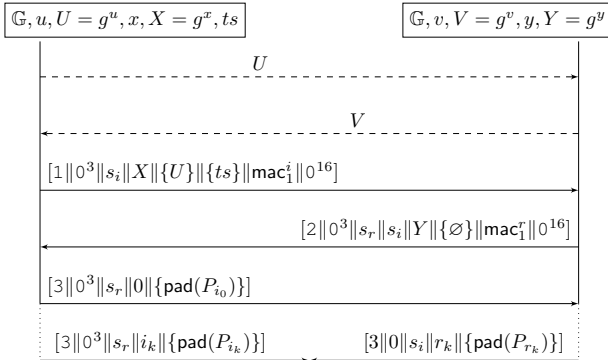

Fig. 3: WireGuard (without cookies).

Message computation. To compute `InitHello`, Initiator uses public values C and I, generates a random session identifier s_i, computes successive hash values h_t, key values k_t and chaining values c_t: $ck = \mathsf{h}(\mathsf{C})$, $h_0 = \mathsf{h}(ck \| \mathsf{I})$, $h_1 = \mathsf{h}(h_0 \| V)$, $c_0 = \mathsf{kdf}_1(ck, X)$, $h_2 = \mathsf{h}(h_1 \| X)$, $(c_1, k_1) = \mathsf{kdf}_2(c_0, g^{xv})$, $\{U\} = \mathsf{aead}(k_1, 0, h_2, U)$, $h_3 = \mathsf{h}(h_2 \| \{U\})$, $(c_2, k_2) = \mathsf{kdf}_2(c_1, g^{uv})$, $\{ts\} = \mathsf{aead}(k_2, 0, h_3, ts)$, $h_4 = \mathsf{h}(h_3 \| \{ts\})$. Finally, a message authentication code is appended to the message, computed on the bitstring $[1 \| 0^3 \| s_i \| X \| \{U\} \| \{ts\}]$, with key $\mathsf{h}(\mathsf{M} \| V)$ (i.e., key is derived from public value M and Responder public key V) and `InitHello` $= [1 \| 0^3 \| s_i \| X \| \{U\} \| \{ts\} \| \mathsf{mac}(\mathsf{h}(\mathsf{M}, V), [1 \| \cdots \| \{ts\}]) \| 0^{16}]$. At reception, Responder performs the necessary computations to obtain the same hash and key values, decrypts $\{U\}$, checks that U is legitimate and decrypts $\{ts\}$.

To compute `RecHello`, Responder generates a random session identifier s_r, computes the next hash values h_t, key values k_t and chaining values c_t: $c_3 = \mathsf{kdf}_1(c_2, Y)$, $h_5 = \mathsf{h}(h_4 \| Y)$, $c_4 = \mathsf{kdf}_1(c_3, g^{xy})$, $c_5 = \mathsf{kdf1}(c_4, g^{uy})$, $(c_6, h_{rt}, k_6) = \mathsf{kdf}_3(c_5, 0)$ if psk $= \varnothing$, $(c_6, h_{rt}, k_6) = \mathsf{kdf}_3(c_5, \mathsf{psk})$ if psk $\neq \varnothing$, $h_6 = \mathsf{h}(h_5 \| h_{rt})$, $\{\varnothing\} = \mathsf{aead}(k_6, 0, h_6, \varnothing)$, $h_7 = \mathsf{h}(h_6 \| \{\varnothing\})$. Similarly as for `InitHello`, a message authentication code is appended to the message, computed on the bitstring $[2 \| 0^3 \| s_r \| s_i \| Y \| \{\varnothing\}]$, with key $\mathsf{h}(\mathsf{M}, U)$ (i.e. key is derived from public value M and Initiator public key U). Finally, `RecHello` $= [2 \| 0^3 \| s_r \| s_i \| Y \| \{\varnothing\} \| \mathsf{mac}(\mathsf{h}(\mathsf{M}, U), [2 \| \cdots \| \{\varnothing\}]) \| 0^{16}]$. At reception, Initiator performs the necessary computations to obtain the same hash and key values, decrypts $\{\varnothing\}$ and checks that the obtained value is $\varnothing$.

After `InitHello` and `RecHello`, both Initiator and Responder share a common session key k_6. From this key they derive two keys $(C^i, C^r) = \mathsf{kdf}_2(k_6, \varnothing)$ and use these keys, respectively, to protect data from Initiator to Responder and from Responder to Initiator.

To compute `TransData`, Initiator takes received Responder session identifier s_r, current counter value i_k, computes $\{\mathsf{pad}(P_{i_k})\} = \mathsf{aead}(C^i, i_k, \mathsf{pad}(P_{i_k}), \varnothing)$ where P_{i_k} is the plaintext sent at this step and $\mathsf{pad}(P_{i_k})$ is the padded plaintext. Responder performs same computation with Initiator session identifier s_i, current counter value r_k and plaintext P_{r_k}. Note that first transport message is always from Initiator to Responder. For WireGuard, counter maximal value is 2^{60} (i.e at most 2^{60} transport messages are encrypted with same session key).

WireGuard protocol embeds a protection against denial of service, based on `CookieRep` messages. To build such message, WireGuard uses information from transport layer, as messages are transported in UDP datagrams [49]: `InitHello` message is transported in packet $[\mathsf{IP}_i \| \mathsf{IP}_r \| \mathsf{Port}_i \| \mathsf{Port}_r \| \mathtt{InitHello}]$ where IP_i and Port_i are public IP and port for Initiator and IP_r and Port_r are public IP and port for Responder. Responder generates a random value R_m, uses IP_i and Port_i from incoming packet and computes the cookie value $\tau = \mathsf{mac}(R_m, \mathsf{IP}_i \| \mathsf{Port}_i)$. This cookie is then en-

Fig. 4: WireGuard (with cookies).

(a) KK and KKpsk2.

(b) IK and IKpsk2.

Fig. 5: KK, KKpsk2, IK and IKpsk2 protocols, where blue bold denotes optional fields.
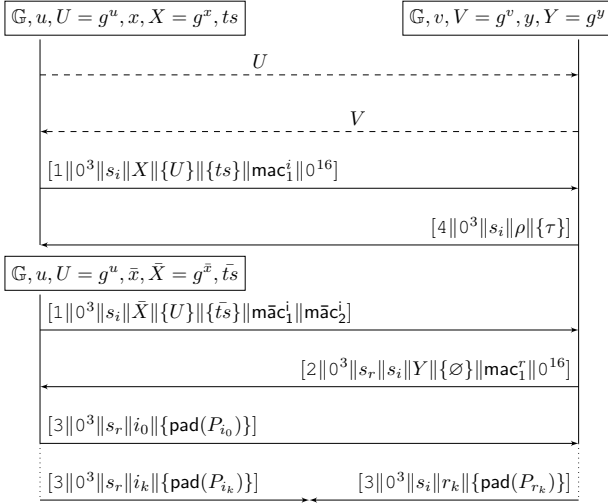
crypted: Responder generates a random nonce ρ and computes $\{\tau\} = \mathsf{xaead}(\mathsf{h}(V), \mathsf{mac}_1^i, \rho, \tau)$, where mac_1^i is extracted from InitHello message. At reception, Initiator decrypts τ, generates a new InitHello message, with same session identifier s_i, a new ephemeral key pair $\bar{x}, \bar{X} = g^{\bar{x}}$, a new timestamp $\bar{ts}$ and a new authentication code $\mathsf{m\bar{a}c}_1^i$ as before, except that now it appends a second authentication code to the message, computed on the first 7th fields $[1\|0^3\|s_i\|\bar{X}\|\{U\}\|\{\bar{ts}\}\|\mathsf{m\bar{a}c}_1^i]$, with key the cookie value τ. At reception, Responder verifies this additional authentication code and continues the protocol as before.

B. WireGuard and Noise Protocols

The Noise framework [47] defines a set of key exchange protocol, among which are the protocols named IK, KK, IKpsk2 and KKpsk2, presented in Fig.5. These four protocols are referenced in WireGuard documentation and source code as the basis for the key exchange protocol inside Wire-Guard.

