A Spectral Analysis of Noise: A Comprehensive, Automated, Formal Analysis of Diffie-Hellman Protocols

Guillaume Girol, Lucca Hirschi, Ralf Sasse, Dennis Jackson, Cas Cremers, David Basin

A Spectral Analysis of Noise: A Comprehensive, Automated, Formal Analysis of Diffie-Hellman Protocols

Guillaume Girol
CEA, List, Université Paris-Saclay, France

Lucca Hirschi
Inria & LORIA, France

Ralf Sasse
Department of Computer Science, ETH Zurich

Dennis Jackson
University of Oxford, United Kingdom

Cas Cremers
CISPA Helmholtz Center for Information Security

David Basin
Department of Computer Science, ETH Zurich

Abstract

The Noise specification describes how to systematically construct a large family of Diffie-Hellman based key exchange protocols, including the secure transports used by WhatsApp, Lightning, and WireGuard. As the specification only makes informal security claims, earlier work has explored which formal security properties may be enjoyed by protocols in the Noise framework, yet many important questions remain open.

In this work we provide the most comprehensive, systematic analysis of the Noise framework to date. We start from first principles and, using an automated analysis tool, compute the strongest threat model under which a protocol is secure, thus enabling formal comparison between protocols. Our results allow us to objectively and automatically associate each informal security level presented in the Noise specification with a formal security claim.

We also provide a fine-grained separation of Noise protocols that were previously described as offering similar security properties, revealing a subclass for which alternative Noise protocols exist that offer strictly better security guarantees. Our analysis also uncovers missing assumptions in the Noise specification and some surprising consequences, *e.g.,* in some situations higher security levels yield strictly worse security.

For reproducibility, the sources of our tool Vacarme *and all Noise protocol models are available [18]. A technical report with additional details and proofs is available at [17].*

1 Introduction

The Noise framework [24] defines a set of protocols that enable two agents to establish a secure channel. Some of its protocols serve as building blocks in widely used protocols, including WhatsApp, Lightning, and WireGuard [13, 19, 23].

In a Noise protocol, the agents first exchange messages that constitute a *handshake*, derive from these messages a symmetric key, which they use to encrypt and integrity protect all following messages exchanged during their session.

Noise allows an unbounded number of distinct handshakes. Each variant can be described by a small, human-readable string, called a *pattern*. Some patterns are for two peers who know each other's long term key before starting the session. Others are designed for a client without a long-term key, who connects to a server whose long-term key is *a priori* unknown. Some patterns have a one round-trip handshake, resulting in low latency, whereas others feature a two or more round-trip handshake, which increases latency but may help hide the identity of peers to outsiders. Moreover, message payloads can even be exchanged during the handshake, protected with the best key currently available, and the properties achieved may therefore differ from message to message until the handshake completes. All this makes Noise protocols very flexible. For example, WhatsApp, WireGuard, and Lightning use different Noise patterns in their transport layer. This flexibility also makes it hard to assess the guarantees provided by these patterns and to choose the best protocol given specific system assumptions.

We summarize prior work in Table 2, discussed in detail in Section 2.3. The most relevant prior work is the Noise Explorer tool [20], designed to analyze the informal security levels described in the specification, which we compare in Section 2.3.1. However, both the Noise specification and all prior works leave crucial questions open: First, which Noise protocol should practitioners use for a given scenario and initial key distribution? Second, Noise theoretically offers an unbounded number of protocols, but are they all interesting, or are some Noise protocols subsumed by others?

We answer both questions rigorously and systematically. We answer the first by providing the strongest threat model under which each protocol is secure, enabling practitioners to make a trade-off between security and privacy. For the second, we give a formal framework and a methodology for comparing patterns, which we implemented in a tool and evaluated on all the patterns from the specification. Our results notably show that there are optimal patterns for each protocol setup, so other patterns from the specification provide no additional benefits.

We establish our results in the symbolic model and use the state-of-the-art Tamarin protocol analysis tool [25] to formally analyze a substantially wider range of properties than previous works. This includes all classical security properties [22], under a broad class of threat models (along the lines of "Know your enemy" [2]), over all protocols in the Noise specification, on a per message basis.

Contribution

Just as a spectral analysis decomposes sound into its constituent parts, we use our new tool Vacarme to decompose Noise into its constituent components and study their interaction. Our primary contribution is a systematic, fine-grained analysis of the Noise protocol family, which answers the following questions: (a) Under which precise threat models are messages secure, *i.e.,* do both secrecy and agreement properties hold? (b) What are the anonymity guarantees for the main Noise protocols? (c) How should one choose a suitable Noise protocol, given a PKI infrastructure and requirements? We expand on these points as well as additional contributions in the following.

Threat Models and Protocol Hierarchies: We approach Noise protocol analysis systematically. For a set of atomic adversary capabilities (*e.g.,* key compromise) and standard security goals (*e.g.,* secrecy), we measure security by all combinations of the latter under the former. In doing so, we provide the most fine-grained analysis of the Noise framework to date (see Section 2.3): we consider ephemeral key reveals (omitted previously), secrecy for the recipient (only previously considered for the sender), anonymity, *etc.* This yields a rich algebra of security properties that captures the full spectrum of use cases and security requirements of the Noise framework. We formally prove how each message in a Noise protocol can be attributed with *maximal* security guarantees in the form of the *strongest threat models* under which confidentiality, authentication, or anonymity holds. Finally, we show how these strongest threat models can be used to compare Noise protocols and determine when one protocol provides better security and anonymity than another, for any threat model.

Analysis with Vacarme: We show how to efficiently compute the strongest threat models using Tamarin as a back-end and we implement this methodology in our Noise protocol analysis tool Vacarme (French for "lots of Noise"). Our push-button tool thus leverages Tamarin's soundness and completeness guarantees [5,25]. Using Vacarme, one can automatically and formally assess under which threat models some requirements hold and compare different handshakes. We thus effectively answer the above questions (a) and (b), and are the first to analyze anonymity properties for Noise protocols. We also ran Vacarme on all Noise protocols listed in the specification both for evaluating our tool and for interpreting the analysis results. The results themselves yield the following contributions.

Refining the Noise levels: In contrast with the informal levels proposed in the Noise specification [24], our results have precise formal definitions, are machine-checked and considerably more granular. Further, our approach *objectively and automatically* assigns a formal meaning to the original levels as a special case.

Our results also uncovered several shortcomings of the Noise levels [24]. First, even though the levels appear to get stronger monotonically, as suggested by the Noise specification, we find that this is not actually the case. This is surprising and can lead to misguided protocol choices in practice. Second, we explain why the levels, as specified in [24], implicitly assumed that ephemeral keys cannot be compromised, which considerably weaken these guarantees. Finally, in contrast to the 9 Noise levels, we provide 74 distinct levels and show why this increased precision is crucial to well-informed protocol choices.

Selecting the Best and Identifying Redundant Protocols: Using our results, we automatically compared almost all Noise protocols listed in the specification and produced a hierarchy thereof. Using this hierarchy, we provide guidelines on which Noise protocol to choose, given a setup that describes what PKI or symmetric keys are available, and the expected range of adversary capabilities (threat model). We also identify redundant handshakes, which provide fewer guarantees than other handshakes, given the same setup. With regards to the Noise specification, we properly separate the threat-model assumptions, security goals, and monotonicity of security properties between handshakes. This allows practitioners to evaluate their environment assumptions independently of the goals they want to achieve, and enables them to pick the appropriate protocol required for their use-case, answering question (c).

Further Results and Recommendations for Noise: We make further contributions to the Noise framework and its application. An example thereof comes from our analysis of Noise protocols using a Pre-Shared Key (PSK): if a (publicly known) *dummy key* is used as PSK (a suggestion made in the specification), we show that, surprisingly, some protocols provide incomparable levels of security when using a dummy PSK compared with when using no PSK at all. Another example concerns anonymity, where our results reveal a missing requirement related to the handling of session identifiers.

Overall, our analysis uncovered numerous subtleties in the Noise specification and its protocols that were previously unknown. We also show how to systematically improve the specification, and we provide a tool to help practitioners.

Organization: In Section 2 we describe background on Noise and Tamarin, followed by detailed discussion of related work. We explain the security goals and threat models in Section 3 and present our tool Vacarme in Section 4. We discuss the results and practical implications in Section 5 and we draw conclusions in Section 6.

Nomenclature	Informal meaning
N	No static key available
K	Static key known before (e.g., via PKI)
X	Static key transmitted over the network
I	Static key transmitted earlier than with X
psk	Pre-shared symmetric key available
$n \in \mathbb{N}$	Appended to any other item, delays its use

Table 1: Summary of Noise options and nomenclature for fundamental patterns. A fundamental pattern consists of two letters and an optional *psk* token. The letter I may only appear in the first position.

2 Background and Related Work

We first describe the Noise handshakes, its pattern syntax, and security properties. Afterwards we provide background on the Tamarin prover and we discuss related work.

2.1 The Noise Framework

The Noise Protocol Framework [24] specifies a family of two-party handshakes for establishing secure channels. In addition to specifying 59 handshakes and claiming various security properties for them, it also specifies how additional handshakes can be derived. The proposed uses are extremely broad, ranging from handshakes between unidentified parties to handshakes between parties having pre-shared static asymmetric and symmetric keys.

2.1.1 Handshakes

Each handshake specified by the Noise Protocol Framework is built from a succinct set of simple primitives: a Diffie-Hellman group, a hash function, a key derivation function, and an Authenticated Encryption with Associated Data (AEAD) cipher. Although the specification is written in a generic fashion, it limits the instantiation of said primitives to a small selection, with a rationale for each choice. Thus, the security properties ascribed to each handshake are only claimed to hold for the given instantiations.

Each handshake is described by a *pattern* following a simple grammar. A pattern has two parts: *pre-messages* and *messages*. Pre-messages describe setup assumptions, namely knowledge that the parties must share before starting the handshake, for example keys given by a Public Key Infrastructure (PKI). Messages describe operations that each party must perform when sending or receiving handshake messages.

Pre-messages, messages, and computations thereon are described by a list of *tokens* and a *direction* specifying sender and recipient. Tokens refer to keys. Each party may have an ephemeral key (usually denoted by the letter `e`), and a static, or long-term, public key (usually denoted by the letter `s`). Additionally, the parties may share a secret called the PSK (a symmetric key usually denoted as `psk`). Not all patterns require all these keys.

