Master Thesis

Formalizing and Verifying the Security Protocols from the Noise Framework

Guillaume Girol

Supervisors:	Dr. L. Hirschi
	Dr. R. Sasse
Professor:	Prof. Dr. D. Basin
Issue Date:	September 14, 2018
Submission Date:	March 14, 2019

Departement of Computer Science, ETH Zürich

Abstract

Security protocols are one of the foundations of the modern Internet: they enable us to ensure authenticity, integrity, and confidentiality of communicated data. Flaws in their design are easy to miss and yet can have far-reaching consequences. Automatic verification of the security properties provided by a protocol in a mechanical fashion is a goal which has attracted a large research effort. Although in general such questions are undecidable, dedicated software like the Tamarin Prover [18] is able to analyse automatically various security properties for a wide range of protocols.

Noise is a framework for designing security protocols where two parties exchange keys to establish a secure channel. In order to adapt to most situations, Noise does not define one protocol but a family of dozens of *patterns* which attempt to fit most possible use cases.

As part of this thesis, we design a tool using the Tamarin prover as a backend to determine the security properties of any given two-way Noise pattern. The properties we consider are chosen to be standard (secrecy and agreement for payloads, anonymity of agents) and are tested in up to 10^{12} distinct threat models. We apply our tool to a set of 53 patterns to confirm and refine existing security analysis on these patterns.

Acknowledgements

I would like to thank my supervisors Lucca Hirschi and Ralf Sasse for dedicating time to guide me during this thesis. Their valuable insight and advice has been a key element to the successful completion this work.

Contents

Introduction

The modern Internet could not exist without secure one-to-one communications. To create such a *secure channel*, having access to suitable key material and robust cryptographic primitives such as hash functions or symmetric encryption is not enough: one must agree on a *protocol*. Designing a secure protocol is a notoriously difficult task. Furthermore, no single protocol can fit every use cases. As a result, many different protocols have been created. As new use cases appear, new protocols are written and the verification work must be done again from the ground up.

Noise The Noise framework [14] defines a set of protocols which enable two agents to establish a secure channel. These protocols fall in the family of authenticated key exchange protocols and are based on Diffie Hellman. The agents first exchange a few messages which constitute a so-called *handshake*, derive from these messages a symmetric key, and then use this key to encrypt and ensure integrity of their session. Noise allows many different handshakes; each variant is described by a small, human-readable string called a *pattern*. One can write patterns for many use cases. Some patterns are written 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 connecting to a server whose long-term key he does not know *a priori*. Some patterns have a one round trip handshake, leading to low latency, whereas others feature a two round trip handshake, which increases the latency but may sometime help hide the identity of peers to outsiders. The Noise specification alone presents 53 different patterns, each describing a protocol to establish a secure channel with slightly different guarantees and trade-offs. This makes the family of Noise-based protocols very flexible. For example, among protocols implemented on top of Noise, WhatsApp, Wireguard and Lightning all use different patterns [6, 9, 13].

Symbolic verification Security protocols are notoriously hard to get right, and a single overlooked detail can void the security of the whole protocol. A possible solution would be to enumerate all possible sessions and check that they all satisfy the security properties we expect. This is more or less the promise of symbolic verification. Provided with a suitably abstract model, software like the Tamarin Prover [18] can produce a formal proof that a protocol satisfies some properties. What do we mean by "suitably abstract"? The model is symbolic, which means that messages are not bitstrings but abstract variables like x. Cryptographic primitives are assumed ideal: if h is the symbol for a hash function, from $h(x)$ one cannot learn anything about x. Typically, from the encryption of a message one cannot learn even its length—unless one knows the key, of course. This level of abstraction thus masks all attacks based for example on the specific flaws of, say, textbook RSA. Computations are also symbolic so attacks relying for example on timing or other hardware side-channels are not easily modelled this way. The focus of tools like Tamarin is to defend against an untrusted network: the Dolev-Yao adversary. He can intercept, duplicate, create, and tamper with messages sent on the network. Therefore, the

scope of the symbolic verification could be described as: "Given perfect cryptographic primitives, can an adversary fully controlling the network use the protocol against us?"

Verifying Noise The number of Noise patterns is large enough to make human review of their individual security properties impractical and error-prone. Furthermore, Noise-based protocols have the specificity that the first four to six messages exchanged usually have distinct security properties. The reason is that contrary to traditional authenticated key exchange protocols, Noise allows agents to not wait for completion of the key exchange before sending payloads. The early payloads will be encrypted with an "incomplete" key, and thus may be less protected. As a result, the number of security properties to check is higher than with traditional authenticated key exchange protocols. For these reasons, it is quite natural to resort to formal methods to determine the security properties of Noise patterns. This is our objective in this thesis.

Related work The Noise specification [14] contains a table of security properties for a selection of handshake patterns. Namely: "source properties" (akin to agreement on messages, from level 0 to level 2), "destination properties" (secrecy of the payloads, from 0 to 5) and "identity hiding properties" (from 0 to 9). These security levels have arguably fuzzy descriptions and are not proved. The task of formally verifying them has attracted some amount of research interest.

In 2017, Donenfeld and Milner [7] have analysed the Wireguard protocol with Tamarin for key secrecy, key agreement, session uniqueness and identity hiding properties. This indirectly verified the properties of the single pattern IKpsk2, on which Wireguard is based. The identity hiding properties of the pattern are modelled as a property and not a hyperproperty with a method called "surrogate term": each time the public key of the anonymous agent is sent, a fresh term, called a surrogate term, is adjoined to it. The model then asserts that this fresh term is secret. Unfortunately, this method has the shortcoming that the surrogate term is fresh when the public key is reused across sessions.

In 2018, Kobeissi and Bhargavan [10] have designed a tool called Noise explorer to verify automatically source and destination properties of payloads. This study originally covered 38 patterns, and was enriched with 21 other patterns in 2019 [11], as this thesis was nearing its end. The properties proved are closely inspired from the security levels of the Noise specification and have therefore the same shortcomings. For example, secrecy of payloads is only considered from the point of view of the sender, and not of the receiver. Due to the limitations of ProVerif [5], the Diffie-Hellman theory is not as complete as one could wish. Notably, inverses are not modelled and multiplication is not fully associative.

In 2018 again, Suter-Dörig [17] verified all source and destination properties on a selection of 59 handshake patterns, and found results comparable to Kobeissi and Bhargavan [10]. This time the prover was Tamarin, which embeds a complete Diffie-Hellman theory, contrary to ProVerif. The analysis also included some other properties like key secrecy, key uniqueness, and executability.

Contributions Our main contribution is also a tool discovering which security properties any given two-way Noise pattern satisfies. It uses the Tamarin prover [18] as a back-end. The difference with previous work mainly lies in the properties we consider. Instead of checking which of the security levels of the Noise specification are satisfied, we focus on standard properties: secrecy of payloads, non-injective agreement and injective agreement on messages as defined by Lowe [12], and anonymity. Anonymity is based on observational equivalence and is a stronger property than when based on surrogate terms. Secrecy is analysed both from the point of view of the sender and the receiver. Each of these properties is analysed in up to 10^{12} distinct threat models, which enables us to cover standard concepts such as perfect forward secrecy and key

compromise impersonation resistance. In a construction reminiscent of Basin and Cremers [3], we also introduce the notion of *best threat model* and show that it accurately describes in which threat models a property holds. This enables us to summarize our results rather concisely. As a proof of concept, we apply our tool to the 53 two-way patterns mentioned in the Noise specification. We show that our results confirm and subsume the destination and source properties announced in the specification. Finally, our tool is designed to be reusable not only to investigate the properties of new patterns, but also to help model custom protocols built on top of a Noise pattern.