WireGuard uses IKpsk2 (and not IK, nor KK, nor KKpsk2). At first glance, as pointed in [2], it seems that WireGuard is closer to KKpsk2 than IKpsk2 because of the initial out of band public keys exchange. However, in KKpsk2, Initiator knows to whom it sends InitHello message and Responder *knows from whom it receives it*, whereas in IKpsk2, Initiator knows to who it sends InitHello message but Responder does not know from whom it receives it. An application built upon KKpsk2 shall ensure both parties know to whom they communicate before starting key exchange, whereas an application built upon IKpsk2 can accept Responder does not know *a priori* who sends an InitHello message, but Responder shall be able to assess if received message is acceptable. This is exactly the path followed by WireGuard: each party has a set of acceptable peers (a list of acceptable public keys), but *discovers* who initiates a key exchange and *checks* it is acceptable during key exchange. WireGuard lets peers use an optional pre-shared key that shall
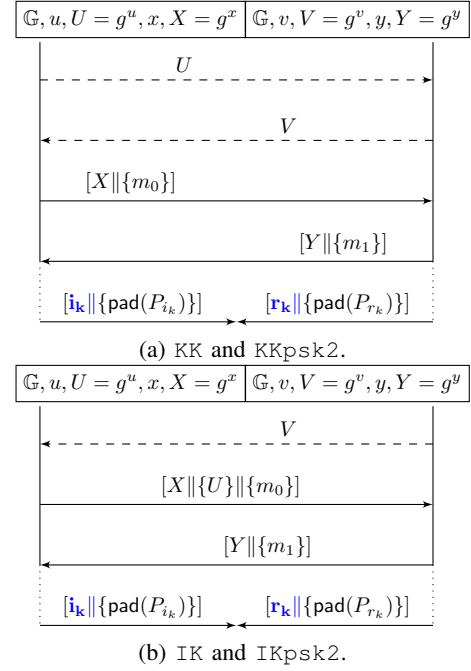
be shared beforehand. When this option is not chosen (which is described as $\mathsf{psk} = \varnothing$), WireGuard still implements IKpsk2: a difference between IK and IKpsk2 is that for IKpsk2 ephemeral keys X and Y are included in the derivation of the session key, which is not the case for IK. This corresponds to the computation of chaining values $c_0 = \mathsf{kdf}_1(ck, X)$ and $c_3 = \mathsf{kdf}_1(c_2, Y)$ (see below) which are specific to IKpsk2. As a consequence, the protocol from Noise framework we compare to WireGuard is IKpsk2. We need however to point similarities and differences.

Similarities between WireGuard and IKpsk2. Definition of IKpsk2 uses the same set of abstract algorithms as WireGuard (a cyclic group $\mathbb{G}$, of generator g, a hash function h, message authentication codes hmac and mac, key derivations $\mathsf{kdf}_1, \mathsf{kdf}_2, \mathsf{kdf}_3$, authenticated encryption algorithms aead and a padding scheme pad) except that IKpsk2 does not instantiate them. It is up to the application based on IKpsk2 to choose cryptographic primitives, as Wire-Guard does. Computation of keys for IKpsk2 is similar to WireGuard: with message m_0, Initiator computes successive hash values h_t, key values k_t and chaining values c_t as follows: $ck = \mathsf{h}(C)$, $h_0 = \mathsf{h}(ck\|\mathsf{I})$, $h_1 = \mathsf{h}(h_0\|V)$, $c_0 = \mathsf{kdf}_1(ck, X)$, $h_2 = \mathsf{h}(h_1\|X)$, $(c_1, k_1) = \mathsf{kdf}_2(c_0, g^{xv})$, $\{U\} = \mathsf{aead}(k_1, 0, h_2, U)$, $h_3 = \mathsf{h}(h_2\|\{U\})$, $(c_2, k_2) = \mathsf{kdf}_2(c_1, g^{uv})$, $\{ts\} = \mathsf{aead}(k_2, 0, h_3, m_0)$, $h_4 = \mathsf{h}(h_3\|\{m_0\})$. Similarly, with message m_1, Responder computes the next hash values h_t, key values k_t and chaining values c_t as follow: $c_3 = \mathsf{kdf}_1(c_2, Y)$, $h_5 = \mathsf{h}(h_4\|Y)$, $c_4 = \mathsf{kdf}_1(c_3, g^{xy})$, $c_5 = \mathsf{kdf}_1(c_4, g^{uy})$, $(c_6, h_{rt}, k_6) = \mathsf{kdf}_3(c_5, 0)$ if $\mathsf{psk} = \varnothing$, $(c_6, h_{rt}, k_6) = \mathsf{kdf}_3(c_5, \mathsf{psk})$ if $\mathsf{psk} \neq \varnothing$, $h_6 = \mathsf{h}(h_5\|h_{rt})$,

$\{m_1\} = \mathsf{aead}(k_6, 0, h_6, m_1)$, $h_7 = \mathsf{h}(h_6\|\{m_1\})$.

IKpsk2 relies on an initial out of band pre-message from Responder to Initiator in which Responder sends its static public key and conversely Initiator sends its static public key in first message. IKpsk2 does not specify how these keys are validated: indeed, Noise specification states *it's up to the application to determine whether the remote party's static public key is acceptable*. WireGuard specification is similar as it states *WireGuard rests upon peers exchanging static public keys with each other*.

Differences between WireGuard and IKpsk2. Wire-Guard involves a **1.5-RTT** key exchange: after messages InitHello and RecHello are correctly sent and received, a first TransData shall be sent from Initiator. After this first TransData message any peer can send other TransData messages. This feature is however not mandatory in Noise protocols and hence in IKpsk2: in IKpsk2, after handshake, transport messages can be either from Initiator to Responder or from Responder to Initiator. This feature is captured differently in previous analyses.

IKpsk2 first message is $[X\|\{U\}\|\{m_0\}]$, whereas Wire-Guard InitHello message is $[1\|0^3\|s_i\|X\|\{U\}\|\{ts\}$ $\|\mathsf{mac}_1^i\|\mathsf{mac}_2^i]$, where mac_2^i can equal 0^{16}. Hence with $m_0 = ts$, messages are similar, however they differ: InitHello has header $[1\|0^3]$, embeds session identifier s_i and fields mac_1^i and mac_2^i. Similarly, IKpsk2 second message is $[Y\|\{m_1\}]$, whereas WireGuard RecHello is $[2\|0^3\|s_r\|s_i\|Y\|\{\varnothing\}\|\mathsf{mac}_1^r\|\mathsf{mac}_2^r]$. Hence with $m_1 = \varnothing$, messages are similar but differ due to header, session identifiers and mac_1^r and mac_2^r fields. Finally transport messages also differ as WireGuard TransData includes header, session identifier and transmits counter in clear. Note that Noise specification [47] allows this clear counter transmission.This transmission is captured differently in previous IKpsk2 analyses. Due to these differences, we consider IKpsk2 as an *incomplete* version of WireGuard.

IV. PREVIOUS ANALYSES ASSESSMENT

In this section we present the security properties analyzed in previous studies, modeled protocols and adversary models.

A. Security Properties of Previous Analyses

WireGuard. Symbolic analyses of WireGuard are proposed in [23] and in [32] (for Post-Quantum WireGuard), both with TAMARIN prover. These two analyses are close as the one from [32] is an update of the one from [23] to account for Post-Quantum version. First [23] symbolically defines *Correctness*, *Key Agreement*, *Key Secrecy*, *Session Uniqueness* and *Identity Hiding*, *Key Secrecy* and *weak Perfect Forward Secrecy* as trace properties. With a model updated from [23], [32] symbolically defines *Session Key Secrecy*, *Perfect Forward Secrecy*,*Session Key Uniqueness*, *Entity Authentication*, *Identity Hiding* and *Deny of Service Protection*, also as trace properties. Difference between *weak Perfect Forward Secrecy* (from [23]) and *Perfect Forward Secrecy* (from [32]) is that the former is defined for a passive adversary while the latter is for an active adversary. *Session Key Secrecy* and *Key Secrecy* refer to the same property, which means that the session key is not known to the adversary. Similarly, *Session Uniqueness* and *Session Key Uniqueness* refer to the same property, which means that different sessions will have different keys. Each security property is tested against a unique key compromise scenario, for which the test succeeds, however this has the strong limitation that other key compromise scenarios are not included in the analyses. Our contribution is to provide an assessment for *a large set of* key compromise scenarios.

IKpsk2. Symbolic analyses of IKpsk2 are proposed in [35] (with PROVERIF prover) and [30] (with TAMARIN prover). [35] symbolically defines agreement and secrecy to fit properties that are informally described in [47], as trace properties. This leads to a restricted analysis as the resulting definitions are only tested against a specific key compromise scenario, hence this analysis shares the same default as [23] and in [32]. As opposed to all previous analysis, [30] proposes a different approach: analyze protocols from the Noise framework against security properties which are *not* the ones informally defined in [47] but are precise standard properties: *secrecy of payloads*, *non-injective agreement* and *injective agreement* on messages as defined in [43], and *anonymity*. Secrecy and agreement are modeled as trace properties while anonymity is modeled with observational equivalence. Furthermore, this analysis assesses *a large set of* key compromise scenarios, including a fine-grained assessment of perfect forward secrecy, which depends on both static keys but also on pre-shared key. We use this analysis as a reference for our WireGuard model in SAPIC$^+$ and we enhance it to assess ecdh pre-computation.

B. Models Assessment

We point disparities between previous models: different protocols are modeled, that all correspond to *incomplete* version of WireGuard. Fig.6 describes these models. Fig.6a describes our model that unifies and enriches all existing models. Note that on left side of Fig.6, all models include initial key distribution (which can be potentially compromised), while on right side, all models assume a safe initial key distribution.

Fig.6b models a protocol composed of three messages: two first messages InitHello, RecHello and a transport message that does not correspond exactly to WireGuard as the encrypted data is $\varnothing$, from Initiator to Responder. This protocol is used in computational analysis of [24]. Fig.6c models a protocol with pre-messages for static keys (U and V) distribution, two first incomplete messages as they correspond to InitHello and RecHello messages without message authentication codes, a first transport message that is in one direction and then transport messages TransData that can be in either direction, with counter in clear. This model is used in computational analysis of [42]. Fig.6d models a protocol composed of three messages used in symbolic analysis of [23]: two first messages that do not correspond exactly to Wire-Guard as message authentication codes in both InitHello and RecHello are replaced by constant bitstrings **MAC1**

(a) Our WireGuard Model.