Definition 1 (Handshake pattern)**.** *A* pre-message token *is* `e` *or* `s`*. A* message token *is* `e`*,* `s`*,* `es`*,* `se`*,* `ss`*,* `ee`*, or* `psk`*. Single letter tokens and* `psk` *are called* key tokens *and two-letter tokens are called* Diffie-Hellman tokens *(or DH tokens for short).*

A direction *is* `->` *or* `<-`*. A* pre-message *(respectively* message*) pattern is a pair of a direction and a non-empty list of pre-message (respectively message) tokens. A* handshake pattern *(or, for brevity, a handshake) has a name, and is a possibly empty list of pre-message patterns (followed by ellipsis if non-empty) and a non-empty list of message patterns. The list of pre-message patterns must contain at most one pre-message pattern per direction, and the message patterns must have alternating directions.*

As an illustration, we depict three handshake patterns in Figure 1. Noise handshake participants always exchange an ephemeral public key with the other party, and may optionally also exchange a static (long-term) public key or one or more PSKs. The name of the handshake defines the *fundamental pattern* and is given by two letters indicating how static keys are used. There are four possibilities for the initiator, and three for the recipient: *N* means that no static key is available, *K* means the partner already knows the static key, and *X* means the static key is transmitted to the partner. For the initiator, *I* is also available and means the static key is transmitted immediately, improving authentication properties for the recipient, at the cost of revealing the initiator's identity and thus a loss of anonymity for the initiator. The handshake's first letter depends on the initiator's key and the second letter on the recipient's key. Table 1 contains a reference summary.

Note that the `psk` key token can be used in a message pattern to indicate that both parties should use their PSK and mix it with their current symmetric key. This adds *psk* to the handshake name, making it a non-fundamental pattern. Note that by default each available key computation and transmission is done as early as possible.

To delay certain actions, *deferred patterns* are created by adding a number to the pattern name, which defers the transmission of the named token (see I1K1 shown in Figures 1 and 2) or the use of derived keys, e.g., `psk` (for *psk*), by a number of messages, usually 1 or 2. For example, *psk1* in a handshake name means that `psk` will be used one message after it could have been used. Deferred patterns were designed to improve identity hiding properties at the expense of latency. A special case of PSK usage is with a publicly known symmetric key, called a *dummy key*, which modifies the protocol's behavior without using any additional secrets.

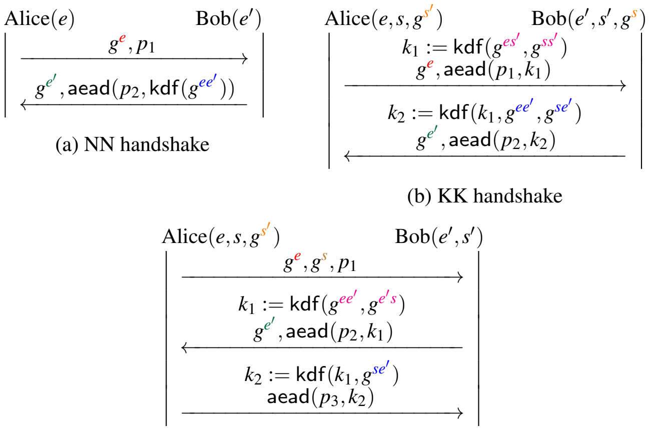
For the sake of brevity, we do not describe the formal semantics of tokens. We direct the interested reader to [17]. In the next example, however, we try to give an intuition of the semantics of a few patterns, omitting some details for simplicity.

```
NN:              KK:              I1K1:
  -> e             -> s             <- s
  <- e, ee         <- s             ...
                   ...              -> e, s
                   -> e, es, ss     <- e, ee, es
                   <- e, ee, se     -> se
```

Figure 1: Three Noise handshakes (colors match those of Figure 2, to help the reader, but are not part of the syntax)

Example 1 (Handshake syntax and semantics)**.** *Consider* NN *(shown in Figures 1 and 2a), a Noise handshake loosely corresponding to an unauthenticated Diffie-Hellman key exchange. There are no pre-messages, so the ellipsis is omitted. In the first message, the initiator (on the left) sends his ephemeral public key g^e, indicated by the key token* e *in the first message pattern. Each message sent ends with a payload, protected using AEAD under the best available key (this step is not materialized as a token in the Noise syntax, but comes implicitly at the end of all message patterns). Here, p_1 is sent in the clear. In the second message, the recipient (on the right) sends his own ephemeral key $g^{e'}$, indicated by the key token* e *in the second message pattern. The token* ee *means that when processing the second message, both parties derive a Diffie-Hellman term from their respective ephemeral keys. Specifically, after the second message, the initiator knows his private key e and the recipient's public key $g^{e'}$. He can thus compute $g^{ee'} = (g^{e'})^e$. The recipient's situation is symmetrical: he knows g^e and e' and can thus compute $g^{ee'} = (g^e)^{e'}$. Colloquially speaking, the DH term obtained by mixing the initiator's and recipient's ephemeral keys is $g^{ee'}$. They will use this value to seed a secret symmetric key, which is the initial* current *key, and is used to protect payloads, here p_2, with AEAD and the hash of all previous computation steps as additional data.*

One can also mix different keys as DH terms. When several such terms are present, they can be mixed *together using a key derivation function to obtain a new current key.*[1] *This is illustrated by KK, a Noise handshake loosely corresponding to an authenticated Diffie-Hellman key exchange shown in Figures 1 and 2b. Both pre-messages contain the same unique key token* s*, meaning both parties should already know their peer's public static key before starting the handshake. Namely, the initiator (respectively recipient) knows his private static key s (respectively s') and his partner's public static key $g^{s'}$ (respectively g^s). Then in the first message the initiator (i) sends g^e (key token* e*) in clear text because the current symmetric key is initially empty (i.e., not yet determined), (ii) computes the DH term $g^{es'}$ (DH token* es*) and mixes it with the current symmetric key that is then no longer empty (i.e., can now be used), (iii) computes the DH term $g^{ss'}$ (DH token* ss*) and mixes it with the current symmetric key (the resulting value is denoted as k_1 in Figure 2b), and finally (iv) sends the*



(a) NN handshake

(b) KK handshake

(c) I1K1 handshake

Figure 2: Alice & Bob notation for the handshakes of Figure 1. e, e' (respectively s, s') are ephemeral (respectively static) private keys and the p_i are payloads exchanged during the handshake. In transport mode, payloads are encrypted with the last key material used in the handshake. For legibility, we omitted the associated data of AEAD encryptions, which roughly corresponds to the hash of all preceding sent messages along with the public keys in pre-messages.

first payload protected using AEAD under the current symmetric key k_1 with the transcript of all messages exchanged so far as additional data. When receiving the corresponding message (i.e., the pair $\langle g^e, c\rangle$, where c is the encrypted payload), the recipient performs the same computations and obtains the symmetric key k_1 and can therefore decrypt c.

For the second message, the recipient sends $g^{e'}$ (key token e*), computes two DH terms corresponding to* ee *and* se*, and obtains the symmetric key k_2 accordingly. Similarly, the message ends with the second payload protected by AEAD with the key k_2 and the hash of all previous computation steps as additional data.*

Finally, the transport mode can start where all payloads are protected with AEAD under a derivative of the final symmetric key k_2 and empty additional data.

2.1.2 Security levels

The specification defines 3 source levels (degree of authentication of the sender provided to the recipient), 6 destination levels (degree of confidentiality provided to the sender), and 10 identity-hiding levels (protection of the sender's or the recipient's public key), where higher numbers indicate better security. The descriptions of these security properties are informal and non-trivial to interpret.

Example 2. *Destination Property 4, quoted from [24]:* '**Encryption to a known recipient, weak forward secrecy if the sender's private key has been compromised.** This payload is encrypted based on an ephemeral-ephemeral DH, and also based on an ephemeral-static DH involving the recipi-

[1] We use *mix* in an overloaded manner, to both denote DH-computation with two half-keys, and KDF-application with two secrets.

ent's static key pair. However, the binding between the recipient's alleged ephemeral public and the recipient's static public key has only been verified based on DHs involving both those public keys and the sender's static private key. Thus, if the sender's static private key was previously compromised, the recipient's alleged ephemeral public key may have been forged by an active attacker. In this case, the attacker could later compromise the intended recipient's static private key to decrypt the payload (this is a variant of a "KCI" attack enabling a "weak forward secrecy" attack).'

This informal description discusses how the encryption key has been derived and describes a possible attack that an attacker could use. However, it does not explore any other circumstances in which the encryption key might be compromised or how this property relates to more traditional notions of message confidentiality and authentication.

Example 3. *Source Property 2, quoted from [24]:* '**Sender authentication resistant to key-compromise impersonation (KCI).** The sender authentication is based on an ephemeral-static DH ("es" or "se") between the sender's static key pair and the recipient's ephemeral key pair. Assuming the corresponding private keys are secure, this authentication cannot be forged.'

The above definition is clearer than Example 2 insofar as it explicitly refers to a well known and established definition. However there is still considerable ambiguity. For example, is this authentication injective [22] (preventing replays)?

2.2 The Tamarin Prover

The Tamarin prover [25] (Tamarin for short) is a protocol verification tool for the *symbolic model*. Tamarin supports stateful protocols, a high level of automation, and equivalence properties [5], which are necessary to model privacy properties such as anonymity. Tamarin has previously been applied to numerous, substantial, real-world protocols with complex state machines, numerous messages, and complex security properties. Examples include TLS 1.3 [6, 10], mobile communication protocols [4, 9], and instant messaging protocols [8].

In the symbolic model, messages are described by terms. For example, $\mathsf{enc}(m,k)$ represents the message m encrypted using the key k. The algebraic properties of cryptographic functions are specified by equations over terms. For example, $\mathsf{dec}(\mathsf{enc}(m,k),k) = m$ specifies the expected semantics for symmetric encryption: decryption using the encryption key yields the plaintext. As is common in the symbolic model, cryptographic messages only satisfy those properties explicitly specified algebraically. This yields the so-called *black-box cryptography assumption*: one cannot exploit potential weaknesses in cryptographic primitives beyond those explicitly specified. Still, a wide range of attacks, including *logical attacks* and attacks based on an explicit algebraic model, are covered.

The protocol itself is described using multi-set rewrite rules. These rules manipulate multisets of *facts*, which model the current system state with *terms* as arguments. These rules yield a labeled transition system describing the possible protocol executions (see [3, 25] for details on syntax and semantics). Tamarin combines the protocol semantics with a Dolev-Yao [12] style adversary. This adversary controls the entire network and can thereby intercept, delete, modify, delay, inject, and build new messages.