This thesis is organized in three chapters. The first chapter is dedicated to defining the formal semantics of patterns. The second one is dedicated to defining and formalizing the properties and threat models we consider, how we encode them for Tamarin and how we can summarize the results with the notion of best threat model. The third one described the results of running our tool on 53 patterns and comparing them to the Noise specification.

Chapter 1

Noise pattern semantics

In this chapter, we first describe the set of *patterns* defined in the Noise specification [14]. Next we give a first intuition of the meaning of a pattern. Then we define formal semantics for Noise patterns. This is done in two steps. First, after giving the necessary background on term algebras, we define so-called *high-level semantics* for Noise patterns, which are abstract enough to allow reasoning about patterns but still too abstract to capture protocol execution in minute detail. Then, after giving the necessary background about labelled multiset rewriting systems, we define *low-level semantics* for Noise patterns, which are fully precise but make reasoning impractical.

Note: The Noise specification is not in its final form yet; this thesis is based on its 34^{th} revision.

1.1 Pattern grammar

A Noise-based protocol always involves exactly two agents. It is divided in two *phases* or *modes*: the handshake phase, and the transport phase. The handshake phase is used to both establish session keys and convey encrypted *payloads*, and the transport phase uses the keys established in the handshake phase to convey further encrypted payloads.

The transport phase is nearly protocol-independent, but the handshake phase can vary greatly. This phase is described by a *pattern* following a rather simple grammar. The pattern is split in two parts: the pre-messages and the messages. Messages describe operations each party must accomplish when sending or receiving handshake messages. Pre-messages describe knowledge the parties must have exchanged before starting the handshake, for example by using a public key infrastructure. They are not necessarily real messages exchanged on the network.

Messages and pre-messages are described by a list of tokens and the direction in which they are sent. Tokens refer to keys. Each party may have an ephemeral key (usually denoted by the letter e), a static, or long-term, key (usually denoted by the letter s). Additionally, the parties may share the knowledge of a secret called pre-shared key (usually denoted as psk). Not all patterns require all these keys, though.

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 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) is a possibly empty list of pre-message patterns 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.

The syntax describing message patterns is straightforward: first, the direction, then the comma-separated list of tokens. For a handshakes pattern, the syntax is as follows: its name is followed by the list of pre-message patterns, then an ellipsis, then the list of message patterns. If there are no pre-messages, the ellipsis may be omitted.

Example 1 (Handshake syntax). Here is NN, a Noise handshake loosely corresponding to an unauthenticated Diffie-Hellman key exchange:

```
NN:
  -> e
  <- e, ee
```

There are no pre-messages, so the ellipsis is omitted. In the first message the initiator sends his ephemeral public key g^e. In the second one the responder sends his own ephemeral key $g^{e'}$. 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 responder's public key $g^{e'}$. He can thus compute $g^{ee'} = (g^{e'})^e$. The responder is in a symmetrical situation: 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 responder's ephemeral keys is $g^{ee'}$. They will use this value as a secret symmetric key to encrypt transport mode payloads.

On can also mix different keys as DH terms, and concatenate them all in a key. This is illustrated by KK, a Noise handshake loosely corresponding to an authenticated Diffie-Hellman key exchange:

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

Both pre-messages contain the same unique token `s`. They mean approximately: parties must already know their peer's public static key before starting the handshake. Then there are two messages. The first one means very roughly "send the initiator's ephemeral public key (token `e`) and then compute a Diffie Hellman term between the initiator's ephemeral key and the responder's static key (token `es`) and another Diffie Hellman term between the initiator's and responder's static keys (token `ss`). Concatenate them, and use them as a key." The second message means "send the responder ephemeral public key, and derive a DH term from both ephemeral keys, and from the initiator's static key and the responder ephemeral key. Then concatenate these DH terms to those obtained after the first message, and use the result as a key for the transport phase."

We will describe the semantics in more detail later, of course.

△

Definition 2 (Role, Action)**.** A *role* is either initiator or responder. The opposite role to role r is denoted as $\tilde{r}$.

An *action* is either send or receive. A message or pre-message pattern with the same direction as the first message pattern of a handshake is said to be sent by the initiator and received by the responder, and vice-versa.

Example 2. Again, considering NN:

```
NN:
  -> e
  <- e, ee
```

The first message was sent by the initiator and the second one by the responder. △

Note that although our generator of Tamarin theory supports either way, from now on, we will always consider handshakes whose first message direction is `->`, *i.e.* the initiator is on the left. A pattern where the initiator is on the right can always be converted.

1.2 Intuitive semantics

In this section we provide the reader with a simplified intuition of the meaning of a Noise pattern. This is not a fully precise depiction of reality, but will hopefully help the reader with tackling the more formal aspects to come.

Noise provides a cryptographic layer. A protocol designer uses it to convey *payloads* of their liking between parties without having to care about the details. We call *application layer* the abstract protocol dealing with payloads disregarding how they are encrypted. What we call the *Noise layer* is responsible for encrypting these payloads properly.

As already mentioned, a Noise-based protocol has two phases: the handshake phase, where a session key is established, and a transport phase, where the session key is used with a symmetric cipher. In common protocols, when the application layer requests the sending of a payload, the cryptographic layer will first wait for completion of the handshake before sending this payload encrypted with the session key derived from the handshake. In contrast, Noise allows payloads to be sent before the end of the handshake phase. This is to minimize latency. Of course, when sending a payload before completion of the handshake, the final session key is not available for encryption. So agents will use a provisional key k. Initially, k is empty. As the handshake progresses, new "ingredients" will be added to k, and at the end of the handshake k is used to derive two keys for the encryption of transport mode payloads (one per direction). When sending a payload is necessary, it will be encrypted with the current value of k if k is not empty. We could describe this as opportunistic encryption: halfway in the handshake, parties may know enough to compute a moderately strong key and can use this provisional key to encrypt the payload. On the contrary, at the beginning of some handshakes, parties can rarely do this because they know nothing of each other yet, so the payload is not encrypted at all.

Now that we have highlighted these peculiarities, we can describe how to interpret a Noise pattern. Each party has, as needed, an ephemeral key and a static key, and they can optionally share a pre-shared symmetric key (PSK).

Pre-messages represent which public keys parties should have exchanged before the start of the handshake. For example, `-> s` means that the responder should know the initiator's static public key before starting the session. How this information is conveyed is not specified; for example, this could be implemented as a conventional public key infrastructure, or information coming from previous sessions.

During the handshake phase, parties maintain a state k, initially empty. Messages are processed by each party in order. Tokens in a message are also processed in order.

- A key token means "send or receive this public key": ephemeral key for `e` and static (long-term) key for `s`. Static keys are encrypted with the current value of k, if k is not empty. Ephemeral keys are never encrypted.
- A DH token means "update the value of k to $\mathsf{kdf}(k, g^{ab})$", where
  - a is the private key of the initiator corresponding to the first letter of the token: s for static and e for ephemeral
  - b is the private key of the responder corresponding to the second letter of the token.
  - kdf is a key derivation function

  The party processing the token should know either a and g^b or b and g^a, depending on their role.
- A `psk` token means that the parties should update k to $\mathsf{kdf}(k, psk)$, where psk is a pre-shared symmetric key.

At the end of each message one payload is sent encrypted with the current k. If the current k is still empty, it is sent in clear text.

The handshake phase is followed by transport mode messages, which are more or less a pair of a sequence number and the payload encrypted with the last value of k.