(b) WireGuard Model from [24].

(c) WireGuard Model from [42].

(d) WireGuard Model from [23].

(e) IKpsk2 Model from [30].

(f) IKpsk2 Model from [35].

Fig. 6: Comparison with other models, where for each model, blue bold denotes part of the protocol that is precisely defined in our model but not in the model, hence for each model, differences with our model are highlighted.

and **MAC2**, followed by the original WireGuard transport message TransData, from Initiator to Responder. Fig.6e concerns IKpsk2: it models a protocol with pre-messages for static keys (U and V) distribution, two first messages and transport messages that can be in either direction, with counter in clear. It is used in symbolic analysis of [30]. Fig.6f concerns IKpsk2 limited to the two first messages (without initial key distribution) and transport messages that can be in either direction, without counters. It is used in symbolic analysis of [35].

It appears that the most precise model is the computa-

tional model of [42], which is however still incomplete. Our contribution is an enriched model, more precise than the computational one, for symbolic analysis.

C. Adversary Models

We also point disparities between *adversary models* in Table I. [23], [35] and [24] all capture security against key leakage and do not consider key modification, [30] and [42] capture key leakage *and* key modification. Finally [35] captures static and pre-shared key compromise while all others [23], [24], [30], [42], capture static, ephemeral and pre-

Reference	[23]	[35]	[30]	[24]	[42]	[32]	This work
WireGuard	✓			✓	✓	✓	✓
Noise IKpsk2		✓	✓				
Model							
Symbolic	✓	✓	✓			✓	✓
Computational				✓	✓	✓	
Adversary model							
Static private key access	✓	✓	✓	✓	✓	✓	✓
Static private key modification			✓		✓		✓
Ephemeral private key access	✓	✓	✓	✓	✓	✓	✓
Ephemeral key modification			✓				✓
Pre-shared key access	✓	✓	✓	✓	✓	✓	✓
Pre-shared key modification			✓		✓		✓
Key distribution compromise			✓		✓		✓
Pre-computation access							✓
Pre-computation modification							✓
Proofs							
Manual				✓	✓	✓	
CRYPTOVERIF					✓		
PROVERIF		✓					✓
TAMARIN	✓		✓			✓	✓
SAPIC$^+$							✓

TABLE I: Adversary Models and Proofs Techniques.

shared keys compromise. Symbolic model from [30], adapted to IKpsk2, is the most precise as it captures key corruption through leakage and modification. We enrich this model with new adversarial capabilities related to pre-computation.

V. OUR SYMBOLIC MODEL

We propose a more detailed formal model of WireGuard in the applied Π-Calculus, the language used by the automatic cryptographic protocol verifier SAPIC$^+$ [15].

A. Adversary Model

We adopt the methodology from [5] with some adaptations to our protocol. An adversary model, as defined in [5], is a combination of adversarial compromise. To outline our model, we consider four dimensions of adversarial compromise: which kind of data is compromised, whose data it is, when the compromise occurs and which type of compromise it is. All combinations of capabilities have been considered in our analysis. Some irrelevant combinations were discarded (See Section VI).

The adversary initially knows the name of all agents belonging to the set *agent* and their corresponding long term public static keys. Since WireGuard is a 2-party protocol, we distinguish, without loss of generality, two types of agents: an *Initiator I* and a *Responder R*. The set *agent* is reduced to $\{I, R\}$ thereof. Let *ltkX* and *ekX* be respectively the long term private key and the ephemeral private key of agent $X \in$ *agent*. Let *psk* be the pre-shared key between I and R and let *pre* be the pre-computation described in Section IV-C. We define the set *data* as the set of data subject to a compromise in our model, that is, long term keys, ephemeral keys, pre-shared key and pre-computation. Thus, let $d \in$ *data*. We consider three kind of d compromise in our model. R_d refers to the case when d is generated or computed honestly yet it is revealed to the adversary. M_d refers to when d is generated or modified and D_d refers to a dishonest distribution of d when d is a long term public key. Compromises can occur any time

during protocol execution. For example, R_{ltkX} represents the private long term key reveal of the agent X. This compromise may appear too strong but the intuition is that a protocol may still function as long as the long term key of the other partner is not revealed. The same argument applies to all other data compromise. Compromise can also occur at the end of the protocol. For this type of compromise, we only consider a reveal-type compromise and we refer to it as R_d^* for relevant data d. Let $\mathcal{A}$ be the set of all atomic adversarial capabilities. Our *adversary model* involves the largest set of atomic compromise for all data in *data* combined with the Dolev-Yao's adversarial capabilities [20].

B. Our Methodology

We evaluate security properties with regard to adversarial compromise. Our final results are of the form *property is guaranteed unless the adversary compromises X_1 or X_2 or ...* where *property* is a security property and X_i is a set of data included in *data*. We search for necessary and sufficient conditions of adversarial compromise under which each security property is violated.

We begin with searching for necessary conditions of compromise under which the property is no longer verified. To motivate our formulation, let us first consider agreement properties. Agreement or authentication properties can be captured using correspondence assertions in SAPIC$^+$ and have the following structure: "on every execution trace of the protocol, the event e_1 is preceded by the event e_2". Let $\mathcal{P}_a$ be an agreement property which can be expressed as $e_1 \implies e_2$ and let $\mathcal{C} \in \mathcal{A}$. If $e_1 \implies e_2$ is false and $e_1 \implies (e_2 \text{ or } \mathcal{C})$ is true considering our adversary model, this means that on every execution trace of the protocol either $\mathcal{P}_a$ is verified or the adversary has the atomic capability $\mathcal{C}$ that breaks this property, that is $\mathcal{C}$ is a necessary condition for which the property $\mathcal{P}_a$ is violated. We test all possible combinations of adversarial compromise present in our adversary model and we compute sufficient combinations among all the necessary combinations for which the property is not true. For example, if we suppose that our adversary model is the combination of R_{ltkI}, R_{psk} and the standard Dolev-Yao's adversarial capabilities and if we obtain the following results for the property $\mathcal{P}_a$, then the second result implies the fourth one, in:

- $e_1 \implies e_2$ is false
- $e_1 \implies (e_2 \text{ or } R_{psk})$ is true
- $e_1 \implies (e_2 \text{ or } R_{ltkI})$ is false
- $e_1 \implies (e_2 \text{ or } R_{psk} \text{ or } R_{ltkI})$ is true.

In this adversary model, we conclude that *psk* reveal is a necessary and sufficient condition for which the property $\mathcal{P}_a$ is violated and we state that property $\mathcal{P}_a$ is guaranteed unless *psk* is revealed to the adversary.

VI. ANALYSIS OF WIREGUARD WITH SAPIC$^+$

We consider 34 security properties; 4 agreement properties: agreement of RecHello message (from Responder to Initiator), agreement of first TransData message (from Initiator to Responder), agreement of next TransData messages (from

Initiator to Responder and from Responder to Initiator); 12 secrecy properties: secrecy and PFS of session key before derivation (named k_6 in protocol description), from Initiator's and Responder's view, secrecy and PFS of derived keys (named C^i and C^r), from Initiator's and Responder's view; anonymity, for WireGuard with or without cookies.

For each property, for each protocol version (with or without cookies), our adversary model implies up to $2^{21} = \textbf{2097152}$ cases of key compromises, as adversary can:

- Read Initiator (resp. Responder) static private key u (resp. v), Initiator (resp. Responder) ephemeral private key x (resp. y), pre-shared key psk, ecdh pre-computation before protocol execution (2^6 cases).
- Read Initiator (resp. Responder) static private key u (resp. v), Initiator (resp. Responder) ephemeral private key x (resp. y), pre-shared key psk, ecdh pre-computation after protocol execution (2^6 cases).
- Modify Initiator (resp. Responder) static private key u (resp. v), Initiator (resp. Responder) ephemeral private key x (resp. y), pre-shared key psk, ecdh pre-computation for Initiator or Responder (2^7 cases).
- Modify Initiator's static public key U (resp. Responder static public key V) distribution (2^2 cases).

We describe here our methodology, depicted in Fig.7, for WireGuard without cookie, the same applies for version of WireGuard with cookies. Our main idea is to capture key modification and key distribution modification through different *models* and key reveal through *queries*. This allows to drastically decrease the number of resolutions. Then we proceed the same way as in [34]: we use SAPIC$^+$ to generate all PROVERIF files, which model all cases, we keep all queries that are satisfied (✓) and combine them (this combination is explained below, it is based on Conjunctive Normal Form and Disjunctive Normal Form computation). From this combination, we deduce a single *lemma* that we finally assess with TAMARIN, in one file generated with SAPIC$^+$.
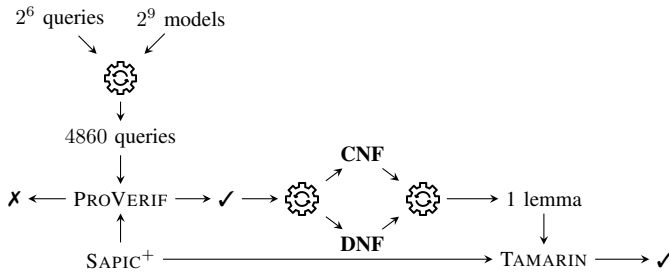

Fig. 7: Methodology.