In Tamarin, security properties are specified in two ways. First, trace properties, such as secrecy or variants of authentication, are specified using formulas in a first-order logic with timepoints. For each specified property, Tamarin checks that the property holds for all possible protocol executions, and all possible adversary behaviors. To achieve this, Tamarin (symbolically) explores all possible executions in a backward manner, starting from attack states, which are counterexamples to the security properties, and trying to reach legitimate starting states. The formulas constituting the specification are called *lemmas* and represent claims to be analyzed.

Equivalence properties, such as anonymity, are expressed by requiring that two instances of the protocol cannot be distinguished by the adversary. Such properties are specified using *diff*-terms (which take two arguments), essentially defining two different instances of the protocol that only differ in some terms. Tamarin then checks observational equivalence (see [5]). That is, it compares the two resulting systems and checks that the adversary cannot distinguish them for any protocol execution and any adversarial behavior.

In fully automatic mode, Tamarin either returns a proof that the property holds, or a counterexample, representing an attack, if the property is violated, or it may fail to terminate as the underlying problem is undecidable. Tamarin can also be used in interactive mode, where users can guide the proof search. Moreover users can supply heuristics called *oracles* to guide the proof search in a sound way. Given the number of handshakes and properties to check, we require fully automatic analyses. We thus rely on handshake-independent *oracles* in our analyses as they allow us to tame the protocol's complexity, as explained in Section 4.1.3.

We also describe the properties of the underlying cryptographic primitives used in Noise and how they are composed. As mentioned previously, Noise uses four distinct cryptographic primitives, which we model in Tamarin:

- Diffie-Hellman (DH) Group: We model both the case of a prime order group and Curve25519, a primitive recommended by the specification which is of non-prime order and contains a small subgroup, following [11]. Our Tamarin model also faithfully captures the symbolic behavior of the exponentiation operator, including the existence of multiplicative inverses, associativity, commutativity, and identity.
- AEAD: We model this as a distinguished function symbol that can either be decrypted (with the correct key

and nonce) or verified (ensuring the authenticity of the associated data).

- Hash and KDF functions: We model both as distinct function symbols that each behave as a random oracle.

2.3 Related Work

Our methodology builds on ideas presented in "Know your enemy" [2], which investigates the systematic integration of adversary capabilities in symbolic models. In this work, we improve and extend its approach, for example by leveraging both static and dynamic analysis and enlarging the set of adversary capabilities. Further, we apply this methodology on a much larger scale than in the original paper in terms of the number of protocols and properties compared.

Previous research [14, 15, 21] has examined the security of a single Noise protocol handshake (IKpsk2) in the context of the WireGuard VPN protocol. However, only two previous works have set out to formally analyze the Noise framework as a whole. In this section we discuss these works in detail and summarize the differences in Table 2.

2.3.1 Noise Explorer [20]

Noise Explorer is a tool that automatically generates formal models for Noise handshakes. The formal models encode the protocols as well as the secrecy and authentication claims drawn from the Noise specification and can be automatically verified using the ProVerif [7] protocol analysis tool.

Noise Explorer presents its analysis results in a human-readable way by translating the formal security claims that were (dis)proved to textual descriptions. Further, it can also be used to automatically generate a reference implementation for a particular handshake, which is, however, not formally related to the verified model, *i.e.,* these implementations are not proven correct or secure.

Methodology: Noise Explorer's approach differs substantially from our own. Their analysis begins with informal security claims in the Noise protocol specification which they manually translate to formal statements. This mapping between natural language in protocol specifications and logical formulas in formal models is subjective and risks human error. Later, in Section 5.2, we will show how our methodology avoids these issues by systematically constructing a granular family of threat models from which we can objectively and automatically recover the correspondence to the Noise protocol specification.

This methodological difference has a practical consequence as both the Noise protocol specification and Noise Explorer associate each security claim with a *level*, a natural number, and interpret it in a monotonic order. However, we show later in Example 9 that the claims ordering, given by logical implication on the associated formulas, is in fact non-monotonic with respect to the levels and consequently, in certain handshakes, an apparently 'stronger' security claim can in fact be weaker than a 'weaker' claim.

Participants and Sessions: Noise Explorer only considers a fixed scenario where an honest initiator interacts with an honest recipient in the presence of a single malicious party. In particular, this excludes an honest agent acting as both an initiator and a recipient, which is common in many real world deployments of Noise (e.g., P2P settings such as Lightning). Additionally, Noise Explorer does not support any additional identities or participants. Hence it does not consider attacks on authentication which require more than two honest participants to perform. In contrast we consider an unbounded number of participants engaging in an unbounded number of sessions, including scenarios in which honest participants act as both an initiator and recipient.

This fixed two party scenario has consequences for models involving a passive adversary. A passive adversary cannot emulate dishonest agents, so the models only consider two fixed, honest agents. As a result, one obtains incorrect results. For example, Bob, who can only act as a recipient, can obtain aliveness of Alice, who can only act as an initiator, upon reception of `e` from the first message of `NN`. However, in practice, the property is actually violated as this ephemeral key could have been sent by any other honest agent.

Security Claims: In the Noise protocol specification the PSK family of handshakes are presented without associated claims. We consider Perfect Forward Secrecy (PFS) in the context of the subsequent compromise of (i) a participants' static keys, (ii) pre-shared keys, or (iii) both static and pre-shared keys. Noise Explorer only evaluates the third scenario. However, in many real world deployments, pre-shared keys are not as well secured as static keys and may be shared across devices. As Noise Explorer does not consider the compromise of a PSK alone, it cannot be used to explore PFS in this scenario.

Noise Explorer uses a relatively weak form of message agreement. The strongest claim it can verify is that if Bob receives a message, then at some point Alice sent that message and intended to send it to Bob. This does not imply the absence of replay attacks (where Bob receives Alice's message more than once), nor does it imply that Alice sent the message in the same session that Bob received it. Contrastingly, our analysis covers these properties, which we discuss further in Section 3.2.

Noise Explorer does not verify security properties in the presence of compromised ephemeral keys. We explore this scenario and provide a full set of results in Section 5 which allow protocol designers to evaluate which handshakes are best suited to scenarios where RNGs may be suspect.

Cryptographic Primitives: Unlike Tamarin, ProVerif does not handle Associative-Commutative (AC) function symbols. Consequently, Noise Explorer has a lower fidelity model of DH exponentiation than our own. In particular, Noise Explorer does not consider $((g^a)^b)^c$ equal to $((g^a)^c)^b$. In contrast, we

	8 handshakes	All Noise handshakes	
	Dowling [16]	Noise Explorer [20]	**Our Work**
Setting (Model)	Computational	Symbolic	Symbolic
Automated & machine-checked	✗	✓(ProVerif)	✓(Tamarin)
Reduction to cryptographic definitions	✓	✗	✗
Systematic wrt. atomic capabilities	✗	✗	✓
Strongest threat model computation	✗	✗	✓
Generates reference implementations	✗	✓†	✗
Intruder-chosen payloads	✓	✗	✓
Compromise `s`/`e`/`PSK`	✓/✓/✗	✓/✗/✗	✓/✓/✓
Dishonest generation `s`/`e`	✓/✓	✗/✗	✓/✓
Active attacker	✓	✓	✓
Anonymous agreement	✗	✗	✓
Identity hiding (anonymity)	✗	✗	✓
PFS of keys/messages	✗/✓	✗/✓*	✓/✓

Table 2: *Not all formal analyses are equivalent.* We compare our framework and tool with prior works in terms of modeling choices, threat models, verification tools, and analyzed goals. Legend: †: These implementations are automatically generated, but not formally verified to be correct or secure. *: PFS results for PSK handshakes are incomplete. For "Compromise `s`/`e`/`PSK`", we require results with and without the corresponding compromise (see Section 3.3.1). "Dishonest generation of `e`/`s`" refers to our $D_{\mathtt{re}}$/$D_{\mathtt{rs}}$ intruder capabilities (see Section 3.3.1).

model DH exponentiation as an AC symbol and the preceding equality holds in our model. Both Noise Explorer and Tamarin model the possibility of small subgroup elements in X25519, however, only Tamarin models the possibility of 'equivalent' public keys (which are bitwise distinct elements that behave equivalently under exponentiation). The DH models used by Tamarin and ProVerif are compared and discussed further in [11].

2.3.2 fACCE Noise Analysis [16]

Recently [16] proposed a new computational model for analyzing multi-stage channel establishment protocols which is of independent interest. Their approach is more scalable than previous computational models and allows the authors to reuse proofs between related protocols in order to reduce the manual burden on the (human) prover.

They demonstrate the flexibility and efficacy of their model on the Noise protocol framework. As in our work they consider ephemeral key reveals and extend the Noise security claims. However, despite their improved model, analyzing each handshake is still a manual effort that requires significant work. Consequently, they focus on a subset of the Noise handshakes (8 of 59) and target strong security properties which hold only for later handshake messages. Contrastingly, we are able to cover the entire handshake space and explore the weaker properties that early handshake messages enjoy.

3 Security Goals and Threat Models

We describe in this section our formal model of the Noise Framework, including how we handle crucial questions such as the encoding of roles and identities, as well as security claims and attacker capabilities. Our descriptions here are mostly semi-formal, due to space constraints. The full formal definitions, theorems, and proofs are given in [17].

3.1 Protocol and Environment Description

Formal models of security protocols must make critical decisions about how to encode abstract notions such as agents' state, identity, and agents' interactions with other protocol participants. In this section, we explain our decisions, describe our model's behaviors, and justify our model's effectiveness. As Noise is a protocol framework designed to be used in concert with a higher level application about whose behavior we can make few assumptions, we shall keep our model as general as possible and avoid artificially restricting handshake behavior.

Agents and sessions: We describe the behavior of protocol participants in terms of agents with local state that engage in protocol sessions with each other. We allow for an unbounded number of agents, engaging in an unbounded number of sessions and allow each agent to engage in multiple concurrent sessions, potentially playing multiple roles. In contrast, some previous verifications of Noise [20, 21] assumed that there are only two 'honest' agents. Whilst this might be appropriate to model a single client talking to a fixed server, it does not capture more general deployments, with multiple clients, multiple servers, or parties that act as both, as in P2P networks like Bitcoin or Lightning. In general, we allow the adversary to determine when entities are created, when they engage in sessions, and with whom they communicate. In Section 3.3, we will discuss explicit adversary actions, such as compromising a party, creating a dishonest agent, etc.