About encryption: Noise uses authenticated encryption with associated data (AEAD). This form of symmetric encryption needs not only a (secret) key k but also so-called associated data a. Let x be a message. If Alice sends $y = \mathsf{aead}(k, a, x)$ to Bob, then Bob can retrieve x from y and k, as usual. Additionally, if he knows the associated data a as well he can check that Alice used the same a when encrypting. This can provide authentication of the message. In Noise, the authenticated data is approximately as follows:

During the transport phase: an empty string;

During the handshake phase: a hash of all previous messages plus all the public keys whose token has been processed, plus the pre-shared symmetric key if the token `psk` has been processed.

To keep notations in this section light we will set this detail aside and act as if Noise were using "traditional" symmetric encryption.

Example 3 (Intuitive semantics of NN). Again, let us consider NN.

```
NN:
  -> e
  <- e, ee
```

Assume that the application layer wants the initiator to send a payload p_1, then the responder to answer $f(p_1, p_2)$ and finally the initiator to send p_3. Both parties start with $k = \mathsf{empty}$.

The initiator sends the first message. He starts with an empty buffer for the outgoing message. When processing `e`, he appends his public ephemeral key g^{e_i} to the buffer. When reaching the end of the first message pattern, k is still empty, so he appends the unencrypted payload p_1 to the buffer. Finally, the initiator sends the content of the buffer: $\langle g^{e_i}, p_1 \rangle$, where angle brackets denote pairing.

When receiving this message, the responder first processes `e`: he expects and parses g^{e_i} from the beginning of the message. Then, reaching the end of the message pattern with empty k, he expects that the rest of the message is an unencrypted payload p_1.

The responder then sends the second message. He starts with an empty buffer. He processes `e` by appending his public ephemeral key g^{e_r} to the buffer. Then he processes `ee` by computing $g^{e_i e_r}$ from g^{e_i} (received as part of the first message) and e_r. He updates k to $\mathsf{kdf}(\mathsf{empty}, g^{e_i e_r})$. Then he processes the end of message by appending the payload $f(p_1, p_2)$ encrypted by the current value of k to the buffer. Finally, he sends the content of the buffer: $\langle g^{e_r}, \text{encrypt}(f(p_1, p_2), \mathsf{kdf}(\mathsf{empty}, g^{e_i e_r}))\rangle$.

When receiving this message, the initiator starts by processing `e`. He expects his peer's public ephemeral key and thus reads g^{e_r} from the beginning of the message. Then he processes `ee`: knowing g^{e_r} and e_i, he computes $g^{e_i e_r}$ and updates k to $\mathsf{kdf}(\mathsf{empty}, g^{e_i e_r})$. Finally, reaching the end of the message pattern, he expects the rest of the message to be the encryption of the payload with the current value of k: he can compute

$$f(p_1, p_2) = \text{decrypt}(\text{encrypt}(f(p_1, p_2), \mathsf{kdf}(\mathsf{empty}, g^{e_i e_r})), \mathsf{kdf}(\mathsf{empty}, g^{e_i e_r}))$$

For the first transport message, the initiator send p_3 encrypted with the last value of k, and the responder decrypts it with this same key.

△

Note how different payloads are encrypted with different keys (or even, not encrypted at all) and have thus different security guaranties. When investigating the properties of Noise-based protocols, we will have to give the properties of each payload separately.

Even with this simple pattern, keys involved quickly become "towers of kdf" so, from now on, when we need to discuss informally such terms, we will use pairing as key derivation function: $\mathsf{kdf}(a, \mathsf{kdf}(b, \mathsf{empty})) = \langle a, b\rangle$.

1.3 High-level semantics of handshakes

Since we want to model Noise protocols in Tamarin, we will describe their semantics in terms of multiset rewriting systems.

To facilitate high-level reasoning on patterns, we will first convert the handshake pattern to so-called *high-level semantics* which directly expose the state of the handshake and only then convert this into a multiset rewriting system.

1.3.1 Term algebra

Here is the background needed to define a labelled multiset rewriting system. It is adapted and simplified from Schmidt [16].

A term algebra is a set of purely syntactic *terms* composed of *variables* and *function symbols*. They do not carry any intrinsic meaning and if $+$ is a function symbol and x, y are variables we have in general $x + y \neq y + x$.

Definition 3 (Signature)**.** A signature is a finite set of pairs of a function symbol and an arity (non-negative integer).

Definition 4 (Term algebra)**.** Let Σ be a signature, $\mathcal{X}$ be a set of variables, and $\mathcal{N}$ a set of names. The term algebra over Σ, denoted as $T_\Sigma(\mathcal{X}, \mathcal{N})$ is the least set such that:

- $\mathcal{X} \subseteq T_\Sigma(\mathcal{X}, \mathcal{N})$

- $\mathcal{N} \subseteq T_\Sigma(\mathcal{X},\mathcal{N})$
- if $f \in \Sigma$ has arity n then $\forall (t_1,\ldots,t_n) \in T_\Sigma(\mathcal{X},\mathcal{N}), f(t_1,\ldots,t_n) \in T_\Sigma(\mathcal{X},\mathcal{N})$

A *term* is an element of the term algebra, and a *ground term* is a term without variables.

Here is the default signature of Tamarin: $\{\langle\cdot,\cdot\rangle, \mathsf{fst}, \mathsf{snd}\}$ for pairing and projection. As said above, terms are to be treated syntactically, so $\mathsf{fst}(\langle x, y\rangle) \neq x$. This is undesirable. Therefore, we will equip the term algebra with an *equational theory*, that is a set of equations, where an equation is a pair of term. The equational theory associated to the signature above is $\{\mathsf{fst}(\langle x, y\rangle) = x, \mathsf{snd}(\langle x, y\rangle) = y\}$.

Messages will be elements of a term algebra modulo an equational theory adapted to Noise. We now describe the signature we chose. We start with the default signature of Tamarin described above.

A Noise handshake in parametrized by a number of cryptographic primitives. We represent each of them as function symbols in our signature.

Noise uses AEAD encryption. We have an authenticated encryption function symbol aead of arity 4: it needs a (symmetric) key, a nonce, associated data and finally the clear text. Then we have a decryption function symbol $\mathsf{decrypt}$ of arity 3, needing only the key, nonce, and cipher text. Finally, we have a verification function symbol verif needing the key, nonce, associated data, and cipher text and will return the constant $\top$ if the cipher text is authentic.

The hash function symbol we use is h_f. The Noise specification [14] also requires a key derivation function `HKDF` of arity 3 which returns a pair if the third argument is 2 and a triplet if the third argument is 3. We model this function with the function symbols kdf_1, kdf_2, kdf_3 and the following equations:

$$\texttt{HKDF}(x, y, 2) = \langle \mathsf{kdf}_1(x, y), \mathsf{kdf}_2(x, y)\rangle$$
$$\texttt{HKDF}(x, y, 3) = \langle \mathsf{kdf}_1(x, y), \mathsf{kdf}_2(x, y), \mathsf{kdf}_3(x, y)\rangle$$

Finally, we add the signature for Diffie-Hellman terms which is built-in in Tamarin.