We consider a first set of adversarial capabilities: key reveal before protocol execution (2^6 cases), key modifications (2^7 cases) and key distribution modifications (2^2 cases). This set is used to assess all agreement and secrecy properties.

Then we consider a second set: key reveal after protocol execution for static keys and pre-shared key (with key reveal before protocol for other keys), key distribution modifications (2^2 cases). This allows to capture precisely Perfect Forward

Secrecy (PFS) and also current WireGuard implementation, where a single configuration file contains both private static key and pre-shared key.

Finally, we consider anonymity for key reveal, which we do not combine with key modification nor key distribution modification.

A. Agreement, Secrecy and Perfect Forward Secrecy

We model agreement as trace property for RecHello message, first TransData message from Initiator to Responder and next TransData messages which can be either from Initiator to Responder or from Responder to Initiator. We model the strongest notion: if a message has been received by a peer, then it must have been sent by the other peer. With notations from Section III-A, we model key secrecy as trace property for keys k_6, C^i, C^r, from Initiator's and Responder's views.

For these properties, we start with a *reference* model, of which an extract is depicted in Fig.8 (up left). In this reference model, keys generated in the main process are: Initiator and Responder static private keys, (˜ltkI, ˜ltkR) and pre-shared key (˜psk). Keys are passed as arguments to two sub processes, Initiator and Responder. Process Initiator has arguments ˜ltkI (its own static private key), pkI = 'g'^˜ltkI (its own static public key), pkR = 'g'^˜ltkR (Responder static public key), ˜psk (pre-shared key), empty (a public term used in WireGuard RecHello message), 'zero_1' (a public term used to model counter used in TransData messages). Responder has similar arguments. Initiator generates its ephemeral private key (˜ekI), Responder also (˜ekR). Five processes model key compromise: RevealPsk for pre-shared key, RevealLtki and RevealLtkr for static keys and RevealPre for pre-computation, RevealEki and RevealEkr for ephemeral keys. Initiator and Responder processes are called in parallel with these compromise processes and replicated. Inside Initiator and Responder processes, computation is parallel to compromise of ephemeral private key. Fig.8 (up right) describes how a model is derived from reference model on a sample: Adversary can modify pre-shared key psk, Initiator and Responder ephemeral keys x and y. In derived model, instructions new ˜psk, new ˜ekI and new ˜ekR are replaced by in(˜psk), in(˜ekI) and in(˜ekR), respectively. Furthermore, as adversary sets these values, assessing their access is not necessary, hence instructions (RevealPsk(˜psk)), (RevealEki(˜ekI)) and (RevealEkr(˜ekR)) are removed. This allows to define 2^9 models. Fig.8 (bottom) describes how queries are derived: key combinations are added in the query as disjunctions. This allows to define up to 2^6 queries. Finally, for each derived model, we keep only the necessary queries, involving keys that adversary *does not modify*. For each agreement and secrecy properties, we obtain a set of **4860** queries to evaluate.

For PFS properties, the methodology is the same. In addition, to capture temporal key compromise, we use the notion of **phase** in the generated PROVERIF

```
──────── Reference process ────────          ──────── Adversary sets x and y and psk ────────
... new ˜ltkI;new ˜ltkR;new ˜psk;new empty;    ... new ˜ltkI; new ˜ltkR;in(˜psk);new empty;
out(<'initiator','g'^˜ltkI>); out(<'responder','g'^˜ltkR>);   out(<'initiator','g'^˜ltkI>); out(<'responder','g'^˜ltkR>);

( (! Initiator(˜ltkI, 'g'^˜ltkI, 'g'^˜ltkR, ˜psk, ...))   ( (! Initiator(˜ltkI, 'g'^˜ltkI, 'g'^˜ltkR, ˜psk, ...))
  | (! Responder(˜ltkR, 'g'^˜ltkI, 'g'^˜ltkR, ˜psk, ...))    | (! Responder(˜ltkR, 'g'^˜ltkI, 'g'^˜ltkR, ˜psk, ...))
  | RevealPsk(˜psk)                                          // | RevealPsk(˜psk)
  | RevealLtki(˜ltkI)                                        | RevealLtki(˜ltkI)
  | RevealLtkr(˜ltkR) | RevealPre(˜ltkI, ˜ltkR) )            | RevealLtkr(˜ltkR) | RevealPre(˜ltkI, ˜ltkR) )

let Initiator(˜ltkI, pkI, pkR, ˜psk, ...) =        let Initiator(˜ltkI, pkI, pkR, ˜psk, ...) =
    ... new ˜ekI; ... let pekI = 'g'^˜ekI in           ... in(˜ekI); ... let pekI  = 'g'^˜ekI in
    (...) | (RevealEki(˜ekI))                          (...) // | (RevealEki(˜ekI))

let Responder(˜ltkR, pkI, pkR, ˜psk, ...) =        let Responder(˜ltkR, pkI, pkR, ˜psk, ...) =
    ... new ˜ekR; ... let pekR = 'g'^˜ekR in           ... in(˜ekR); ... let pekR  = 'g'^˜ekR in
    (...) | (RevealEkr(˜ekR)) ...                      (...) // | (RevealEkr(˜ekR)) ...
```
```
query i:time,j:time, pki:bitstring, pkr:bitstring, peki:bitstring, pekr:bitstring,  psk:bitstring, ck:bitstring;
(event(eRConfirm(pki, pkr, peki, pekr, psk, ck))@i) ==> (((event(eIConfirm(pki, pkr, peki, pekr, psk, ck))@j) && (j < i)))
...
query i:time,j:time, pki:bitstring, pkr:bitstring, peki:bitstring, pekr:bitstring, psk:bitstring, ck:bitstring,
j1:time, j2:time, j3:time, j4:time, j5:time, j6:time;
(event(eRConfirm(pki, pkr, peki, pekr, psk, ck))@i) ==> (((event(eIConfirm(pki, pkr, peki, pekr, psk, ck))@j) && (j < i))
||((event(eRevPri(pki))@j1) && (j1 < i)) || ((event(eRevPri(pkr))@j2) && (j2 < i))
||((event(eRevPri(peki))@j3) && (j3 < i)) || ((event(eRevPri(pekr))@j4) && (j4 < i))
||((event(eRevPre(pki, pkr))@j5) && (j5 < i)) || ((event(eRevPsk(psk))@j6) && (j6 < i))).
```
──────── Queries ────────

Fig. 8: SAPIC$^+$ Processes and Queries, where blue bold denotes differences between reference process and modified process.

files: (RevealPsk(˜psk)) is replaced with (phase 1: RevealPsk(˜psk)). We add the same modification for RevealLtki and RevealLtkr. In PROVERIF, **phase 0** denotes protocol execution and **phase 1** allows to set key compromise after protocol execution. For each PFS properties, we obtain a set of **64** queries to evaluate.

Once all queries are assessed, we obtain a set of results that we need to combine. We process them in a manner similar to the one used in [30]. In this aim, we use symbols to model key compromise:

- R_x (resp. R_y) Initiator's (resp. Responder's) ephemeral key is revealed, M_x (resp. M_y) Initiator's (resp. Responder's) ephemeral key is modified.
- R_u (resp. R_v) Initiator's (resp. Responder's) static key is revealed, M_u (resp. M_v Initiator's (resp. Responder's) static key is modified. R_u^* (resp. R_v^*) Initiator's (resp. Responder's) static key is revealed *after* protocol execution.
- R_s pre-shared key is revealed, M_s, modified. R_s^* pre-shared key is revealed *after* protocol execution.
- R_c pre-computation is revealed, M_i (resp. M_r) Initiator's (resp. Responder's) pre-computation is modified.
- D_u (resp. D_v) Initiator's (resp. Responder's) static key is compromised during initial distribution.

With these symbols, the set of all 2^9 models corresponds to the conjunction: $\bigwedge_{\alpha \in \{u,v,x,y,s,i,r\}} M_\alpha \bigwedge_{\beta \in \{u,v\}} D_\beta$. Similarly, the set of all 2^6 queries corresponds to the disjunction: $\bigvee_{\gamma \in \{u,v,x,y,c,s\}} R_\gamma$. Similarly, for PFS, a smaller number of models and queries is involved: $\bigwedge_{\beta \in \{u,v\}} D_\beta$ and $\bigvee_{\gamma \in \{u,v,s\}} R_\gamma^*$.