Identities: Some formal models endow agents with unique identifiers, which are used in the protocol or in the protocol's security claims. Although internally we use unique identifiers to distinguish the local state of each agent, we do not otherwise use these artificial labels. Instead, agents represent each other's identities in terms of the keys used in each session. This captures behavior in handshakes with long-lived keys reused between sessions, as well as handshakes relying on PSKs for authentication or handshakes only providing ephemeral keys. This ensures we do not impose any artificial restrictions on applications using the Noise protocol framework, which make their own decisions as to how agents are identified.

The Noise framework does describe an explicit session identifier that is output to the application when a handshake concludes, which we use to identify specific sessions. We do not (a priori) assume that this identifier is unique or that the application keeps it secret. We will see how this conservative decision allows us to find a previously undocumented application requirement in Section 5.5. Additionally, we treat the identities of remote parties as a tuple of exchanged key material, for example, the other entity's public ephemeral, public static, or pre-shared keys which have been exchanged. This allows us to define a meaningful notion of identity even for handshakes without any long term secrets, i.e., that provide anonymous connections for one or more participants.

Pre-messages: In some handshakes, Noise supports pre-distributed public keys or PSKs, which one or both participants may have access to. In practice, an application using the Noise framework will describe how this information would be transferred and authenticated. Consequently, we treat this part of the framework abstractly and simply distinguish when the provided information is authentic, or when the adversary has tampered with it due to some compromise of the authentication infrastructure. For example, an application using a certificate-based system cannot distinguish between legitimate certificates and those an adversary has generated after compromising the CA. In a Trust on First Use model, this would mean (correctly) trusting an honest key or incorrectly trusting an adversary controlled key. We describe in Section 3.3 how we can use these recorded labels, in conjunction with our parameterized adversary, to capture the full spectrum of authentication behavior.

PSK: Similarly, we support the Noise PSK modes, which offer an alternative and complementary notion of a pre-distributed token. Noise does not specify how PSKs should be treated. For example, they could be uniquely issued to a specific pair of agents, thus authenticating each party to the other, or to a group of entities and thus provide only authentication to this group, which is weaker than pairwise authentication. In protocols using dummy keys, like WireGuard [13], the PSK may even be publicly known. We allow the adversary to assign shared keys to any combination of agents it wishes, which includes all of the previously described scenarios. This includes shared keys that are intended to be secret, but to which the adversary legitimately has access or shared keys that the adversary can access through dishonest means such as compromising an agent.

Payloads: As the Noise Framework allows an application to transmit data alongside message payloads, we carefully model this functionality to give the adversary the maximum possible power. For example, when we later consider agreement properties, we allow the adversary to specify each message payload, as well as the handshake's prologue. This can be interpreted as the adversary influencing or even dictating the application-level protocol. However, when checking for the secrecy of a given payload, we must model this one payload as a randomly drawn value, as is customary in the symbolic model.

Transport mode: When a handshake finishes, the Noise Framework describes a transport phase, where applications can send or receive messages to or from the other party. We treat these messages like the handshake payloads in the previous paragraph, with the addition of an explicit sequence number as described in the specification. Although in principle there can be many transport phase messages, and applications are not required to alternate between sending and receiving, we show that it suffices to consider the initial transport phase messages sent by each party, allowing us to exclude further transport phase messages from our model [17, page 34]. Intuitively, this is due to the fact that key material remains unchanged.

Consequently, in the remainder of this paper, we consider the worst case scenario for the application layer and make minimal assumptions, letting the adversary choose payloads except the ones for which secrecy should be proven. Our model is also useful for future application designers wishing to check the specific combination of their application with a particular Noise handshake. We make it easy to plug a Tamarin model of an application layer into our handshake pattern models. One can thereby derive specific guarantees about the composition of both protocols, which will be at least as strong as the guarantees we discuss in this paper, as we assume the worst case application layer in our work.

3.2 Security Claims

Noise allows the application layer to send payloads alongside handshake messages, using the best available protection at that stage of the protocol. Consequently, these payloads may have weaker security guarantees than payloads sent later after the handshake's completion. The Noise specification claims informal security properties for each handshake message and for the first two payloads after the handshake's completion.

We analyze the security of each potential payload (*i.e.,* reasoning on a per message basis) but consider well-defined security claims based on a comprehensive set of threat models. We now describe these security claims and describe the threat models in the next section. Our claims can be parameterized by a role (Initiator I or recipient R), and a payload position, which indicates its location in the handshake.

Definition 2 (Claims)**.** *We consider the following claims:*

Secrecy *of a particular payload at position $i \in \mathbb{N}$, from the perspective of a given role r.*

Non-injective agreement *from the perspective of a recipient accepting a payload at position $i \in \mathbb{N}$ on the payload content, its additional data, and the sets S_s, S_r of (supposedly) exchanged keys identifying respectively the sender and the recipient. S_s and S_r may contain PSK, public ephemeral key, or/and public static keys. If the claim is true, this means that if the recipient, identified by S_r, accepts a payload from a peer he believes is identified by S_s, it was at some point sent by a peer identified by S_s with an intended recipient identified by S_r. However, there is no injective correspondence between these events, i.e., replay is possible.*

Injective agreement *additionally requires that any successfully received message must correspond to a unique legitimate transmission, ruling out replay attacks.*

Anonymity *of a given role r with respect to its public static key.*

These claims have a standard formalization. Agreement claims are written as in [22]. Anonymity claims are encoded as observational equivalence, as is standard in the symbolic model setting [1, 5]. Specifically, anonymity is falsified when an adversary conforming to a given threat model can distinguish an agent using a public, static key g^s known to the adversary from an agent using a second public, static key $g^{s'}$ also known to the adversary. When this happens, given a list of 'candidate' public keys containing an agent's key, the adversary can recognize this agent.

Secrecy and agreement claims broadly correspond to the families of informal security levels given in the Noise specification: source and destination properties. Anonymity claims model a part of identity hiding properties, which are an informal notion used in the Noise specification that refers to the identities not being deducible by the attacker. In conjunction with our threat models, explained in the next section, we will later see that these claims encompass the informal descriptions from the Noise protocol specification and go considerably further in many respects.

3.3 Security Properties

We evaluate claims with respect to a range of threat models, which are modeled by describing the adversary's capabilities. A claim and a threat model together specify a security property, which we can evaluate. In this section, we describe the adversary's possible capabilities, how we combine these capabilities into threat models, and how we can concisely summarize the resulting information.

To motivate our formulation, let us focus first on secrecy and agreement properties. These have the general form of $\tau \implies C \vee t$, where τ represents a 'trigger' that occurs whenever the claim in question applies (*e.g.,* upon reception of a message for agreement claims), C describes the guarantees expected to hold for that claim (*e.g.,* secrecy of the exchanged payload for a secrecy claim), and t describes a threat model, which describes a combination of adversarial capabilities. Note that when combined with a claim to form a security property, threat models are *implicitly negated*, see Section 3.3.2. Thus a security property is a statement that the protocol provides the guarantees of the claim C we consider, unless the adversary has access to the capabilities described in t. We have already defined the claims we consider in the previous subsection. We now describe how we formulate the threat model t.

$\mathtt{active}$	Active adversary
$R_{\mathtt{e}}$	Actor ephemeral key is revealed
$R_{\mathtt{re}}$	Peer's ephemeral key is revealed
$R_{\mathtt{s}}$	Actor's static key is revealed
$R_{\mathtt{rs}}$	Peer's static key is revealed
$R_{\mathtt{psk}}$	The pre-shared key owned by the actor for this session is revealed
$R_{\mathtt{e}}^{<}$	Actor's ephemeral key is revealed before the claim
$R_{\mathtt{re}}^{<}$	Peer's ephemeral key is revealed before the claim
$R_{\mathtt{s}}^{<}$	Actor's static key is revealed before the claim
$R_{\mathtt{rs}}^{<}$	Peer's static key is revealed before the claim
$R_{\mathtt{psk}}^{<}$	The pre-shared key owned by the actor for this session is revealed before the claim
$D_{\mathtt{pki}}$	Dishonest pre-message PKI
$D_{\mathtt{re}}$	Peer's ephemeral key is dishonestly generated
$D_{\mathtt{rs}}$	Peer's static key is dishonestly generated

Figure 3: Atomic Adversary Capabilities. We refer to the set of capabilities as A.

3.3.1 Adversary Capabilities

Intuitively, our threat models can each be expressed as a combination of atomic adversarial capabilities. We summarize these capabilities in Figure 3 and explain their meaning here.

The symbol $\mathtt{active}$ denotes that the adversary is active. A passive adversary can only read, drop, and reorder messages, but not modify, send, or replay messages. R denotes a reveal or compromise of some key, and comes in two flavors: one where the reveal occurs before the time of the claim (*e.g.,* $R_{\mathtt{psk}}^{<}$) and one where the reveal can occur at any time (*e.g.,* $R_{\mathtt{psk}}$). D refers to dishonest key generation. Namely, $D_{\mathtt{pki}}$ expresses that the keys received by anyone as pre-messages, for instance through a PKI, can be dishonestly generated; *i.e.,* no assurance is provided of their well-formedness and received keys can be, e.g., $g^{s^{-1}}$ or g. $D_{\mathtt{rs}}$ expresses that the peer's static public key could be dishonestly generated.

Note that there is no D_s or D_e as we assume that the actor's private static and ephemeral keys were honestly generated. However, we only make this assumption for the actor, that is the honest agent for which a security guarantee must be provided, and not for other actors, most notably the actor's peer (see $D_{\mathtt{rs}}$, $D_{\mathtt{re}}$, and $D_{\mathtt{pki}}$).

These capabilities capture a realistic class of adversarial capabilities, including the ability to compromise the private state of the local or remote party, interfere with the application layer authentication system, and register malicious agents with the adversary's choice of key. In [17, Section 2.2.4], we provide a formal interpretation in our model of each of these capabilities, which we lack the space to explore here.

Anonymity claims: The anonymity claims are substantially more complex to model and analyze as they rely on observational equivalence (see Section 2.2). Such properties are well-known to be computationally much more expensive to analyze than trace properties.

For this reason, we analyze anonymity claims with respect to a strict subset of adversary capabilities, namely $A_a = \{R_{\mathtt{rs}}, R_{\mathtt{psk}}, \mathsf{active}\}$ instead of A. The adversary does not have access to other capabilities: ephemeral keys cannot be revealed, the PKI is honest, and the peer of the role whose identity we try to hide always receives honest static keys. We also assume that there is at most one initiator and one recipient, and that Diffie Hellman operations are implemented on a prime order group.