The resulting signature is then:

$$\Sigma = \{\langle\cdot,\cdot\rangle, \mathsf{fst}, \mathsf{snd}, \mathsf{aead}, \mathsf{decrypt}, \mathsf{verif}, \top, h_f, \mathsf{kdf}_1, \mathsf{kdf}_2, \mathsf{kdf}_3, 1, \cdot^{\wedge}\cdot, \cdot^{-1}, \cdot \star \cdot\}$$

with the equational theory $\mathcal{E}$:

$$\begin{aligned}
\mathsf{fst}(\langle x, y\rangle) &= x \\
\mathsf{snd}(\langle x, y\rangle) &= y \\
\mathsf{decrypt}(k, n, \mathsf{aead}(k, n, ad, x)) &= x \\
\mathsf{verif}(k, n, ad, \mathsf{aead}(k, n, ad, x)) &= \top \\
(x^y)^z &= x^{y\star z} \\
x \star (y \star z) &= (x \star y) \star z \\
x \star 1 &= x \\
x^1 &= x \\
x \star y &= y * x \\
x \star x^{-1} &= 1
\end{aligned}$$

From now on, terms will be elements of the quotient algebra $\mathcal{T} = T_\Sigma(\mathcal{X},\mathcal{N})/_{=_\mathcal{E}}$ where $\mathcal{N}$ is the disjoint union of public names PV and fresh names FV and $=_\mathcal{E}$ is the equality relation induced by $\mathcal{E}$ on $T_\Sigma(\mathcal{X},\mathcal{N})$. We also assume that $\mathbb{N} \subseteq PV$.

1.3.2 Noise state

Noise handshakes are stateful. The state of each party is composed of values of $\mathcal{T} \cup \{\mathsf{empty}\}$, where empty is a special placeholder for unavailable or unknown values. These values include two types of keys: *primitive keys*, which are provided by the application layer, and *derived keys*, which are computed during the handshake as specified by the Noise specification. A Noise handshake involves at most 5 primitive keys: the ephemeral and static keys of each party, plus the pre-shared key. At most one ephemeral key for each party can be used per session. Technically, all these keys are optional, even though validity rules (described later) effectively mandate the use of one ephemeral per party. The state an agent maintains contains the primitive keys he knows, some derived keys, and some other transient values: a counter and a hash.

Definition 5 (Noise state)**.** The state of a noise handshake is defined relative to a role as a named tuple of the following elements of $\mathcal{T} \cup \{\mathsf{empty}\}$.

- e the ephemeral private key of this role
- re the ephemeral public key of the other role
- s the static private key of this role
- rs the static public key of the other role
- psk the pre-shared key
- k a (derived) key
- ck a (derived) chained key
- h a hash
- n a non-negative integer (modelled as a public name). It is called a "nonce" in the specification but is rather a counter.

For a state σ, we denote the value of e as $\sigma[e]$ and the same state but with e updated to the value v as $\sigma[e := v]$. $\{e = x\}$ will also denote the state where e is the term x, and all other values are empty.

The rules for state evolution are described in Sections 5 and 9 of the specification. To describe them, we must first define the concept of needed keys, and then of the initial state.

Definition 6 (Needed key)**.** A key is said to be needed by a role if one of these cases apply:

- it is a private key owned by this role and the corresponding message or pre-message token (`e` or `s`) appears in a pattern sent by this role
- it is a public key owned by the other role and the corresponding message or pre-message token (`e` or `s`) appears in a pattern sent by the other role
- it is the pre-shared key and the message token `psk` appears in the handshake pattern.

Example 4 (Needed keys in NK). Here is NK:

```
NK:
  <- s
  ...
  -> e, es
  <- e, ee
```

The initiator needs an ephemeral private key and his peer's public static key. The responder needs a static and an ephemeral private key.

△

Part of the initial state of a role must be provided by the application layer. This includes needed keys and a so-called prologue. The prologue is a public[1] term on which parties must agree. If they do not, the protocol is not executable. Internally, this prologue is augmented by a string describing the handshake pattern (its name) and which cryptographic primitives were used. This prevents protocol mismatches. If parties do not agree on this string, the protocol is not executable either.

The initial value of transient elements in the state is determined by pre-messages.

Definition 7 (Semantics of pre-message patterns)**.** For a given role r, a prologue π, and a value for all needed keys, the initial state σ_0 of the handshake is defined as follows. Let σ_I be the state with all values empty except needed keys. Let m_I (respectively m_R) be the first pre-message pattern sent by the initiator (respectively responder), or the empty list. Let ν be the name of the handshake pattern. Let p be a boolean signalling whether the full handshake pattern contains the token `psk`. The initial state of the handshake for r is σ_0 as defined below in ML-style:

$$
\begin{aligned}
\mathsf{mixhash}\ \sigma\ x &:= \sigma[h := h_f(\langle h, x\rangle)] \\
\mathsf{mixkey}\ \sigma\ x &:= \sigma[k := \mathsf{kdf}_1(\langle \sigma[ck], x\rangle), ck := \mathsf{kdf}_2(\langle (ck, x)), n := 0] \\
\mathsf{get}\ \ell\ \sigma\ \mathsf{e} &:= \text{if } \ell \text{ then } g^{\sigma[e]} \text{ else } \sigma[re] \\
\mathsf{get}\ \ell\ \sigma\ \mathsf{s} &:= \text{if } \ell \text{ then } g^{\sigma[s]} \text{ else } \sigma[rs] \\
\text{preprocess true } \ell\ \sigma\ \mathsf{e} &:= \mathsf{mixkey}\ (\text{preprocess false } \ell\ \sigma\ \mathsf{e})\ (\mathsf{get}\ \ell\ \sigma\ \mathsf{e}) \\
\text{preprocess } p\ \ell\ \sigma\ t &:= \mathsf{mixhash}\ \sigma\ (\mathsf{get}\ \ell\ \sigma\ t) \\
\text{premsg } p\ \ell\ \sigma\ [] &:= \sigma \\
\text{premsg } p\ \ell\ \sigma\ (t :: ts) &:= \text{premsg } p\ \ell\ (\text{preprocess } p\ \ell\ \sigma\ t)\ ts \\
\sigma'_I &:= \mathsf{mixhash}\, \sigma_I[h := \nu, ck := \nu, n := 0]\ \pi \\
\sigma''_I &:= \text{premsg } p\ (r \stackrel{?}{=} \text{Initiator})\ \sigma'_I\ m_I \\
\sigma_0 &:= \text{premsg } p\ (r \stackrel{?}{=} \text{Responder})\ \sigma''_I\ m_R
\end{aligned}
$$

mixhash and mixkey are the analogue of the functions of the same name in the specification [14]. The function get computes a public key from a state: if ℓ is the boolean true, then it computes r's public key, otherwise if computes the public key of r's peer.

Example 5. With NK (see Example 4), for the initiator, with $\sigma_I = \{e = e_I, rs = s_R\}$ and $\pi =$ 'foo', we get $\sigma_0 = \{e = e_I, rs = s_R, h = h_f(\langle h_f(\langle$'NK', 'foo'$\rangle), s_R\rangle), ck =$ 'NK', $n = 0\}$. △

When receiving a message, an agent updates their state accordingly and decrypts the payload. When sending a payload an agent updates their state and sends the corresponding message. Therefore, we will model the processing of message patterns by a given role as a function from state to a pair of state and message. The message is sent if the action of this role on this message pattern is "send". If the role has the action "receive", then the message is input. Inputting a term means receiving a message and doing pattern matching against the structure of the term. This will be formally described later.

[1] More specifically, it should not contain key material. Key material should be used as a pre-shared key instead [14, Section 6].

At the end of each message, a payload is encrypted opportunistically with $\sigma[k]$ where σ is the current state. Note that this is mandatory: if an application layer wants to wait for completion of the handshake before sending payloads, it *must* send empty payloads during the handshake. It is easier for later reasoning to make this explicit by adding a new message token with this effect.

Definition 8 (Decorated handshake pattern)**.** A message pattern is decorated by appending the token `msg` to it. A handshake pattern is decorated by decorating all its message patterns.

Example 6 (Decorated NK). We are still considering NK:

```
NK:
  <- s
  ...
  -> e, es
  <- e, ee
```

The corresponding decorated handshake pattern is:

```
  <- s
  ...
  -> e, es, msg
  <- e, ee, msg
```

△

In what follows, we define semantics for decorated patterns only. The semantics of a non decorated handshake patterns is obtained by decorating it.