Note that this set of compromises is similar to the one in [30], with two exceptions: we model pre-computation compromise and [30] gathers D_u and D_v in a single compromise named D_{pki} (which models "key distribution is compromised"). After assessment with PROVERIF, we consider all

queries which are proven **true** in the model. To illustrate, consider the model $M_s \wedge M_x$, where adversary can set pre-shared key and Initiator private ephemeral key. For agreement of third message, in this model, there are 5 disjunctions that correspond to **true** queries (among 16): $R_u \vee R_v \vee R_y$, $R_u \vee R_v \vee R_c$, $R_u \vee R_y \vee R_c$, $R_u \vee R_v \vee R_y \vee R_c$, $R_u \vee R_y$. We then compute the conjunction $M_s \wedge M_x \wedge (R_u \vee R_v \vee R_y) \wedge (R_u \vee R_v \vee R_c) \wedge (R_u \vee R_y \vee R_c) \wedge (R_u \vee R_v \vee R_y \vee R_c) \wedge (R_u \vee R_y)$ and we reduce it to its Conjunctive Normal Form (**CNF**), equal to $M_s \wedge M_x \wedge (R_u \vee R_v) \wedge (R_u \vee R_v \vee R_c)$ and to its Disjunctive Normal Form (**DNF**), equal to $(M_s \wedge M_x \wedge R_u) \vee (M_s \wedge M_x \wedge R_c \wedge R_y) \vee (M_s \wedge M_x \wedge R_v \wedge R_y)$.

We repeat this process for all models and once all models are evaluated, we obtain a set of 2^9 CNFs and DNFs: each DNF gives an interpretation at model level and all CNFs considered together provide a result for the evaluated security property. To this aim, we compute the disjunction of all CNFs and the DNF of the result, which gives us an interpretable result, at property level. This final result is comparable with results from previous analyses. Obtained DNFs, labeled **DNFx**, are detailed in Table II. We also compute a simplified version of these DNFs as some compromises are implied by others. These simplified DNF are labeled **DNFx⋆**. At last step, we translate **DNFx⋆** to a *lemma* that we assess with TAMARIN. Note that **DNFx⋆** show security properties are all breakable with read access to cryptographic keys, as key modification is not involved. Results for versions of WireGuard with or without cookies are the same.

Results. DNF for agreement of RecHello and transport message from Responder to Initiator are equal (**DNF1** and simplified version **DNF1⋆**) and DNF for agreement of first and second TransData message, from Initiator to Responder, are equal (**DNF2** and simplified version **DNF2⋆**). For secrecy, DNFs contains key compromises for two phases, as
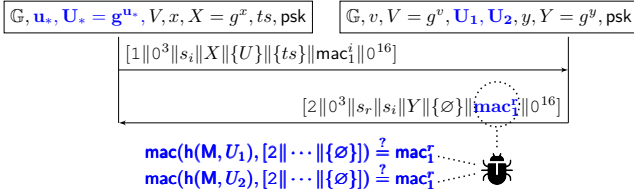
Fig. 9: Attack against Anonymity, where blue bold denotes attacker computation: attacker captures mac_1 field from `RecHello` message, then compares with two possible values, computed with public keys $\mathbf{U_1}$ and $\mathbf{U_2}$.

we model static keys (u and v) and pre-shared key leakage first during protocol execution, then after protocol execution, to capture perfect forward secrecy. To distinguish these two compromises, the first refers to R_u, R_v and R_s while the later refers to R_u^*, R_v^* and R_s^*: DNF for secrecy of k_6, C^i, C^r from Initiator's view are all equal (**DNF3** and simplified version **DNF3***); DNF for secrecy of k_6, C^i, C^r from Responder's view are all equal (**DNF4** and simplified version **DNF4***).

Our result show that for **DNF1**, ephemeral key y (from Responder) does not appear, whereas ephemeral key x (from Initiator) appears. This is due to our model as we only considers active adversary: compromise $(R_s \wedge R_v \wedge R_y)$ also breaks the property, but for DNF computation, $(R_s \wedge R_v \wedge R_y) \vee (R_s \wedge R_v)$ simplifies to $(R_s \wedge R_v)$. Hence if adversary is active and has compromised Responder's static key v and pre-shared key, then adversary can generate its own ephemeral key and send messages to Initiator, while if adversary is passive, adversary needs to compromise all keys from Responder to break security property. The same analysis holds for **DNF2**.

B. Anonymity

We model anonymity with observational equivalence in the following context: two Initiators, of public keys U_1 and U_2, can establish a WireGuard session with a common Responder, of public key V. The property is satisfied if an adversary, which has access to these public keys and to exchanged messages, cannot assess which Initiator has established a session. We found that, as opposed to initial claims of original specification [21] and to symbolic analysis with TAMARIN [23], and in accordance with computational analysis with CRYPTOVERIF [42], this property is not satisfied: Fig.9 depicts an attack against anonymity, identified on our model with SAPIC$^+$ prover. An Initiator, of public key $U_* \in (U_1, U_2)$, establishes a session with Responder. They exchange a `RecHello` message whose 7th field equals $\mathsf{mac}(\mathsf{h}(\mathsf{M}, U_*), [2 \parallel \cdots \parallel \{\varnothing\}])$, where $[2 \parallel \cdots \parallel \{\varnothing\}]$ are the first 6 fields of `RecHello` message and M is a public constant. Adversary, which knows U_1 and U_2, can then compute $\mathsf{mac}(\mathsf{h}(\mathsf{M}, U_1), [2 \parallel \cdots \parallel \{\varnothing\}])$ and $\mathsf{mac}(\mathsf{h}(\mathsf{M}, U_2), [2 \parallel \cdots \parallel \{\varnothing\}])$ and assess which public key U_1 or U_2 has been used in message authentication code by comparison with transmitted value $\mathsf{mac}(\mathsf{h}(\mathsf{M}, U_*), [2 \parallel \cdots \parallel \{\varnothing\}])$. Finally, adversary can distinguish between the two Initiators.

Anonymity in previous analyses. Analysis from [23] proposes a proof with TAMARIN prover of a property named *identity hiding*, modeled as a trace property. As noticed in Section IV-B, it does not include compliant message authentication codes in `InitHello` and `RecHello` messages. Result from this analysis can be adapted with symbols from Section VI-A: identity hiding has DNF $R_u \vee R_v \vee R_x$. Similarly, analysis from [30] models and proves anonymity for `IKpsk2` with observational equivalence with TAMARIN prover. We can adapt its results with symbols from Section VI-A: anonymity for Initiator has DNF $R_v \vee R_u$ and anonymity for Responder has DNF $R_v \vee R_u$. Finally, analysis from [42] describes the same attack as the one we identified and propose a proof of identity hiding property with CRYPTOVERIF prover of the protocol, without message authentication codes in `InitHello` and `RecHello` messages. The property is however different as ours it is modeled as a secrecy property. We can adapt its result as before: this property has DNF R_y. These three results cannot directly apply to WireGuard as all do not consider message authentication codes in the two first messages, which is exactly the reasons why adversary can break anonymity.

Proposed fixes. We propose original fixes that ensure anonymity and does not change messages content. For this, we remark that message authentication codes in `InitHello` and `RecHello` messages only involve as keys data potentially known by an adversary: public value M and public keys of Initiator and Responder, hence allowing to attack anonymity, as these authentication codes leak used public key. To counter this attack, we propose a modification of message authentication code computation to ensure the key for message authentication code is only known by Initiator and Responder. For this, we propose to use as key for message authentication code in `RecHello` message either the value $\mathsf{h}(U \parallel g^{uv})$ or the value $\mathsf{h}(U \parallel \mathsf{psk})$ (instead of the value $\mathsf{h}(\mathsf{M} \parallel U)$ currently used). We modeled the modified protocol and anonymity with observational equivalence in the same context as before. Our modified protocol reaches anonymity and we computed the associated DNF. For message authentication key $\mathsf{h}(U \parallel g^{uv})$, DNF is $R_u \vee R_v \vee R_x \vee R_c$; for message authentication key $\mathsf{h}(U \parallel \mathsf{psk})$, DNF is $R_v \vee R_s \vee R_x$. We finally remark that we could complete results from [30]: removing mac computations from `InitHello` and `RecHello` messages leads to a key exchange close to `IKpsk2`. We also verified that this other protocol modification reaches anonymity, with DNF $R_u \vee R_v \vee R_x \vee R_c$.

C. Performances

We evaluated our models on a dedicated server, equipped with 256 cores of CPU 1.5 GHz, on which we ran in parallel all PROVERIF queries. Agreement, secrecy and PFS are verified with PROVERIF in around 15 minutes (for each property) and anonymity is verified with PROVERIF in around 9 hours for fix based on g^{uv} and 2 hours for fix based on psk. We tested our result for trace properties (agreement, secrecy, PFS) with a lemma in TAMARIN for a full version of the protocol and TAMARIN confirmed properties are satisfied in around 5 hours.