Albeit strict, these restrictions still allow us to gain useful insights about Noise's anonymity guaranties as we will see in Section 5.4.

3.3.2 Threat Models

We now model an adversary who possesses a given subset of these capabilities, including all or none of them. We first describe how these capabilities can be combined into a threat model and afterward how they can be used to evaluate a security property.

Definition 3. *Let A be the set of adversary capabilities given in Figure 3. We define the set of* threat models*, denoted by T, to be the (subset of) propositional logic formulas, built from A, $\wedge$, $\vee$, and the bottom element $\bot$, which represents an empty threat model.*

Note that when combined with a claim to form a security property, threat models are *implicitly negated*. That is, a threat model does *not* describe the adversary's permitted capabilities, but rather its excluded capabilities, *i.e.,* the threat model determines under what circumstances the claim is not required to hold. Hence the empty threat model $\bot$ affords the attacker the most power as the claim must hold in all circumstances (and vice versa for the maximal threat model). However, not all combinations of capabilities are meaningful. For example, $R^<_{\mathtt{rs}} \wedge R_{\mathtt{rs}}$ is intuitively equivalent to threat model $R^<_{\mathtt{rs}}$ as revealing the key prior to the claim also satisfies the requirement to reveal the key at any point. We define a notion of redundancy that we use to eliminate such redundant threat models.

Definition 4. *We define $\preceq$ to be the smallest reflexive and transitive relation over threat models containing: $R^<_x \preceq R_x$ for $x \in \{\mathtt{e}, \mathtt{re}, \mathtt{s}, \mathtt{rs}, \mathtt{psk}\}$, $D_{\mathtt{pki}} \preceq D_{\mathtt{rs}}$, and $D_x \preceq$ active, for $x \in \{\mathtt{rs}, \mathtt{re}, \mathtt{pki}\}$.*

For $t_1, t_2 \in T$, we say that t_1 subsumes *t_2 when $t_1 \preceq t_2$.*

We can use this to reduce the number of relevant threat models using the following result.

Theorem 1. *Let C be a claim, and $t_1, t_2 \in T$ be such that $t_1 \preceq t_2$. If C holds in threat model t_1 then it also holds in t_2.*

Our ordering $\preceq$ induces a partial order on the set of threat models T, and thus yields an equivalence relation $\simeq$ defined as $t_1 \simeq t_2$ when $t_1 \preceq t_2$ and $t_2 \preceq t_1$. We denote as $\bar{T}$ the quotient of the set of threat models by $\simeq$. The set $\bar{T}$ represents the set of *distinct threat models* that we will consider.

$\bar{T}$ is still large: it contains more than 10^{12} elements. Although this indicates how fine-grained our analysis is, this large number poses two problems. First, the raw results of evaluating these threat models against each claim would be beyond human comprehension. Therefore, we develop a technique to condense these results into a single summary statement without any loss of precision. Second, evaluating all of these threat models would take substantial computational resources. We address this problem in Section 5.

3.4 Finding the Strongest Threat Model

We now show that for a given claim, there exists a unique element of $\bar{T}$ that subsumes exactly those threat models under which the claim holds. This allows us to summarize succinctly the conditions under which a claim holds.

Theorem 2 (Strongest threat model)**.** *Let C be a claim. Let $\bar{T}_1(C)$ be the set of threat models under which C is true. There exists a unique element in $\bar{T}_1(C)$ that subsumes all other threat models in $\bar{T}_1(C)$. We denote this element by $B(C)$ and call it the* Strongest Threat Model (STM) *for C.*

Without loss of generality, we can represent the unique STM by a representative of $B(C)$ in Disjunctive Normal Form (DNF), wherein there are sequence of clauses connected by disjunctions ($\vee$), and each clause is composed of conjunctions ($\wedge$) of adversarial capabilities. Each clause corresponds directly to a minimal set of capabilities required for the adversary to violate the security claim. Informally we will refer to this representative in DNF as *the STM*.

Example 4. *Secrecy of the third payload of I1N from the perspective of the initiator (called claimer) holds under the strongest threat model: $R_{\mathtt{re}} \vee D_{\mathtt{re}} \vee (R_{\mathtt{e}} \wedge R_{\mathtt{s}})$. This is equivalent to the following statement, where p denotes the payload:*

$$Trigger(p) \implies Secret(p) \vee (R_{\mathtt{re}} \vee D_{\mathtt{re}} \vee (R_{\mathtt{e}} \wedge R_{\mathtt{s}}))$$

Which means one of the following must be true:

- *The secrecy claim on the payload p holds, i.e., $Secret(p)$.*
- *The adversary compromised the peer's ephemeral key.*
- *The peer's ephemeral key was generated by the adversary.*
- *The adversary compromised both the claimer's ephemeral and static key (as modeled in the conjunct $R_{\mathtt{e}} \wedge R_{\mathtt{s}}$).*

These cases cover all possible attacks using combinations of atomic adversarial capabilities. Furthermore, each case is minimal, e.g., in the fourth case it must be that no attack is possible if the adversary only compromises the claimer's ephemeral key ($R_{\mathtt{e}}$) but not the claimer's static key ($R_{\mathtt{s}}$).

Consequently, for each claim C (secrecy, non-injective agreement, injective agreement, and anonymity), we can condense the result to a single threat model $B(C)$, which summaries the exact capabilities the adversary needs to violate the property given by C. This reduces our set of results for all claims to where they can be inspected by hand.

We now exemplify how this choice of threat models, combined with security claims, is expressive enough to encode well-known standard security notions.

Example 5. *Secrecy under $R^{<}_{\mathtt{rs}} \lor R^{<}_{\mathtt{s}}$ captures a form of PFS where payload secrecy holds* ***unless*** *the actor's or the actor's peer's static, private key is compromised before the claim.*

Example 6 (KCI resistance)**.** *Key Compromise Impersonation (KCI) resistance can be modeled as injective agreement under the threat model $R_e \lor R_{\mathtt{re}} \lor D_{\mathtt{re}} \lor R_{\mathtt{rs}} \lor D_{\mathtt{rs}}$. In plain English, agreement holds unless the actor's ephemeral key is compromised, or an asymmetric key of the actor's peer is either compromised or was in fact generated by the adversary. Hence even if the actor's static key is compromised, agreement still holds.*

However, we must still compute this STM. Naively, a brute force strategy enumerating all threat models in $\bar{T}$ would suffice, where we submit all proof obligations as lemmas to Tamarin for each claim. However, this would yield more than 10^{12} proof obligations per handshake, message, and security claim. We refine this approach so that computation is manageable in Section 4.1.

4 Vacarme

In the previous section, we described the security properties we consider. We now present our tool, called Vacarme, which is available at [18], and how we evaluated it.

Vacarme can take any two-way Noise pattern and computes the STM for each of its messages and security claim. Vacarme builds upon Tamarin [25] by first converting the pattern into a set of Tamarin proof obligations, running Tamarin on them, and finally analyzing the results.

4.1 Performance optimizations

Our methodology involves generating one Tamarin proof obligation (lemma) for each claim and each threat model. However, as noted previously, a naive brute force approach invoking Tamarin for each of them would require prohibitive computational resources. Instead, we use several techniques to reduce the overall computation time.

First, we employ static analysis to reduce the number of proof obligations required (Section 4.1.1). A runtime framework, described in Section 4.1.2, uses dynamic analysis to minimize the invocations to Tamarin given the results of already examined proof obligations. To further reduce analysis time, we developed a dedicated, but handshake-independent, provably sound Tamarin heuristic, which reduces Tamarin's proof search, that we describe in Section 4.1.3.

4.1.1 Static Analysis

We start with several a priori observations that reduce the number of Tamarin invocations.

Threat Models and Handshakes: Taking the quotient of T by $\simeq$ reduces the number of distinct conjunctions of atomic capabilities in A from 16,384, to 1,701. We explain and prove in [17] how and why results for conjuncts are enough to compute the STMs. We can also consider how elements of A interact with the claim under consideration. For example, if a claim considers a point in the protocol where an ephemeral key for a party has not yet been instantiated, we need not consider this key's reveal. Similarly, where the handshake pattern has no pre-messages, $D_{\mathtt{pki}}$ gives the adversary no additional power.

Threat Models and Claims: Next, we note that when analyzing agreement, a passive adversary cannot make use of any knowledge gained to affect the views or actions of the other participants, because they cannot insert their own messages. Hence we need not consider key reveals whilst considering agreement for passive adversaries. Similarly, for a passive adversary attempting to violate a secrecy property, the timing of a key reveal does not change the adversary's ultimate knowledge set, which means that there is no difference between the timed and untimed variants of a reveal, and we can infer the result for one by evaluating the other. Furthermore, revealing a key after a claim does not increase the adversary's ability to violate non-injective agreement at the time of the claim.

Trivial Attacks: In many threat models, the adversary may have enough knowledge to perform a *trivial attack* on a handshake. For example, if the adversary completely compromises the peer's state, then secrecy and agreement properties no longer hold. Similarly, if the adversary learns any PSKs present and at least one private key for each Diffie Hellman operation, they can compute the session key. We generalize these observations into a wider category of trivial attacks, which are important for our tool's efficiency. For such cases, we can immediately conclude that the claim is false based on a simple static analysis.

The above observations allow us to immediately infer that over 99% of proof obligations are false. This leaves us with, on average, only 63 proof obligations per remaining claim, as opposed to the 16,384 naive ones. There is a varying number of claims per pattern, depending on, *e.g.,* its number of messages. Overall, this leaves us with about 410,000 proof obligations for 53 patterns. Note that we show in [17] that all our reductions are sound in that we have formally proven that they not impact the actual results.

4.1.2 Dynamic Analysis

Our static analysis techniques substantially reduce the number of proof obligations, but the required computational effort would still be substantial. However, there are further relationships between these tasks that we can exploit. We lift our definition of subsumption (Definition 4) from threat models to proof obligations in the natural way. Namely, the subsumption relation is the smallest reflexive, transitive relation such that:

- if $t \preceq t'$ (t is a stronger threat model than t') then for any claim C, the lemma 'C holds in t' subsumes the lemma C holds in t';
- for any given threat model, message, and set of keys, injective agreement subsumes non-injective agreement; and
- for any given threat model and message, if S and S' are sets of keys where $S \subseteq S'$, then non-injective agreement on this message and keys S' subsumes non-injective agreement on the same message and keys S.