Definition 9 (Semantics of message tokens)**.** Given a role r, an action a, a boolean p representing whether the token `psk` is present in the whole handshake pattern, and a payload z, the semantics of a message token t is a function $\mathsf{process}\ r\ a\ p\ z\ t$ from a state σ and a term y to a new pair state, term.

The function process is defined as follows:

$$
\begin{aligned}
\mathsf{encrypthash}\ \sigma\ x\ \text{where}\ \sigma[k] = \mathsf{empty} &:= (\mathsf{mixhash}\ \sigma[n := \sigma[n]+1]\ x, x)\\
\mathsf{encrypthash}\ \sigma\ x &:= \text{let}\, c = \mathsf{aead}(\sigma[k], \sigma[n], \sigma[h], x)\ \text{in}\\
&\quad\ (\mathsf{mixhash}\ \sigma[n := \sigma[n]+1]\ c, c)\\
\mathsf{dh}\ r\ \mathtt{ee}\ \sigma &:= \sigma[re]^{\sigma[e]}\\
\mathsf{dh}\ r\ \mathtt{ss}\ \sigma &:= \sigma[rs]^{\sigma[s]}\\
\mathsf{dh}\ \text{Initiator}\ \mathtt{es}\ \sigma &:= \sigma[rs]^{\sigma[e]}\\
\mathsf{dh}\ \text{Responder}\ \mathtt{se}\ \sigma &:= \sigma[rs]^{\sigma[e]}\\
\mathsf{dh}\ \text{Initiator}\ \mathtt{se}\ \sigma &:= \sigma[re]^{\sigma[s]}\\
\mathsf{dh}\ \text{Responder}\ \mathtt{es}\ \sigma &:= \sigma[re]^{\sigma[s]}\\
\mathsf{process}\ r\ \text{Send false}\ z\ \mathtt{e}\ \sigma\ y &:= (\mathsf{mixhash}\ \sigma\ g^{\sigma[e]}, \langle y, g^{\sigma[e]}\rangle)\\
\mathsf{process}\ r\ \text{Send true}\ z\ \mathtt{e}\ \sigma\ y &:= \text{let}\, \sigma', y' = \mathsf{process}\ r\ \text{Send false}\ z\ \mathtt{e}\ \sigma\ y\\
&\quad\ \text{in}\ (\mathsf{mixkey}\, \sigma'\ g^{\sigma'[e]}, y')\\
\mathsf{process}\ r\ \text{Recv false}\ z\ \mathtt{e}\ \sigma\ y &:= \text{let}\, re = \mathsf{newvar}\ \text{in}\ (\mathsf{mixhash}\ \sigma[re := re]\ re, \langle y, re\rangle)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{process}\ r\ \text{Recv true}\ z\ \mathtt{e}\ \sigma\ y &:= \mathrm{let}\,\sigma', y' = \mathsf{process}\ r\ \text{Recv false}\ z\ \mathtt{e}\ \sigma\ y \\
&\quad \mathrm{in}\ (\mathsf{mixkey}\,\sigma'\ \sigma'[re], y') \\
\mathsf{process}\ r\ \text{Send}\ p\ z\ \mathtt{s}\ \sigma\ y &:= \mathrm{let}(\sigma', c) = \mathsf{encrypthash}\ \sigma\ g^{\sigma[s]} \\
&\quad \mathrm{in}\ (\sigma', \langle y, c\rangle) \\
\mathsf{process}\ r\ \text{Recv}\ p\ z\ \mathtt{s}\ \sigma\ y &:= \mathrm{let}\,rs = \mathsf{newvar};\ (\sigma, c) = \mathsf{encrypthash}\ \sigma\ rs \\
&\quad \mathrm{in}\ (\sigma[rs := rs], \langle y, c\rangle) \\
\mathsf{process}\ r\ a\ p\ z\ \mathtt{psk}\ \sigma\ y &:= (\mathsf{mixkey}\ (\mathsf{mixhash}\ \sigma\ \mathsf{kdf}_3(\langle\sigma[ck], \sigma[psk]], \sigma[psk]\rangle))\ \sigma[psk], y) \\
\mathsf{process}\ r\ a\ p\ z\ \mathtt{msg}\ \sigma\ y &:= \mathrm{let}\ (\sigma', c) = \mathsf{encrypthash}\ \sigma\ z \\
&\quad \mathrm{in}\ (\sigma', \langle y, c\rangle) \\
\mathsf{process}\ r\ a\ p\ z\ t\ \sigma\ y &:= (\mathsf{mixkey}\,\sigma\ (\mathsf{dh}\ r\ t\ \sigma), y)
\end{aligned}
$$

The term y represents the initially empty buffer on which each part of the soon-to-be sent message are stored. encrypthash is the analogue of the function `encryptAndHash` of the specification. dh $r\ t\ \sigma$ is the value of the Diffie Hellman term derived by role r from token r and state σ. Finally, newvar is a nullary function returning a new, unused variable.

Remark: Readers already familiar with the Noise specification will notice that we ignore all considerations depending on `HASHLEN`, effectively assuming that all terms, being symbolic, have infinite size.

As already mentioned, if σ is a state, then $\sigma[n]$ is a term, for which $+$ is not defined. $\sigma[n]$ is always a public value in $\mathbb{N}$, and we model $+1$ as an operation from public value to public value.

What if we try to compute $\mathsf{empty}^{\,x}$ or x^{empty} where x is a term? Then semantics of this pattern is ill-defined and the pattern will be made invalid later (by Definition 13, namely). Therefore, we do not consider this case.

Definition 10 (Semantics of message patterns)**.** Given a role r, action a and boolean p representing whether the token `psk` is present in the whole handshake pattern, the semantics of a message pattern ts is a function msg $r\ a\ p\ ts$ from a state σ and a term z (the payload) to a pair of a state and a term (the message to be sent or received).

The function msg is defined as follows:

$$
\begin{aligned}
\mathsf{msg}\ r\ a\ p\ ts\ z\ \sigma &:= \mathsf{msg}'\ r\ a\ p\ ts\ z\ \sigma\ \mathsf{empty} \\
\mathsf{msg}'\ r\ a\ p\ (t :: ts)\ z\ \sigma\ y &:= \mathsf{msg}'\ r\ a\ p\ ts\ z\ (\mathsf{process}\ r\ a\ p\ z\ t\ \sigma\ y) \\
\mathsf{msg}'\ r\ a\ p\ []\ z\ \sigma\ y &:= (\sigma, y)
\end{aligned}
$$

with the convention that $\langle \mathsf{empty}, x\rangle = x$.

Definition 11 (High-level semantics of a handshake pattern)**.** For a given role r, a prologue π, the value of all keys needed by this role in σ_I, and payloads $z_1, \ldots, z_n$ for each of the n message patterns in the handshake, let p be the boolean whether the handshake contains the token `psk`.

The high-level semantics of the whole handshake pattern is $(\sigma_0, (\sigma_1, m_1), \ldots, (\sigma_n, m_n))$ where

- σ_0 is the semantics of pre-message patterns under r, p, σ_I, and π;
- $(\sigma_{i+1}, m_{i+1}) = \mathsf{msg}\ r\ a\ p\ ts\ z_{i+1}\ \sigma_i$, with a being the action r completes with message $i+1$ and ts being the (decorated) message token list of message $i+1$.

By convention, unless otherwise noted, we will always manipulate semantics defined for a prologue π equal to the variable of the same name, needed keys equal to the variables of the corresponding name, and as payloads variables $z_1, \ldots, z_n$.