Results	**Properties**: `RecHello` agreement, Next `TransData` (R to I).
[35]	$(R_s \wedge R_v)$ (for `IKpsk2` with PROVERIF)
[23]	$(R_s \wedge R_v) \vee (R_s \wedge R_u \wedge R_x)$ (for WireGuard with TAMARIN)
[30]	$(D_v \wedge R_s) \vee (M_s \wedge R_v) \vee (M_v \wedge R_s) \vee (R_s \wedge R_v) \vee (R_s \wedge R_u \wedge R_x)$ (for `IKpsk2` with TAMARIN)
DNF1	$(D_v \wedge M_s) \vee (D_v \wedge R_s) \vee (M_s \wedge M_v) \vee (M_s \wedge R_v) \vee (M_v \wedge R_s) \vee (R_s \wedge R_v) \vee (M_i \wedge M_s \wedge M_x) \vee (M_i \wedge M_s \wedge R_x)$ $\vee (M_i \wedge M_x \wedge R_s) \vee (M_i \wedge R_s \wedge R_x) \vee (M_r \wedge M_s \wedge M_x) \vee (M_r \wedge M_s \wedge R_x) \vee (M_r \wedge M_x \wedge R_s) \vee (M_r \wedge R_s \wedge R_x)$ $\vee (M_s \wedge M_u \wedge M_x) \vee (M_s \wedge M_u \wedge R_x) \vee (M_s \wedge M_x \wedge R_c) \vee (M_s \wedge M_x \wedge R_u) \vee (M_s \wedge R_c \wedge R_x) \vee (M_s \wedge R_u \wedge R_x)$ $\vee (M_u \wedge M_x \wedge R_s) \vee (M_u \wedge R_s \wedge R_x) \vee (M_x \wedge R_c \wedge R_s) \vee (M_x \wedge R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x)$
DNF1⋆	$(D_v \wedge R_s) \vee (R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x)$
Results	**Properties**: First `TransData` agreement, Next `TransData` (I to R).
[35]	$(R_s \wedge R_u)$ (for `IKpsk2` with PROVERIF)
[23]	$(R_s \wedge R_u) \vee (R_s \wedge R_v \wedge R_y)$ (for WireGuard with TAMARIN)
[30]	$(M_s \wedge R_u) \vee (M_u \wedge R_s) \vee (R_s \wedge R_u) \vee (R_s \wedge R_v \wedge R_y)$ (for `IKpsk2` with TAMARIN)
DNF2	$(D_u \wedge M_s) \vee (D_u \wedge R_s) \vee (M_s \wedge M_u) \vee (M_s \wedge R_u) \vee (M_u \wedge R_s) \vee (R_s \wedge R_u) \vee (M_i \wedge M_s \wedge M_y) \vee (M_i \wedge M_s \wedge R_y)$ $\vee (M_i \wedge M_y \wedge R_s) \vee (M_i \wedge R_s \wedge R_y) \vee (M_r \wedge M_s \wedge M_y) \vee (M_r \wedge M_s \wedge R_y) \vee (M_r \wedge M_y \wedge R_s) \vee (M_r \wedge R_s \wedge R_y)$ $\vee (M_s \wedge M_v \wedge M_y) \vee (M_s \wedge M_v \wedge R_y) \vee (M_s \wedge M_y \wedge R_c) \vee (M_s \wedge M_y \wedge R_v) \vee (M_s \wedge R_c \wedge R_y) \vee (M_s \wedge R_v \wedge R_y)$ $\vee (M_v \wedge M_y \wedge R_s) \vee (M_v \wedge R_s \wedge R_y) \vee (M_y \wedge R_c \wedge R_s) \vee (M_y \wedge R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y)$
DNF2⋆	$(D_u \wedge R_s) \vee (R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y)$
Results	**Properties**: Secrecy of k_6, C^i, C^r from Initiator's view, including PFS.
DNF3	$(D_v \wedge M_s) \vee (D_v \wedge R_s) \vee (M_s \wedge M_v) \vee (M_s \wedge R_v) \vee (M_v \wedge R_s) \vee (R_s \wedge R_v) \vee (M_i \wedge M_s \wedge M_x) \vee (M_i \wedge M_s \wedge R_x)$ $\vee (M_i \wedge M_x \wedge R_s) \vee (M_i \wedge R_s \wedge R_x) \vee (M_r \wedge M_s \wedge M_x) \vee (M_r \wedge M_s \wedge R_x) \vee (M_r \wedge M_x \wedge R_s) \vee (M_r \wedge R_s \wedge R_x)$ $\vee (M_s \wedge M_u \wedge M_x) \vee (M_s \wedge M_u \wedge R_x) \vee (M_s \wedge M_x \wedge R_c) \vee (M_s \wedge M_x \wedge R_u) \vee (M_s \wedge R_c \wedge R_x) \vee (M_s \wedge R_u \wedge R_x)$ $\vee (M_u \wedge M_x \wedge R_s) \vee (M_u \wedge R_s \wedge R_x) \vee (M_x \wedge R_c \wedge R_s) \vee (M_x \wedge R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x)$ $\vee (R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$
DNF3⋆	$(D_v \wedge R_s) \vee (R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x) \vee (R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$
Results	**Properties**: Secrecy of k_6, C^i, C^r from Responder's view, including PFS.
DNF4	$(D_u \wedge M_s) \vee (D_u \wedge R_s) \vee (M_s \wedge M_u) \vee (M_s \wedge R_u) \vee (M_u \wedge R_s) \vee (R_s \wedge R_u) \vee (M_i \wedge M_s \wedge M_y) \vee (M_i \wedge M_s \wedge R_y)$ $\vee (M_i \wedge M_y \wedge R_s) \vee (M_i \wedge R_s \wedge R_y) \vee (M_r \wedge M_s \wedge M_y) \vee (M_r \wedge M_s \wedge R_y) \vee (M_r \wedge M_y \wedge R_s) \vee (M_r \wedge R_s \wedge R_y)$ $\vee (M_s \wedge M_v \wedge M_y) \vee (M_s \wedge M_v \wedge R_y) \vee (M_s \wedge M_y \wedge R_c) \vee (M_s \wedge M_y \wedge R_v) \vee (M_s \wedge R_c \wedge R_y) \vee (M_s \wedge R_v \wedge R_y)$ $\vee (M_v \wedge M_y \wedge R_s) \vee (M_v \wedge R_s \wedge R_y) \vee (M_y \wedge R_c \wedge R_s) \vee (M_y \wedge R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y)$ $\vee (R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$
DNF4⋆	$(D_u \wedge R_s) \vee (R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y) \vee (R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$

TABLE II: Computed DNFs for WireGuard and comparisons with results from [35], [23] and [30] for agreement properties (secrecy properties are not directly comparable and anonymity is not reached for WireGuard).

Regarding anonymity, it is important to note that SAPIC⁺ does not currently provide support for the translation of equivalence properties into TAMARIN.

D. Comparison with Previous Analyses

We compare our results with symbolic results on Wire-Guard [23] and on `IKpsk2` ([35] and [30]). In addition to ecdh pre-computation, we give here other insights.

Comparison with [23]. This analysis only assesses one specific case, for which the evaluated property is satisfied. Adapting its results with our notations, these are: agreement on `RecHello` holds unless $(R_s \wedge (R_v \vee (R_u \wedge R_x)))$, agreement on first `TransData` message holds unless $(R_s \wedge (R_u \vee (R_v \wedge R_y)))$, secrecy holds unless $(R_s \wedge ((R_u \wedge R_x) \vee (R_v \wedge R_y)))$, weak PFS holds unless $(R_s^* \wedge ((R_u^* \wedge R_x) \vee (R_v^* \wedge R_y)))$. For secrecy, the modeled property is different to ours as it is conditioned on agreement: if Initiator and Responder agrees upon a key, then this key shall be secret, while we model secrecy of key on both Initiator and Responder's view. We compute DNFs for agreement results of [23], and compare them in Table II: our results extend them. For secrecy, results

are not directly comparable as modeled properties are different, however we note that $(R_s^* \wedge ((R_u^* \wedge R_x) \vee (R_v^* \wedge R_y)))$ has DNF $(R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y)$, which is a combination we capture in our results.

Comparison with [35]. This analysis also assesses one specific key compromise scenario, for `IKpsk2`, for which the evaluated property is satisfied. Adapting these results with our notations, these are: agreement on second message holds unless $(R_s \wedge R_v)$, agreement on transport message from Initiator to Responder holds unless $(R_s \wedge R_u)$, agreement on transport message from Responder to Initiator holds unless $(R_s \wedge R_v)$. For secrecy, the modeled property is different than ours as it concerns PFS of payloads (and not secrecy of keys). With our notations, obtained results are: PFS of payload of second message and of transport message from Responder to Initiator holds unless $(R_s^* \wedge R_u^*)$, PFS of payload of transport message from Initiator to Responder holds unless $(R_s^* \wedge R_v^*)$. We confirm these results related to agreement and secrecy,

Comparison with [30]. As explained in Section IV-A, this analysis assesses all possible key compromises and results are already DNFs for each property. Adapting these results

with our notations, these can be summarized as follows (note that [30] does not distinguish which key is compromised during distribution, but refers to one global compromise termed D_{pki}, furthermore, *active* refers to an active adversary):

- Agreement on second message and on transport message from Responder to Initiator hold unless $(active \wedge R_y \wedge R_v \wedge R_s) \vee (active \wedge R_x \wedge R_u \wedge R_s) \vee (M_y \wedge R_v \wedge R_s) \vee (M_v \wedge D_{pki} \wedge R_y \wedge R_s) \vee (M_y \wedge M_v \wedge D_{pki} \wedge R_s)$.
- Agreement on first `TransData` message from Initiator to Responder holds unless $(active \wedge R_x \wedge R_u \wedge R_{psk}) \vee (active \wedge R_y \wedge R_v \wedge R_{psk}) \vee (D_x \wedge R_u \wedge R_{psk}) \vee (D_u \wedge R_x \wedge R_{psk}) \vee (D_x \wedge D_u \wedge R_{psk})$.