The resulting subsumption over-approximates, but does not coincide with, entailment, *i.e.,* for lemmas P_1, P_2, if subsumption relates P_1 and P_2, and P_1 is true then P_2 must also be true.

We can now consider the proof obligations (written as Tamarin lemmas, combining a claim and a threat model) we submit to Tamarin as nodes in a directed acyclic graph, whose edges are determined by the subsumption relationship which we can statically compute. In order to calculate the STM under which a claim holds, we must label each node in this graph with True or False. However, we can use the subsumption relationship to speed up this labeling. For example, if a property is true, then all weaker properties must also be true. Likewise, if there is a counterexample for a property, then that counterexample also holds for any stronger property.

We still have to choose a tree traversal strategy, *i.e.,* in which order we perform the individual proof obligations. We designed a dedicated heuristic that approximates the expected payoff, *i.e.,* how many tasks we could save from analysis. Overall, this reduces the number of proof obligations from 410,000 to 150,000. Again, the reduction is provably sound [17].

4.1.3 Proof Search Heuristic in Tamarin

We also introduce a new proof heuristic (also called an *oracle*) to improve Tamarin's performance and to prevent looping. Tamarin uses heuristics to prioritize which constraints should be satisfied whilst constructing a proof or counterexample.

By construction, poor heuristics cannot render Tamarin unsound, but they can slow it down. Previously, Tamarin's heuristics used only limited information about the current set of open constraints in order to determine which constraints should be prioritized. We improve upon that by additionally examining the entire constraint system from a *global* view.

Thanks to this extra flexibility, we are able to design a heuristic that delays the introduction of new identities or sessions into a trace. We ensure that all constraints concerning the already present sessions are first satisfied, before we consider constraints that might require the introduction of a new party. This ensures that we find straightforward contradictions early on, before investigating more complex scenarios. In this work, we can ensure this condition syntactically by inspecting the constraints output by Tamarin.

We stress that our oracle is handshake-independent and does not impact the validity of our results: Tamarin remains sound, and for trace properties, complete.

4.2 Toolchain and Evaluation

Toolchain: Vacarme's core consists of 5k lines of Rust. First, a generator converts any Noise pattern into a set of Tamarin input files that describe the protocol and all proof obligations using aforementioned static analysis. Given these Tamarin models, our runtime framework, a combination of Python and bash scripts, runs Tamarin with our oracle using the aforementioned dynamic analysis. The complete toolchain is *push-button*: given any Noise handshake, written in the Noise syntax as in Figure 1, or any Noise handshake name from the specification, it returns a table of STMs for all claims and messages. We also provide tool support to interpret the results and compare handshakes, as explained in Section 5.

Evaluation: While Vacarme can take an arbitrary two-way Noise pattern as input, we ran it on all such patterns that are listed in the specification, both for evaluating our tool and for interpreting the analysis results. To determine the STM for secrecy and agreement properties for the 53 two-way Noise patterns described in the specification, Vacarme required a total of 150,000 lemma evaluations, requiring 74 CPU-days on cores ranging from 2.2 to 2.6 GHz, and a peak requirement of 75 GB RAM. Anonymity proofs for a relevant subset of 46 patterns took another 97 CPU-days, with a peak memory usage of 125 GB RAM. The complete results and the tool to reproduce them are available at [18], and a large subset is also available at [17, Appendix B, pages 78–95]. We discuss these results in Section 5.

5 Analysis Results and Practical Implications

We describe our analysis results and their implications for the Noise protocol community. First, in Section 5.1, we use Vacarme to infer the strongest security properties for all two-way handshakes mentioned in the specification, which allows us to construct a protocol hierarchy. This enables one to optimally choose a handshake given, e.g., the PKI context of a given application. This includes a discussion on deferred patterns, for which we show that few of them offer useful trade-offs. Their improvement is for privacy only, and generally only one deferred pattern is relevant for each fundamental or PSK-based pattern. Second, in Section 5.2, we revisit the security levels claimed in the Noise specification and derive a formal interpretation of them from first principles. Along the way, we uncover some surprising properties about the security levels claimed by the specification. In Section 5.3, we revisit some of the subtleties surrounding protocols with a PSK and how they relate to non-PSK modes. Furthermore we make specific security claims for the various PSK handshakes, which are missing from the Noise protocol specification. Finally, we present anonymity results in Section 5.4: *e.g.,* session identifiers put privacy at risk, and some identity-hiding levels are flawed. We conclude by summarizing our most important recommendations for the Noise specification in Section 5.5.

General Properties: Our systematic approach also enables us to discover some general properties that all analyzed handshakes satisfy. First, for injective agreement we observe one of two cases: (i) the STM for it is either exactly the same threat model as for non-injective agreement, or (ii) injective agreement fails under all (even the weakest) threat models. In other words, messages can either be trivially replayed, or never. Second, in handshakes where public keys are distributed by a PKI, an agent only needs to trust that its own channel to the PKI is secure (and not necessarily the channel between the PKI and the peer). Finally, we observe that the guarantees offered by successive handshake messages are monotonic: subsequent messages enjoy, at worst, the same properties as previous messages.

5.1 Selecting patterns using hierarchies

Our results can be used to choose a suitable Noise pattern for a given context, such as a given key infrastructure. To do so, we must first define our system parameters, which allow us to partition the set of patterns into classes that correspond to distinct real-world use cases. Second, we introduce an order on a handshake's security properties. Together, these enable us to systematically infer the optimal handshake for a given scenario.

System Parameters: System parameters describe which parties are capable of storing static keys or shared symmetric keys and whether they are available in advance to remote parties. For example, in settings like web browsing, initiators may not require any authentication, but servers require authenticating against a pre-shared static public key. Thus, we identify the following *system parameters*:

1. Which roles have their own individual static key. This can be either none, initiator, recipient, or both.
2. If there are static keys, whether these are transferred before or during the handshake.
3. Whether a PSK is available.

For example, if a designer has access to a pre-shared static key for the recipient, a non-pre-shared static key for the initiator, and no PSK, then the Noise specification offers 4 possible patterns: KX, K1X, KX1, and K1X1. The last three are *deferred patterns*, which were designed to improve identity hiding properties at the expense of latency. As we will see later, our results show that these deferred patterns in fact provide no better security properties than KX. To establish such results, we shall formally compare handshakes, as explained next.

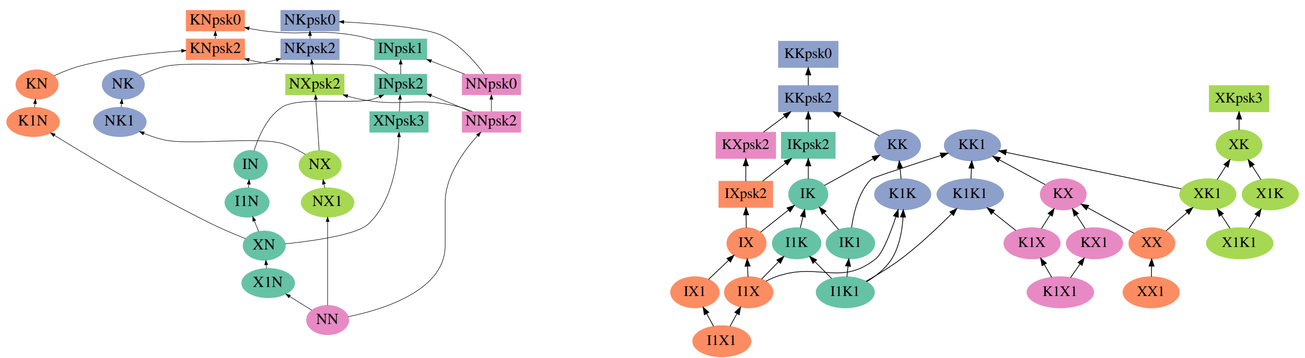
Order on handshakes: We say that a handshake A offers better security properties than B if for all claims C and threat models t, if B satisfies C in t, then A also satisfies C in t. Intuitively, this means that the handshake A provides better security than B for every claim, i.e., it is secure against stronger adversaries. This relation is easily computable from the STMs we obtained for each claim.

Hierarchy: We apply the previous methodology to the 46 handshakes[2] where we could compute a STM for anonymity in under about 100 CPU-hours per proof obligation. For all these handshakes, we could also compute the STMs for secrecy and agreement claims. A selection of these results is shown in Figure 4. Overall we see that *(i)* in most cases adding a PSK improves properties, and the earlier the PSK is used the better; *(ii)* few deferred handshakes are actually useful. In the remainder of this section, we expand upon and justify the latter claim.

Redundant patterns: Overall secrecy and agreement can be optimized together by avoiding deferred patterns: the earlier payloads are encrypted, the better (Figure 4a). In contrast, secrecy/agreement and anonymity are antagonistic: for example, KK1 provides better anonymity but worse secrecy/agreement than KK. They require the same system parameters (no PSK, pre-shared static keys for both agents) and represent thus two incomparable trade-offs. This can be seen on Figure 4b as KK and KK1 are two *maximal, incomparable elements* among blue, oval nodes. Nevertheless we show that each class of identical system parameters admits at most two maximal elements. Practitioners need only consider these handshakes, as other ones, which we call *redundant*, offer inferior security properties. Overall, only 2 deferred patterns are not redundant (NK1 and KK1).[3] For example, KK and KK1 make K1K and K1K1 redundant, and KX makes K1X, KX1, and K1X1 redundant. In particular, although the Noise specification introduced deferred patterns for their better anonymity properties, we found out that KX has strictly stronger anonymity guarantees than KX1. Table 3 summarizes these remarks for all non-PSK system parameters.

[2] 4 patterns timed out: IKpsk1, X1X, X1X1, XXpsk3; NN and its derivatives do not involve static keys.

[3] NK offers stronger secrecy and agreement than NK1, as shown in Figure 4a, but NK1 offers better anonymity properties, which are not considered in Figure 4a.



(a) Hierarchy of secrecy and agreement guarantees for protocols that do not require both the initiator and recipient having a static key. In this figure, we do *not* include anonymity properties to highlight the structure with only two maximal elements.

(b) Hierarchy of secrecy, agreement, and anonymity guarantees for protocols that assume both the initiator and recipient have a static key.

Figure 4: Excerpts of our Noise protocol hierarchy . An arrow from K1K to KK means that for every threat model t, if a security property holds for K1K in t, then it also holds for KK in t. In other words, KK offers stronger properties than K1K. Rectangles indicate variants that assume a PSK, and ovals variants without a PSK. Protocols with identical system parameters have the same color.