Example 7 (High-level semantics of NK). Building on Example 5, here is the high-level semantics of NK for the initiator:

$$\begin{aligned}
\sigma_0 &= \{e = e, rs = rs, h = h_f(\langle h_f(\langle \text{'NK'}, \text{'foo'}\rangle), rs\rangle), ck = \text{'NK'}, n := 0\} \\
\sigma_1 &= \{e = e, rs = rs, h = h_f(h_f(\langle \sigma_0[h], g^e\rangle), \mathsf{snd}\ \ m_1), ck = \mathsf{kdf}_1(\langle \sigma_0[ck], rs^e\rangle), \\
&\qquad k = \mathsf{kdf}_2(\langle \sigma_0[ck], rs^e\rangle), n = 1\} \\
m_1 &= \langle g^e, \mathsf{aead}(\sigma_1[k], 0, h_f(\langle \sigma_0[h], g^e\rangle), z_1)\rangle \\
\sigma_2 &= \{e = e, re = re, rs = rs, h = h_f(h_f(\langle \sigma_1[h], re\rangle), \mathsf{snd}\ \ m_2), \\
&\qquad ck = \mathsf{kdf}_1(\langle \sigma_1[ck], re^e\rangle), k = \mathsf{kdf}_2(\langle \sigma_1[ck], re^e\rangle), n = 1\} \\
m_2 &= \langle re, \mathsf{aead}(\sigma_2[k], 0, h_f(\langle \sigma_1[h], g^e\rangle), z_2)\rangle
\end{aligned}$$

This must be interpreted as follows: m_1 and m_2 are the successive messages which are exchanged during the handshake phase, and σ_i describes the state after the i-th message. Receiving a message m_1 means receiving any message, attempting pattern matching on it, and if the message does not match, then refuse the message and abort the handshake.

△

Notice that for any Noise handshake, the series $(\sigma_i[x])_i$ where x is a primitive key (that is, e, re, s, rs, or psk) is either constant, or starts with a repetition of empty and is then constant.

Definition 12 (Known key)**.** A key x is known after the i-th message if $\sigma_i[x] \neq \mathsf{empty}$.

The knowledge of an agent is the restriction of its state to primitive keys. For simplicity, asymmetric keys will be represented by the corresponding public key:

$$\hat{\sigma} = (g^{\sigma[e]}, \sigma[re], g^{\sigma[s]}, \sigma[rs], \sigma[psk])$$

Example 8. Still in NK, after the first message, the initiator knows his own ephemeral key e and his peer's static public key rs. After the second message, he still knows these keys, plus his peer's ephemeral public key re.

△

1.3.3 Pattern validity

The Noise specification defines criteria for so-called valid patterns. Since until here our definition for handshake patterns was a bit more permissive than the one in the specification, this definition of validity will compensate for this gap.

Definition 13 (Validity of a handshake)**.** A handshake pattern is valid if all of the following hold:

- Operations on empty are never needed to compute the semantics of this pattern, except the convention $\langle \mathsf{empty}, x\rangle = x$. Namely, one never has to compute x^{empty} or empty^x.
- Key tokens appear at most once per direction in the union of pre-messages and messages.
- Diffie-Hellman tokens appear at most once each in the union of all messages.
- If the initiator sends `msg` or reaches the end of the handshake after processing `se`, then he must also have processed `ee`.
- If the responder sends `msg` or reaches the end of the handshake after processing `es`, then he must also have processed `ee`.

- If the initiator sends `msg` or reaches the end of the handshake after processing `ss`, then he must also have processed `es`.
- If the responder sends `msg` or reaches the end of the handshake after processing `ss`, then he must also have processed `se`.

From now on, we will only consider valid handshakes.

1.4 Low-level semantics

The high-level semantics defined in Definition 11 are useful to reason about patterns but are not precise enough to fully describe a Noise-base protocol. We need a finer approach to actually describe what messages are exchanged *etc.* To this end, we will define *low-level semantics* of handshake patterns, namely a set of Tamarin rules suitable for encoding any protocol based on said pattern.

First, we will describe how Tamarin encodes protocols with multiset rewriting systems; then, we will describe how we can encode a Noise pattern in Tamarin with its low-level semantics.

1.4.1 Multiset rewriting systems

Tamarin models protocols with multiset rewriting systems. We continue recalling the necessary definitions as needed, again adapted from Schmidt [16].

The state of the system is described by a multiset of facts. Facts are built upon terms.

Definition 14 (Fact)**.** Given a fact signature Σ_F partitioned into *linear* and *persistent* facts symbols, a *fact* is of the form $F(t_1, \ldots, t_n)$ where $t_i \in T$ and $F \in \Sigma_F$ with arity n.

Example 9. From the term $g^{a \star b}$ and the fact symbol In we can form the fact $In(g^{a \star b})$. △

A rule can transform some facts matching a pattern into new facts. We must define formally what "matching" means here. This is done with the theory of term rewriting.

Definition 15 (Substitution)**.** A *substitution* is a function σ from the set of variables $\mathcal{X}$ to the term algebra such that there exists a finite set of variables $D \subseteq \mathcal{X}$ such that $\forall x \in \mathcal{X} \setminus D, \sigma(x) = x$.

We lift naturally this application from terms to terms with:

$$\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n)).$$

Example 10. Let σ be the substitution mapping a to 1 and other variables to themselves. $\sigma(In(g^{a \star b})) = In(g^{1 \star b})$.

△

Definition 16 (Matching)**.** A term t *matches* another term l if there exists a substitution σ such that $t =_{\mathcal{E}} \sigma(l)$, where equality is take modulo our equational theory $\mathcal{E}$.

Example 11. $In(g^b)$ matches $In(g^{a \star b})$ by the substitution of the previous example. When we say "inputting $g^{a \star b}$", we mean "inputting any term matching $g^{a \star b}$", for example g^b.

△

Definition 17 (Ground instances)**.** t is a *ground instance* of l if t is ground and matches l.

Example 12. $In(g^b)$ is not a ground instance of $In(g^{a \star b})$, but $In(g)$ is. △

Definition 18 (Labelled multiset rewriting rule)**.** A *rule* is composed of three sequences of facts l, a and r and is written as $l \xrightarrow{a} r$. Facts in a are called action facts.

A rule encodes a transformation of a system. Before, the state of the system contained l, we can apply the rule and replace it by r. a is only a label used to keep track of the fact that the rule was used.

Definition 19 (Multiset rewriting system)**.** A *multiset rewriting system* is defined by a set R of labelled multiset rewriting rules. R is required to contain the rule $[] \xrightarrow{[]} [Fr(x)]$ where Fr is a linear fact symbol of arity one which is also required to belong to the fact signature. This rule means that one can generate new, fresh values (here, x). The fact Fr may not appear on the right-hand side of any other rule (more on this later).

The semantics of the system are defined with a state evaluation S. S is a multiset of ground facts. This multiset has the following quirk: persistent facts do not see their multiplicity decremented by the multiset difference operation. They can only be added, and then never disappear from the set.

The small steps semantics of a multiset rewriting system are as follows: A state S' succeeds S if there exists a ground instance $l \xrightarrow{a} r$ of a rule in R such that $S' = S \setminus l \cup r$. Then one writes: $(S, l \xrightarrow{a} r, S') \in \text{step}(R)$.

Definition 20 (Execution, trace)**.** An execution is an alternating sequence of states and ground rule instances

$$S_0, l_1 \xrightarrow{a_1} r_1, S_1, l_2 \xrightarrow{a_2} r_2, S_2, \ldots, S_n$$

where

- S_0 is empty
- $\forall 1 \leq i \leq n, (S_{i-1}, r_i \xrightarrow{a_i} l_i, S_i) \in \text{step}(R)$
- For any fresh variable name x, there is only one instance of the fresh rule for x: $[] \xrightarrow{[]} [Fr(x)]$.