Hence, agreement of second message and transport message from Responder to Initiator relies on conjunction $(M_y \wedge M_v \wedge D_{pki} \wedge R_s^c)$, which is captured in our results through a simpler conjunction $(D_v \wedge R_s)$. Term D_{pki} however does not appear in DNF of agreement of first transport message, while conjunction $(D_u \wedge R_s)$ appears in our results. Furthermore, our model only considers active adversary, hence simplifications occur in DNFs: e.g for **DNF1**, $(M_s \wedge R_v) \vee (M_v \wedge R_s) \vee (R_s \wedge R_v)$ corresponds to $(active \wedge R_y^c \wedge R_v^c \wedge R_s^c)$ in [30] (same approach for **DNF2**). For secrecy, as for [35], the modeled property is different than ours as it concern secrecy and PFS of payloads (and not secrecy of keys), [30] obtains:

- Secrecy and PFS of payload of second message and of transport messages from Initiator's view holds unless $(R_y \wedge R_v^* \wedge R_s^*) \vee (R_x \wedge R_u^* \wedge R_s^*) \vee (M_y \wedge R_y \wedge R_s) \vee (M_v \wedge D_{pki} \wedge R_y \wedge R_s) \vee (M_y \wedge M_v \wedge D_{pki} \wedge R_s)$.
- Secrecy and PFS of payload of second message and of transport messages from Responder's view holds unless $(R_x \wedge R_u^* \wedge R_s^*) \vee (R_y \wedge R_v^* \wedge R_s^*) \vee (D_x \wedge R_u \wedge R_s) \vee (D_u \wedge R_x \wedge R_s) \vee (D_x \wedge D_u \wedge R_s)$.

The difference for secrecy is the same as for agreement: simplification due to DNFs computation and refinement on key distribution compromise.

E. Results Interpretation

Our results show message agreement and key secrecy are reached whereas anonymity cannot currently be reached. Below we propose recommendations, one is structural as it implies a protocol modification, the two others are aimed at WireGuard users, in particular against active adversaries.

Importance of ecdh **pre-computations**. We include ecdh pre-computation in our analysis as current implementations propose it as optimization when `InitHello` messages are received: we estimate it gives a new attack path for an adversary, e.g when Yubikeys are used to protect static keys. Now **DNF1*** in Table II contains combinations $(R_c \wedge R_s \wedge R_x) \vee (R_u \wedge R_s \wedge R_x)$ and **DNF2*** contains combinations $(R_c \wedge R_s \wedge R_y) \vee (R_v \wedge R_s \wedge R_y)$: these mean that an adversary with access to ecdh pre-computation has the same power than an adversary with access to static private key. As explained in Section IV-C, this contradicts ecdh property as adversary with access to pre-computation shall not be so powerful. We therefore recommend to remove this implementation optimization and to compute ecdh at `InitHello` reception.

Anonymity We proved that including g^{uv} or psk in key for message authentication code guarantees this property is reached, hence we recommend to update computation in this direction. Note that this implies that at `InitHello` reception, order of operation shall be adapted : currently at `InitHello` reception, authentication code is checked, then $\{U\}$ is decrypted, then U is checked. With our proposed fix based on g^{uv}, order of operations at `InitHello` reception shall be: $\{U\}$ is decrypted, then U is checked, then authentication code is checked.

Importance of pre-shared key psk. As we consider active adversary, **DNF1** and **DNF2** both contain combinations with two keys: $(D_v \wedge R_s) \vee (M_s \wedge R_v) \vee (M_v \wedge R_s) \vee (R_s \wedge R_v)$ for DNF1 and $(D_u \wedge R_s) \vee (M_s \wedge R_u) \vee (M_u \wedge R_s) \vee (R_s \wedge R_u)$ for DNF2. All these combinations involve psk which appears to have an important role against. However, for WireGuard, psk is optional, hence when psk $= \varnothing$, security against an active adversary only relies on static keys. We recommend that WireGuard users systematically use a pre-shared key.

Importance of initial key distribution. As opposed to WireGuard specification, we choose to model compromise of static public keys distribution, as we estimate this distribution is an attack path for an adversary. This leads to combinations $(D_v \wedge R_s)$ in **DNF1** and **DNF4** and $(D_u \wedge R_s)$ in **DNF2** and **DNF3**. This shows that the assumption in WireGuard specification about initial correct out of band key distribution is a key factor of WireGuard security and cannot be eluded: WireGuard is safe as soon as static keys are correctly distributed.

VII. CONCLUSION AND FUTURE WORK

We proposed a unified symbolic analysis of WireGuard and we give a precise model in SAPIC$^+$ which unifies of all previous analyses. We also improved the adversary model, with a more precise adversary that can read or set static, ephemeral or pre-shared keys, read or set ecdh pre-computations, control key distribution. This allows us to rediscover an attack on anonymity shown only in the computational model. We also demonstrated that ecdh pre-computations could lead to concrete attacks when static keys are protected in smart card and we propose original fixes that improves the security of WireGuard, as these two attacks are mitigated.

Our methodology allowed us to analyze agreement, secrecy, perfect forward secrecy and anonymity and to rapidly obtain results that are compact and comparable with other analyses of the same protocol because we compute DNFs for each security property. Currently our tool takes as input a model of WireGuard and queries are named in accordance with events positioned in WireGuard model, we think it could be adapted to other protocols.

We used computational analyses to compare modeled protocols and enrich our symbolic model, we did not compare their results with ours. Although security properties correspond to a close intuitive idea, their formal definitions differ hence they are not directly comparable. A future work could be to adapt our implementation in SAPIC$^+$ for a computational analysis.

REFERENCES

[1] M. Abadi, B. Blanchet, and C. Fournet, "The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication," *Journal of the ACM (JACM)*, vol. 65, no. 1, pp. 1 – 103, Oct. 2017. [Online]. Available: https://hal.inria.fr/hal-01636616

[2] J. Appelbaum, C. Martindale, and P. Wu, "Tiny WireGuard tweak," in *AFRICACRYPT 19: 11th International Conference on Cryptology in Africa*, ser. Lecture Notes in Computer Science, J. Buchmann, A. Nitaj, and T. eddine Rachidi, Eds., vol. 11627. Rabat, Morocco: Springer, Heidelberg, Germany, Jul. 9–11, 2019, pp. 3–20.

[3] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein, "BLAKE2: Simpler, smaller, fast as MD5," in *ACNS 13: 11th International Conference on Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science, M. J. Jacobson Jr., M. E. Locasto, P. Mohassel, and R. Safavi-Naini, Eds., vol. 7954. Banff, AB, Canada: Springer, Heidelberg, Germany, Jun. 25–28, 2013, pp. 119–135.

[4] M. Barbosa, G. Barthe, K. Bhargavan, B. Blanchet, C. Cremers, K. Liao, and B. Parno, "SoK: Computer-aided cryptography," in *2021 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 24–27, 2021, pp. 777–795.

[5] D. Basin and C. Cremers, "Know your enemy: Compromising adversaries in protocol analysis," *ACM Trans. Inf. Syst. Secur.*, vol. 17, no. 2, nov 2014. [Online]. Available: https://doi.org/10.1145/2658996

[6] D. J. Bernstein, "Curve25519: New Diffie-Hellman speed records," in *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, ser. Lecture Notes in Computer Science, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., vol. 3958. New York, NY, USA: Springer, Heidelberg, Germany, Apr. 24–26, 2006, pp. 207–228.

[7] 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*. San Jose, CA, USA: IEEE Computer Society Press, May 17–21, 2015, pp. 535–552.

[8] K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified models and reference implementations for the TLS 1.3 standard candidate," in *2017 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, May 22–26, 2017, pp. 483–502.

[9] 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*. Berkeley, CA, USA: IEEE Computer Society Press, May 18–21, 2014, pp. 98–113.

[10] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P.-Y. Strub, "Implementing TLS with verified cryptographic security," in *2013 IEEE Symposium on Security and Privacy*. Berkeley, CA, USA: IEEE Computer Society Press, May 19–22, 2013, pp. 445–459.

[11] K. Bhargavan and G. Leurent, "Transcript collision attacks: Breaking authentication in TLS, IKE and SSH," in *ISOC Network and Distributed System Security Symposium – NDSS 2016*. San Diego, CA, USA: The Internet Society, Feb. 21–24, 2016.

[12] B. Blanchet, "Modeling and verifying security protocols with the applied pi calculus and proverif," *Foundations and Trends® in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, 2016. [Online]. Available: http://dx.doi.org/10.1561/3300000004

[13] M. Brinkmann, C. Dresen, R. Merget, D. Poddebniak, J. Müller, J. Somorovsky, J. Schwenk, and S. Schinzel, "ALPACA: Application layer protocol confusion - analyzing and mitigating cracks in TLS authentication," in *USENIX Security 2021: 30th USENIX Security Symposium*, M. Bailey and R. Greenstadt, Eds. USENIX Association, Aug. 11–13, 2021, pp. 4293–4310.

[14] R. Canetti and H. Krawczyk, "Security analysis of IKE's signature-based key-exchange protocol," in *Advances in Cryptology – CRYPTO 2002*, ser. Lecture Notes in Computer Science, M. Yung, Ed., vol. 2442. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 18–22, 2002, pp. 143–161, https://eprint.iacr.org/2002/120/.