Role	Early use	Deferred use
Initiator	Always better properties	Always worse properties
recipient	Better secrecy/agreement	Better or worse privacy

Table 3: When to defer using a static key with the `es` or `se` tokens. The only non-redundant deferred patterns among those we analyzed are patterns where the first use of the recipient's static key (with `es`) is delayed by one round-trip.

5.2 Security Levels in the Noise Standard

The Noise protocol specification lists informal payload security properties called *levels* for each payload message of a handshake (*e.g.,* Examples 2 and 3). Correctly mapping a formal security model to informal prose is generally challenging. In this section, we explain how we automatically derive a formal definition of the Source and Destination Levels.

Relating Threat Models and Security levels: As we have seen, our fine-grained analysis allows us to associate each handshake payload with the STM under which secrecy for the sender and non-injective agreement between sender and recipient holds. Considering every handshake payload, we discover 74 unique STMs. That is, the security of any message payload is represented by one of those 74 STMs. These 74 distinct security classes can be related to the 9 levels discussed in the Noise specification by considering fewer atomic capabilities: instead of considering the STM among all possible threat models constructed from the set A of atomic adversarial capabilities as in Theorem 2, we consider the strongest of those threat models where no ephemeral key is revealed and the PKI is attacker-controlled, which we call the *simplified STMs*. Then the equivalence classes yielded by the relation relating handshakes with the same simplified STM fit exactly the Source and Destination Levels of the Noise specification, except for Source Level 0, as shown in Tables 4 and 5. In other words, our method is able to automatically derive a classification of payloads by their security properties that not only fits the Noise specification but additionally refines it by considering more adversarial capabilities.

Security	Level	Simplified Strongest Threat Model
Destination	0	$\top$
	1	$\mathsf{active} \wedge D_{\mathtt{re}}$
	2	$R_{\mathtt{rs}} \vee (D_{\mathtt{rs}} \wedge \mathsf{active})$
	3	$\mathsf{active} \wedge D_{\mathtt{re}} \wedge (D_{\mathtt{rs}} \vee R_{\mathtt{rs}})$
	4	$\mathsf{active} \wedge D_{\mathtt{re}} \wedge (D_{\mathtt{rs}} \vee R^{<}_{\mathtt{rs}} \vee (R^{<}_{\mathtt{s}} \wedge R_{\mathtt{rs}}))$
	5	$\mathsf{active} \wedge D_{\mathtt{re}} \wedge (D_{\mathtt{rs}} \vee R^{<}_{\mathtt{rs}})$
Source	1	$\mathsf{active} \wedge (R^{<}_{\mathtt{s}} \vee (D_{\mathtt{re}} \wedge (R^{<}_{\mathtt{rs}} \vee D_{\mathtt{rs}})))$
	2	$\mathsf{active} \wedge D_{\mathtt{re}} \wedge (R^{<}_{\mathtt{rs}} \vee D_{\mathtt{rs}}))$

Table 4: Interpretation of the source and destination levels of the Noise specification in terms of simplified STMs. The simplification consists of ignoring ephemeral key reveals and assuming the PKI is dishonest.

Using Figure 3 and Tables 4 and 5, we can now straightforwardly translate back the formal definitions of the levels we uncovered into intuitive, yet unambiguous statements. These threat models can be translated back into prose as well:

Example 7. *We can now properly define* ***Destination Level 4*** *(described in Example 2) as: Secrecy of the payload holds unless the adversary is active, the recipient's ephemeral key was generated by the adversary and*

- *the recipient's static key was generated by the adversary, or*
- *the recipient's static key is revealed before the message is sent, or*
- *the recipient's static key is revealed at any time and the sender's static key is revealed before the message is sent.*

Refining Source Level 0 (agreement): Although Source Level 0 is divided into four further levels, these levels are

Sub-levels of Source Level 0	Simplified STM
Source Level 0.0	$\top$
Source Level 0.1	active
Source Level 0.2	$\mathsf{active} \wedge (D_{\mathtt{re}} \vee R^{<}_{\mathtt{s}})$
Source Level 0.3	$\mathsf{active} \wedge D_{\mathtt{re}}$

Table 5: We found that Source Level 0 can be sub-divided into 4 sub-levels. As in Table 4, we assume here a dishonest PKI and the absence of ephemeral key reveals and higher number means stronger.

very weak. They range between 'The property never holds ($\top$)' and 'The adversary must be active and transmit a value.' The Noise specification refers to all such messages as being at the same level. The relevance of the additional security offered by the subdivision depends on the threat model. For example, in threat models where being active is very costly or impossible for the adversary, e.g., as in some mass surveillance scenarios, these subdivisions are meaningful.

Refining Destination Levels (secrecy): Unlike the Noise specification, we consider secrecy not only from the sender's point of view, but also from the recipient's point of view. We uncovered two new levels with corresponding simplified STMs: $R_{\mathtt{s}} \vee D_{\mathtt{re}}$ and $R_{\mathtt{s}} \vee (R_{\mathtt{re}} \wedge (R^{<}_{\mathtt{rs}} \vee D_{\mathtt{rs}}))$, which we will call $0'$ and $0''$ respectively, as they are incomparably strong. Notably, Level $0'$ is much weaker than Destination Level 1, but is found on messages that are Destination Level 2 for the sender's point of view. The following example illustrates why this could come as a surprise to some readers of the Noise specification.

Example 8 (Asymmetry of secrecy for the sender and recipient)**.** *We consider a threat model where no key is revealed and the agents have a way to ensure the authenticity of preshared static keys. We also consider the first message of X1K, which is Destination Level 2 for the initiator (sender). In such a threat model, when the initiator sends a payload, he has a guarantee that the attacker cannot learn the sent payload. In contrast, this message is Level $0'$ for the recipient of this message. When a recipient receives this message, it could be that the attacker knows the decrypted payload (for example, because the attacker impersonated the alleged sender). Depending on the application layer and the purpose of such a message, the recipient may need message confidentiality to be guaranteed.*

Uncovering Missing Assumptions in the Specification: The fact that when we consider threat models without ephemeral key reveals we obtain roughly the classification of the Noise specification suggests that the Noise specification actually *assumes* that ephemeral keys cannot be revealed. Currently, this assumption is only explicit for Destination Level 5 and Identity-hiding levels, which seems to (wrongly) imply it is not assumed for the others. In practice, weak ephemeral keys are possible with the many mobile phones, routers, and IoT devices suffering from poor quality RNGs. It is therefore prudent to systematically investigate this aspect of the threat model. Using our results and non-simplified threat models, it is possible to check whether a particular message would be revealed to an adversary in the event that one, or even both, of the involved parties had a faulty RNG. This allows a protocol designer to select a Noise handshake to mitigate this issue, when it is a real-world concern.

Pattern	Message	Destination Level	STM (secrecy for the sender)
IX	2	3	$(R_{\mathtt{re}} \wedge (R_{\mathtt{rs}} \vee R_{\mathtt{e}})) \vee (R_{\mathtt{e}} \wedge R_{\mathtt{s}}) \vee (\mathsf{active} \wedge (D_{\mathtt{re}} \wedge (R^{<}_{\mathtt{rs}} \vee R^{<}_{\mathtt{e}} \vee D_{\mathtt{rs}})) \vee (D_{\mathtt{rs}} \wedge R^{<}_{\mathtt{re}}))$
X1X	3	5	$(\mathsf{active} \wedge ((D_{\mathtt{rs}} \wedge R^{<}_{\mathtt{re}}) \vee (D_{\mathtt{re}} \vee (R^{<}_{\mathtt{rs}} \vee D_{\mathtt{rs}}))) \vee R_{\mathtt{e}} \vee (R_{\mathtt{re}} \wedge R_{\mathtt{rs}})$

Table 6: Strongest threat models (STM) for secrecy of some messages from the point of view of the sender, to illustrate Example 9.

Uncovering Non-monotonicity of Levels: A perhaps surprising consequence of considering such threat models is that the security levels given in the specification are *not monotonic*, while users and readers will most likely understand from the specification and their association with (linearly ordered) numbers that they are monotonic. Worryingly, a user may upgrade from one handshake to another with a higher Destination Level and, yet, lose security.

Example 9 (Non-monotonic secrecy upgrade)**.** *Consider the choice of a protocol with the goal of transmitting one payload from Alice to Bob. For the designer, only the secrecy of this payload, from Alice's point of view, matters. In the initial setup, both parties have a static key (but do not know their peer's static key in advance) and no PSK is available.*

We consider a scenario where Alice's ephemeral key can be revealed (i.e. Alice's device has a weak or faulty RNG) and the attacker is passive. Obviously no security is possible in the scenario where one party is fully compromised so we exclude it and we get the following threat model: $t = (R_e \wedge R_s) \vee R_{re} \vee \mathsf{active}$.

With Alice as the recipient, we see in Table 6 that the second message of pattern IX (labeled Destination Level 3) fulfills our requirements. Upgrading our guarantee from Level 3 to Level 5 should be an improvement, so we alternatively consider sending the sensitive payload as message 3 of pattern X1X with Alice as the initiator, which is labelled Level 5. However, under the above threat model t, this payload is not secret with X1X, since there is an attack when Alice's ephemeral key can be revealed ($R_{\mathtt{e}}$). This means that an 'upgrade' from a Level 3 to a Level 5 leads to an attack on secrecy, which the Level 3 handshake message would have prevented.

Note that we automatically identified this example from the table of all STMs, again illustrating our framework's expressiveness.

5.3 PSK Handshakes

There are no security claims about PSK patterns in the Noise specification. Our method automatically discovers the detailed Source and Destination properties of PSK patterns and we discuss our results here.

5.3.1 Degrees of PFS

As an illustration, we focus here on the insight we can gain from applying this methodology to secrecy from the sender's point of view. In particular, our analysis reveals three slightly different flavors of PFS, whose distinction can be crucial for protocol designers.

The first flavor (mapping to non-PSK Destination Level 5) corresponds to PFS relying on asymmetric keys: the attacker must have compromised the static keys before the session took place in order to break the secrecy of the messages. However, they can compromise the PSK before or after the session.

The second flavor is reverse: the PFS guarantee relies on the PSK only. The attacker must have compromised the PSK before the session took place, but they may compromise the static keys before or after the session.

The third flavor is the intersection of the two previous sets of guarantees, which is a kind of PFS that leverages both the PSK and asymmetric keys. The attacker must have compromised both the static key and the PSK before the session took place.