The *trace* corresponding to this execution is the sequence $(a_i)_{1 \leq i \leq n}$. The set of executions of R is denoted as $\text{exec}(R)$.

The last requirement means that a fresh value can only be generated once. This is why fresh values are usually used to model nonces.

1.4.2 Protocol modelling in Tamarin

Tamarin comes with a built-in set of rules dedicated to encode protocols in multiset rewriting systems. These rules model the Dolev Yao adversary.

$$Out(x) \to K(x) \tag{1.1}$$

$$K(x) \xrightarrow{K(x)} In(x) \tag{1.2}$$

$$Fr(x) \to K(x) \tag{1.3}$$

$$\forall f \in \Sigma, K(x_1, \ldots) \to K(f(x_1, \ldots)) \tag{1.4}$$

The fact $Out(x)$ means that x is sent over the network, $In(x)$ means that x is received from the network. $K(x)$ is a persistent fact denoting that the adversary *knows* x. The first two rules

thus mean that the Dolev Yao adversary *is* the network: everything sent is known to him, and he can send anything he can compute to anyone. The third rule means that the adversary can create his own fresh values, and the last one means that it can compute new values from terms known to him. Note however that he cannot deconstruct terms. From $K(\mathsf{aead}(k, n, ad, x))$ alone he cannot derive x. This is where the equational theory comes into play: if he knows additionally k and n then he can derive $K(\mathsf{decrypt}(k, n, \mathsf{aead}(k, n, ad, x))) =_{\mathcal{E}} K(x)$.

There are a few restrictions on what rules are accepted by Tamarin; most of them stem from common sense:

- In and Fr facts can only be on the left-hand side of a rule;
- Out facts can only be on the right-hand side of a rule;
- K facts can only appear in the built-in rules above;
- No variable may appear in the right-hand side if it does not also appear in the left-hand side;

and a few others like multiplication restriction. Note that the aforementioned built-in rules are exempt (and violate) these restrictions.

Example 13 (Simple Tamarin theory). An unauthenticated Diffie-Hellman key exchange can be modelled by the following theory (*i.e.*, set of rules): Create an initiator with its ephemeral key:

$$Fr(e) \to Initiator(e)$$

Create a responder similarly:

$$Fr(e) \to Responder(e)$$

The initiator sends his public key:

$$Initiator(e) \to Initiator_2(e), Out(g^e)$$

The responder receives it, sends his and derives the shared key:

$$Responder(e), In(re) \xrightarrow{Key_R(re^e)} Responder_2(re^e), Out(g^e) \tag{1.5}$$

The Initiator receives the key, and derives the shared key:

$$Initiator_2(e), In(re) \xrightarrow{Key_I(re^e)} Initiator_3(re^e) \tag{1.6}$$

There exists an execution of this system where the state ends up containing $Initiator_3(x)$ and $Responder_2(x)$ for some x: this protocol is executable, and can end up with agreement on the shared key. On the other hand, there exists an execution where the state ends up containing $Initiator_3(x)$ and $K(x)$. Therefore, this protocol provides no secrecy. △

Example 14 (Tamarin rules and pattern matching). Consider the rule

$$In(\langle x, x\rangle), Agent() \to Agent'(x)$$

It can be interpreted as "an agent receives $\langle x, x\rangle$". This rule contains a variable x. The set of possible usages of this rule are all its corresponding ground instances. Among them, we can find:

$$\begin{aligned} In(\langle 1, 1\rangle), Agent() &\to Agent'(1) \\ In(\langle g^s, g^s\rangle), Agent() &\to Agent'(g^s) \\ In(\mathsf{snd}(\langle 1, \langle 2, 2\rangle\rangle)), Agent() &\to Agent'(2) \end{aligned}$$

but not

$$In(3), Agent() \to Agent'(3).$$

This means that this agent can receive the messages $\langle 1, 1\rangle$, $\langle g^s, g^s\rangle$, $\mathsf{snd}(\langle 1, \langle 2, 2\rangle\rangle)$ but not 3 for example. This explains our previous formulation: inputting a term means pattern matching against it.

△

1.4.3 Tamarin modelling of Noise handshake patterns

A handshake pattern is not a full protocol. It is a cryptographic layer designed to convey payloads specified by an application layer while adding security more or less transparently. More precisely, it presents an API to the application layer. This API can be roughly approximated with the following actions:

- initialisation of new handshake with needed keys and a prologue
- send and receive a payload
- query some information like public key of the peer, or the session ID[2]

To model this, we used the design pattern called *channel*. A channel is a pair of facts which should be substituted to In and Out to convey messages while ensuring special properties. For example, here is a confidential channel:

$$Send(x) \to Recv(x)$$
$$In(x) \to Recv(x)$$

If a message is sent with $Out(x)$ then the adversary learns x because of the built-in rules $Out(x) \to K(x)$. If it is sent with $Send(x)$, this does not happen. The recipient can receive it with $Recv(x) \to \ldots$ instead of $In(x)$. Since a confidential channel is not authenticated, the adversary is given the possibility to forge arbitrary message through this channel (second rule). Other examples of channels are available in the Tamarin manual [18].

Note that a channel can limit the abilities of the Dolev Yao adversary; in our modelling we will take care to use channels which only limit the capabilities of the adversary when our threat model requires it.

Our Noise channel is more complicated because it is stateful. To distinguish between identical channels with different underlying states, each channel will be associated to a fresh value id. Of course, the channel for the initiator and the responder are distinct.

Example 15 (Noise channel for the initiator of NK). Again, we use the example of NK. The initiator needs a value for the prologue, its own ephemeral key and the static key of the peer. The application is required to provide them by creating a fact $Init\_Handshake\_Initiator(id, prologue, e, rs)$. Then to send the payload z_1 of the first message of the handshake, the application layer is required to create a fact $Handshake\_Snd\_1(id, z_1)$ instead of $Out(z_1)$. To receive the second one, the application layer would input $Handshake\_Snd\_2(id, z_2)$ instead of $In(z_2)$.

After switching from the handshake phase to the transport phase, the protocol become nearly stateless. The only state left is the sequence number of the payload. For modelisation convenience, it is provided by the application layer: $I\_Snd\_to\_R(id, 3, z_3)$ to send a payload of sequence number 3, and $I\_Recv\_from\_R(id, 4, z_4)$ to receive a payload of sequence number 4.

[2] see section 11.2 of the Noise specification, about `getHandshakeHash`

To provide information back from Noise to the application layer, some facts are also created during the handshake: $PeerEk\_Initiator(id, re)$ contains the ephemeral of the responder (intended for the initiator's application layer), for example.

The first few rules implementing this channel are:

```
/* Handshake initialisation with initial knowledge for Initiator */

rule Init_Handshake_Initiator:
 let hash0 = h(<'NK', prologue>)
     hash1 = h(<hash0, ('g'^rs)>) in
  [ Init_Handshake_Initiator(id, prologue, ~e, ('g'^rs)) ]
--[ State('initiator', hash1, 'NK', ~e, ('g'^rs))) ]->
  [ HandshakeState_Initiator_0(id, hash1, 'NK', ~e, ('g'^rs)) ]

/* Initiator is sending this message: -> e es */

rule Handshake_Initiator_Snd_1:
 let hash1 = h(<hash, ('g'^~e0)>)
     ck2 = kdf1(<ck, (('g'^rs)^~e0)>)
     k3 = kdf2(<ck, (('g'^rs)^~e0)>)
     c4 = aead(k3, '0', hash1, payload)
     hash5 = h(<hash1, c4>) in
  [
   HandshakeState_Initiator_0(id, hash, ck, ~e0, ('g'^rs)),
   Handshake_Snd_1(id, payload),
  ]
--[ State('initiator', hash5, ck2, k3, ~e0, ('g'^rs)) ]->
  [
   HandshakeState_Initiator_1(id, hash5, ck2, k3, ~e0, ('g'^rs)),
   Send(<('g'^~e0), c4>)
  ]
```

Note about Tamarin syntax: tilde denotes fresh variables and single quotes denote public values. The generator g for asymmetric keys is `'g'`.