[15] V. Cheval, C. Jacomme, S. Kremer, and R. Künnemann, "Sapic+: protocol verifiers of the world, unite!" in *USENIX Security Symposium (USENIX Security), 2022*, 2022.

[16] V. Cortier and S. Kremer, "Formal Models and Techniques for Analyzing Security Protocols: A Tutorial," *Foundations and Trends in Programming Languages*, vol. 1, no. 3, p. 117, Sep. 2014. [Online]. Available: https://hal.inria.fr/hal-01090874

[17] C. Cremers, M. Horvat, S. Scott, and T. van der Merwe, "Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication," in *2016 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, May 22–26, 2016, pp. 470–485.

[18] C. J. F. Cremers, "Key exchange in IPsec revisited: Formal analysis of IKEv1 and IKEv2," in *ESORICS 2011: 16th European Symposium on Research in Computer Security*, ser. Lecture Notes in Computer Science, V. Atluri and C. Díaz, Eds., vol. 6879. Leuven, Belgium: Springer, Heidelberg, Germany, Sep. 12–14, 2011, pp. 315–334.

[19] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, J. Protzenko, A. Rastogi, N. Swamy, S. Zanella-Béguelin, K. Bhargavan, J. Pan, and J. K. Zinzindohoue, "Implementing and proving the TLS 1.3 record layer," in *2017 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, May 22–26, 2017, pp. 463–482.

[20] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[21] J. A. Donenfeld, "WireGuard: Next generation kernel network tunnel," in *ISOC Network and Distributed System Security Symposium – NDSS 2017*. San Diego, CA, USA: The Internet Society, Feb. 26 – Mar. 1, 2017.

[22] ——. (2021) Wireguard. https://www.wireguard.com.

[23] J. A. Donenfeld and K. Milner, "Formal verification of the wireguard protocol," https://www.wireguard.com/papers/wireguard-formal-verification.pdf, 2018.

[24] B. Dowling and K. G. Paterson, "A cryptographic analysis of the WireGuard protocol," in *ACNS 18: 16th International Conference on Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science, B. Preneel and F. Vercauteren, Eds., vol. 10892. Leuven, Belgium: Springer, Heidelberg, Germany, Jul. 2–4, 2018, pp. 3–21.

[25] B. Dowling, P. Rösler, and J. Schwenk, "Flexible authenticated and confidential channel establishment (fACCE): Analyzing the noise protocol framework," in *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part I*, ser. Lecture Notes in Computer Science, A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, Eds., vol. 12110. Edinburgh, UK: Springer, Heidelberg, Germany, May 4–7, 2020, pp. 341–373.

[26] D. Felsch, M. Grothe, J. Schwenk, A. Czubak, and M. Szymanek, "The dangers of key reuse: Practical attacks on IPsec IKE," in *USENIX Security 2018: 27th USENIX Security Symposium*, W. Enck and A. P. Felt, Eds. Baltimore, MD, USA: USENIX Association, Aug. 15–17, 2018, pp. 567–583.

[27] M. Fischlin, F. Günther, B. Schmidt, and B. Warinschi, "Key confirmation in key exchange: A formal treatment and implications for TLS 1.3," in *2016 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, May 22–26, 2016, pp. 452–469.

[28] S. Frankel and S. Krishnan, "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap," Internet Requests for Comments, RFC Editor, RFC 6071, February 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6071.txt

[29] S. Gajek, M. Manulis, O. Pereira, A.-R. Sadeghi, and J. Schwenk, "Universally composable security analysis of TLS," in *ProvSec 2008: 2nd International Conference on Provable Security*, ser. Lecture Notes in Computer Science, J. Baek, F. Bao, K. Chen, and X. Lai, Eds., vol. 5324. Shanghai, China: Springer, Heidelberg, Germany, Oct. 31 – Nov. 1, 2008, pp. 313–327.

[30] G. Girol, L. Hirschi, R. Sasse, D. Jackson, C. Cremers, and D. A. Basin, "A spectral analysis of noise: A comprehensive, automated, formal analysis of Diffie-Hellman protocols," in *USENIX Security 2020: 29th USENIX Security Symposium*, S. Capkun and F. Roesner, Eds. USENIX Association, Aug. 12–14, 2020, pp. 1857–1874.

[31] "Specifications for the keyed-hash message authentication code," National Institute of Standards and Technology (NIST), FIPS PUB 198, U.S. Department of Commerce, Mar. 2002.

[32] A. Hülsing, K.-C. Ning, P. Schwabe, F. Weber, and P. R. Zimmermann, "Post-quantum WireGuard," in *2021 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 24–27, 2021, pp. 304–321.

[33] O. Inc. (2021) Openvpn. https://openvpn.net/.

[34] C. Jacomme, E. Klein, S. Kremer, and M. Racouchot, "A comprehensive, formal and automated analysis of the EDHOC protocol," in *USENIX Security '23 - 32nd USENIX Security Symposium*, Anaheim, CA,

United States, Aug. 2023. [Online]. Available: https://inria.hal.science/hal-03810102

[35] N. Kobeissi, G. Nicolas, and K. Bhargavan, "Noise explorer: Fully automated modeling and verification for arbitrary noise protocols," in *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 2019, pp. 356–370. [Online]. Available: https://doi.org/10.1109/EuroSP.2019.00034

[36] H. Krawczyk, "SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols," in *Advances in Cryptology – CRYPTO 2003*, ser. Lecture Notes in Computer Science, D. Boneh, Ed., vol. 2729. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 17–21, 2003, pp. 400–425.

[37] ——, "Cryptographic extraction and key derivation: The HKDF scheme," in *Advances in Cryptology – CRYPTO 2010*, ser. Lecture Notes in Computer Science, T. Rabin, Ed., vol. 6223. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 15–19, 2010, pp. 631–648.

[38] A. Langley, M. Hamburg, and S. Turner, "Elliptic Curves for Security," Internet Requests for Comments, RFC Editor, RFC 7748, January 2016. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7748.txt

[39] A. Langley and Y. Nir, "ChaCha20 and Poly1305 for IETF Protocols," Internet Requests for Comments, RFC Editor, RFC 7539, May 2015. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7539.txt

[40] X. Li, J. Xu, Z. Zhang, D. Feng, and H. Hu, "Multiple handshakes security of TLS 1.3 candidates," in *2016 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, May 22–26, 2016, pp. 486–505.

[41] Y. Li, S. Schäge, Z. Yang, F. Kohlar, and J. Schwenk, "On the security of the pre-shared key ciphersuites of TLS," in *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, ser. Lecture Notes in Computer Science, H. Krawczyk, Ed., vol. 8383. Buenos Aires, Argentina: Springer, Heidelberg, Germany, Mar. 26–28, 2014, pp. 669–684.

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

[43] G. Lowe, "A hierarchy of authentication specifications," in *Proceedings 10th Computer Security Foundations Workshop*, 1997, pp. 31–43.

[44] J. Ludwig, "Wireguard key on an openpgp card," https://www.procustodibus.com/blog/2023/03/openpgpcard-wireguard-guide/, 2023.

[45] R. Merget, M. Brinkmann, N. Aviram, J. Somorovsky, J. Mittmann, and J. Schwenk, "Raccoon attack: Finding and exploiting most-significant-bit-oracles in TLS-DH(E)," in *USENIX Security 2021: 30th USENIX Security Symposium*, M. Bailey and R. Greenstadt, Eds. USENIX Association, Aug. 11–13, 2021, pp. 213–230.

[46] C. Meyer and J. Schwenk, "SoK: Lessons learned from SSL/TLS attacks," in *WISA 13: 14th International Workshop on Information Security Applications*, ser. Lecture Notes in Computer Science, Y. Kim, H. Lee, and A. Perrig, Eds., vol. 8267. Jeju Island, Korea: Springer, Heidelberg, Germany, Aug. 19–21, 2014, pp. 189–209.

[47] T. Perrin. (2018) The noise protocol framework. http://www.noiseprotocol.org/.

[48] D. Poddebniak, F. Ising, H. Böck, and S. Schinzel, "Why TLS is better without STARTTLS: A security analysis of STARTTLS in the email context," in *USENIX Security 2021: 30th USENIX Security Symposium*, M. Bailey and R. Greenstadt, Eds. USENIX Association, Aug. 11–13, 2021, pp. 4365–4382.

[49] J. Postel, "User Datagram Protocol," Internet Requests for Comments, RFC Editor, RFC 768, August 1980. [Online]. Available: http://www.rfc-editor.org/rfc/rfc768.txt

[50] M.-J. O. Saarinen and J.-P. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)," Internet Requests for Comments, RFC Editor, RFC 7693, November 2015. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7693.txt

[51] A. Suter-Döring, "Formalizing and verifying the security protocols from the noise framework, bachelor thesis," https://ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/information-security-group-dam/research/software/noise_suter-doerig.pdf, 2018.

[52] S. Taylor, "Wireguard vpn: Best vpns that support wireguard in 2023," https://restoreprivacy.com/vpn/wireguard/, 2023.

[53] L. Torvalds. (2020) Merge. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=bd2463ac7d7ec51d432f23bf0e893fb371a908cd.