To illustrate why these distinctions are beneficial, consider the example of WireGuard (based on IKpsk2). WireGuard justifies using a PSK pattern by invoking post-quantum resistance [13, § V. B.], and allows the use of a public dummy PSK. PFS relying on asymmetric keys only is not post-quantum resistant, and PFS relying on the PSK only fails when using a public PSK. Therefore only the aforementioned third form of PFS is suitable for WireGuard's goals. In contrast, the Noise specification defines 6 levels, and in particular cannot distinguish between these flavors of PFS, while we do make this distinction and, moreover, effectively distinguish between 16 levels.

5.3.2 Non-PSK versus dummy PSK

As mentioned before, the Noise specification allows PSK-patterns to be used without securely distributing a PSK by setting the PSK to a public value like 0, called a *dummy* PSK. We model the public nature of these dummy PSKs by considering PSK-patterns where the PSK is immediately revealed to the adversary. We now compare such protocols with public dummy keys to the corresponding non-PSK patterns. Surprisingly, we find their security properties differ.

Before the first `psk` or DH token, PSK and non-PSK patterns have a different policy with respect to payload encryption. Non-PSK patterns send payloads in clear text, whereas PSK patterns send them with AEAD with a public value as the symmetric key. This has consequences for agreement and anonymity properties.

Agreement: The first message of some non-PSK patterns like NN (see Figure 2a) is Source Level 0.0 because the recipient cannot distinguish them from the second message of the handshake. Thus, even with a passive adversary, a recipient can mistake the second message of another handshake for a first message and falsely conclude that he has agreed upon a session with this non-existent initiator. In the corresponding PSK-handshake, encryption of the first payload prevents message confusion (even though the encryption key is public), and the payload of the first message becomes Source Level 0.1. Note, however, that Source Level 0.0 and 0.1 are both very weak (agreement is violated against an active adversary), and 0.1 could also be achieved in non-PSK handshakes using mere message tagging.

Anonymity: For KNpsk0 and KXpsk2, our analysis revealed that the anonymity of the initiator never holds in the PSK handshake with a dummy PSK, whereas it holds in some threat models in the corresponding non-PSK handshake. In the case of KXpsk2, this is due to the early encryption of payloads described above. Indeed, the authenticated data associated with the AEAD of the first payload contains a hash of the initiator's public key. Given a candidate public key, the attacker can compute the corresponding associated data and verify the integrity of the AEAD encrypted payload received over the network. The verification operation succeeds if and only if the candidate public key is correct, thus breaking the initiator's anonymity. This also affects several other PSK handshakes that are not part of our formal analysis.

We investigated the following modification of the Noise specification: the function `encryptAndHash` returns the cleartext when called before the first `psk` or DH token. With this modification, our tool proves that the initiator's anonymity is now guaranteed by KXpsk2 if the adversary is passive. This shows that encrypting payloads before the first `psk` or DH token can actually strictly weaken anonymity guarantees in some circumstances. Our tool also proves that this modification has no other effect on secrecy and agreement guarantees, except the effect on agreement discussed above. This modification has no effect on KNpsk0, however, as this handshake begins with a `psk` token. Strengthening the anonymity guarantees of KNpsk0 would require more involved modifications to `encryptAndHash`.

Except for the cases above, agreement, secrecy, and anonymity properties are the same for the dummy PSK and non-PSK handshakes we considered.

5.4 Anonymity Results

We did not run all anonymity proofs to completion, as observational equivalence is considerably more expensive to check than trace properties. Yet, our results for 46 patterns give some interesting insights. Firstly, the Noise specification allows applications to use the hash of the transcript of the handshake as a session identifier [24, § 11.2]. This hash is computable solely from public values, including the public keys of peers. Therefore, if the adversary has access to this hash and a list of public keys, he can discover the identity of the peers. As a result, anonymity cannot hold, even for a passive adversary. For this reason, the specification should clearly state that this session identifier must be kept secret by applications with anonymity requirements.

We did not model all nine Identity-hiding levels of the specification, but we do refine Level 7 of the Noise specification for example. It reads as follows: ‘An active attacker who pretends to be the initiator without the initiator’s static private key, who subsequently learns a candidate for the initiator private key, can then check whether the candidate is correct.’ The initiator of KN achieves Identity-hiding Level 7, and yet we find an attack against his anonymity under weaker assumptions: an active adversary and no key compromise. The adversary impersonates the recipient while guessing the initiator’s public key. The initiator accepts the second message of the handshake if and only if the guess is correct. This suggests that investigating every identity hiding level of the Noise protocol framework would be fruitful future work.

Finally, our work provides the very first machine-checked anonymity results for the Noise framework. We discuss our results at greater length in [17, Section 4.2], where we showcase the precision we can achieve through examples that meaningfully distinguish different classes of privacy attacks.

5.5 Summary of Analysis Insights

We summarize some of the insights provided by our analysis and make explicit recommendations for the Noise specification.

Session identifiers: If the session identifier defined in §11.2 of the Noise specification is public, then anonymity never holds (see Section 5.4). The specification should explicitly state that applications requiring identity hiding for their handshake must treat this session identifier as a secret value.

Early encryption in PSK patterns: As explained in Section 5.3, some payloads are encrypted before the first `psk` or DH token with a public value as a symmetric key. This brings some marginal benefits (agreement against a passive adversary for some early payloads, which could also be achieved by mere tagging) but violates the anonymity guarantees of several patterns like KXpsk2. The specification should highlight this unexpected impact on anonymity when using a PSK pattern with a dummy key.

Security claims: As explained in Section 5.2, security levels are given under the assumption that ephemeral keys cannot be compromised. This assumption should be made explicit, along with the consequence that the security levels are not monotonic. It should also be explained that secrecy from the recipient’s point of view is sometimes weaker than secrecy from the sender’s point of view as given by the Destination Level. Protocol designers may otherwise incorrectly assume there is no distinction between perspectives.

6 Conclusion

We have presented a fine-grained analysis of the protocols from the Noise specification, revealing subtle differences that were previously unknown and discovering classes of handshakes that should not be used. Our results help practitioners in selecting the right Noise handshake for their circumstances, for example by using our hierarchy in Figure 4b.

Our methodology is generic and not tailored to Noise. Hence it can be directly applied to other families of security protocols. One possible item of future work would be to compare the Noise handshakes and their properties with the security properties provided by other non-Noise authentication protocols. We would also like to further optimize our use of equivalence properties to analyze anonymity in greater detail and for more handshakes.

References

[1] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing Unlinkability and Anonymity Using the Applied Pi Calculus. In *Computer Security Foundations Symposium (CSF)*, pages 107–121. IEEE, 2010.

[2] David Basin and Cas Cremers. Know your enemy: Compromising adversaries in protocol analysis. *ACM Trans. Inf. Syst. Secur.*, 17(2):7:1–7:31, November 2014.

[3] David Basin, Cas Cremers, Jannik Dreier, Sasa Radomirovic, Ralf Sasse, Lara Schmid, and Benedikt Schmidt. The Tamarin Manual. `https://tamarin-prover.github.io/manual/`, 2019. Accessed: 2019-11-14.

[4] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirović, Ralf Sasse, and Vincent Stettler. A formal analysis of 5G authentication. In *Conference on Computer and Communications Security (CCS)*, pages 1383–1396. ACM, 2018.

[5] David Basin, Jannik Dreier, and Ralf Sasse. Automated symbolic proofs of observational equivalence. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1144–1155. ACM, 2015.

[6] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 483–502, May 2017.

[7] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and ProVerif. *Foundations and Trends in Privacy and Security*, 1(1–2):1–135, October 2016.

[8] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1802–1819. ACM, 2018.

[9] Cas Cremers and Martin Dehnel-Wild. Component-based formal analysis of 5G-AKA: Channel assumptions and session confusion. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

[10] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1773–1788. ACM, 2017.

[11] Cas Cremers and Dennis Jackson. Prime, Order Please! Revisiting Small Subgroup and Invalid Curve Attacks on Protocols using Diffie-Hellman. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*, pages 78–93. IEEE, 2019.

[12] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, March 1981.

[13] Jason A. Donenfeld. WireGuard: Next generation kernel network tunnel. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.

[14] Jason A. Donenfeld and Kevin Milner. Formal verification of the WireGuard protocol. Technical report, July 2017.

[15] Benjamin Dowling and Kenneth G Paterson. A cryptographic analysis of the WireGuard protocol. In *International Conference on Applied Cryptography and Network Security*, pages 3–21. Springer, 2018.

[16] Benjamin Dowling, Paul Rösler, and Jörg Schwenk. Flexible authenticated and confidential channel establishment (fACCE): Analyzing the Noise protocol framework. In *Proceedings of IACR International Conference on Practice and Theory of Public-Key Cryptography (PKC 2020)*, 2020.

[17] Guillaume Girol. Formalizing and Verifying the Security Protocols from the Noise Framework. Master's thesis, ETH Zurich, 2019. https://doi.org/10.3929/ethz-b-000332859.

[18] Guillaume Girol, Lucca Hirschi, Ralf Sasse, Dennis Jackson, Cas Cremers, and David Basin. Vacarme tool and all results. Available at https://github.com/symphorien/spectral-noise-analysis-usenix-artifact.

[19] WhatsApp Inc. WhatsApp encryption overwiew—Technical white paper. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf, December 2017. Accessed: 2019-11-14.

[20] Nadim Kobeissi, Georgio Nicolas, and Karthikeyan Bhargavan. Noise Explorer: Fully automated modeling and verification for arbitrary Noise protocols. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 356–370. IEEE, 2019.

[21] Benjamin Lipp, Bruno Blanchet, and Karthikeyan Bhargavan. A mechanised cryptographic proof of the WireGuard virtual private network protocol. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 231–246. IEEE, 2019.

[22] Gavin Lowe. A hierarchy of authentication specifications. In *Proceedings 10th Computer Security Foundations Workshop*, pages 31–43. IEEE, 1997.

[23] The Lightning Network. Bolt 8: Encrypted and authenticated transport. https://github.com/lightningnetwork/lightning-rfc/blob/130bc5da2c05f212fba09ae309e53fec8cde2c6d/08-transport.md, December 2017. Accessed: 2019-11-14.

[24] Trevor Perrin. The Noise Protocol Framework, July 2018. Revision 34, https://noiseprotocol.org/noise.html.

[25] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *Computer Security Foundations Symposium (CSF)*, pages 78–94. IEEE, 2012.