The reader will recognise a family of facts $HandshakeState\_Initiator_i(id, \sigma_i)$ where σ_i are the successive states we defined in the high-level semantics of handshakes. For simplicity, empty values in σ_i are omitted.

Note that in the actual implementation, the ephemeral is not provided by the application layer at initialisation time as depicted here, but when the message containing the sent `e` token is built, but the result is the same.

△

Definition 21 (Learned key)**.** Considering a handshake and its high-level semantics for a role r with the notations of definition 11.

A key x ($x = re$, the peer's ephemeral public key, or $x = rs$, the peer's static public key) is learned by role r at message i if $\sigma_{i-1}[x] = \mathsf{empty}$ and $\sigma_i[x] \neq \mathsf{empty}$.

Definition 22 (Low-level semantics of handshake patterns)**.** The low-level semantics of a handshake pattern are defined as a set of multiset rewriting rules implementing one channel per role. For flexibility, this channel will not use `In` and `Out` but another generic channel `Recv` and `Send`.

The rules implementing the channel for this role are:

$$Init\_Handshake_r(id, \pi, \sigma_I) \xrightarrow{State(r,\sigma_{r,0})} HandshakeState_{r,0}(id, \sigma_0) \tag{1.7}$$

For all $1 \leq i \leq n$ where the i-th message is sent by r:

$$\begin{aligned} HandshakeState_{r,i-1}(id, \sigma_{i-1}),\ Handshake\_Snd_i(id, z_i) \xrightarrow{State(r,\sigma_{r,i})} \\ HandshakeState_{r,i}(id, \sigma_i), Send(c_i) \end{aligned} \tag{1.8}$$

or if the i-th message is received by r:

$$\begin{aligned} HandshakeState_{r,i-1}(id, \sigma_{i-1}), Recv(c_i) \xrightarrow{State(r,\sigma_{r,i})} \\ HandshakeState_{r,i}(id, \sigma_i),\ Handshake\_Recv_i(id, z_i) \end{aligned} \tag{1.9}$$

To the right-hand-side of this rule, we append $PeerEk_r(id, \sigma_i[re])$ if re is learned in this message and $PeerLtk_r(id, \sigma_i[rs])$ if rs is learned in this message by r.

To switch from handshake phase to transport phase:

$$\begin{aligned} HandshakeState_{r,n}(id, \sigma_n) \rightarrow \\ !Transport_r(id, kdf_1(\sigma_n[ck]), kdf_2(\sigma_n[ck])), Sessionid(id, \sigma_i[h]) \end{aligned} \tag{1.10}$$

And transport mode channel: For r = Initiator:

$$Snd_r(id, j, x), !Transport_r(id, k, k') \rightarrow Send(\langle j, \mathsf{aead}(k, j, 0, x)\rangle) \tag{1.11}$$

$$Recv(\langle j, \mathsf{aead}(k', j, 0, x)\rangle), !Transport_r(id, k, k') \rightarrow Recv_r(id, j, x) \tag{1.12}$$

For r = Responder:

$$Snd_r(id, j, x), !Transport_r(id, k', k) \rightarrow Send(\langle j, \mathsf{aead}(k, j, 0, x)\rangle) \tag{1.13}$$

$$Recv(\langle j, \mathsf{aead}(k', j, 0, x)\rangle), !Transport_r(id, k', k) \rightarrow Recv_r(id, j, x) \tag{1.14}$$

In all these rules, we replace the σ_i and c_i by their value given π and σ_I in the definition 11 of the high-level semantics of the handshake for a role r. Note that one builds a fact from terms, and σ_i is not a fact but a named tuple of facts; we use the convention that σ_i is inlined in the fact: $F(\sigma_i)$ means $F(\sigma[e], \sigma[re], \ldots, \sigma[n])$

Note that the Noise specification allows implementations to send transport mode messages either "bare" or accompanied by their sequence number [14, section 11.4]. We choose the latter option because it simplifies modelling and it discloses more information to the attacker.

Automatic generation The low-level semantics of a pattern can be used to model in Tamarin any protocol relying on this pattern with little modification. In the next chapter, we will build upon these rules to encode and verify various security properties of a generic application layer on top of Noise. This generic application layer is meant to be as permissive as possible with the adversary, so as not to lose generality, but the author of a new protocol based on a Noise pattern could want to take into account the specificities of his use case or his application layer. In this case he can use the low level semantics defined above, add the Tamarin rules modelling his own application layer and write the lemmas encoding the security properties he is interested in.

To this end, we wrote a generator able to convert a pattern in a set of rules very close to the low-level semantics described above. The differences are minor; for example, some labels are added which will make writing an executability lemma easier. Assume we designed a protocol on top of this variation of XX:

```
XXnew:
  -> e
  <- e, ee, s, es
  -> s, se, ss
```

We store this pattern in a file `XXnew.spec`. By executing `vacarme XXnew.spec > XXnew.spthy`, we obtain a file `XXnew.spthy` where we only need to append rules modelling our application layer without needing to worry about the details of Noise. It is possible that, to model some properties, adding labels to the generated rules is required, but these modifications are minor.

Chapter 2

Noise security properties

In the previous chapter, we explained how to encode a Noise-based protocol in Tamarin. In this chapter we explain which security properties we verify and how. First, we give some background on the properties and hyperproperties Tamarin can handle. Then, we define one Noise-based protocol per handshake pattern, and describe the security properties we will investigate for this protocol. Finally, we describe the notion of best threat model, which enables us to concisely describe in which of hundreds of threat models a security property holds.

2.1 Security specifications in Tamarin

Tamarin can handle two types of proofs: trace properties and hyperproperties. We first formally define trace properties, and then describe how Tamarin can approximate a certain class of hyperproperties.

2.1.1 Trace properties

The simpler type of statements are trace properties. For example, a very simplistic way to express secrecy could be: "for any execution of the protocol, for all terms labelled as secret in this execution, the adversary does not know this term". In the phrase "for any execution", the concept of execution will be expressed by the notion of *trace*. We will prove that for all traces, the trace satisfies a certain property in the form of a formula. This leads to the concept of *trace formula*.

This section is adapted and simplified from Schmidt [16].

Definition 23 (Trace formulas)**.** A trace formula is a first order formula over time variables and terms in the following atoms:

an action $f@i$

time point ordering $i \prec j$

time point equality $i \approx j$

term equality $t \approx t'$

true $\top$

false $\bot$

where f is an action fact, $t, t' \in \mathcal{T}$ are terms, $i, j \in \Theta$ are time points (we denote the set of time point variables as Θ). We can combine these atoms with the following operators: $\breve{\wedge}$ conjunction, $\breve{\vee}$ disjunction, $\neg$ negation, and the usual quantifiers $\exists$ and $\forall$. The set of trace formulas is denoted as Φ.

θ is a *valuation* if it maps each variable to $\mathbb{Q}$ for timepoints or ground terms for terms. If τ is a trace, we define inductively the satisfiability relation $\breve{\models}$ as follows:

$$
\begin{aligned}
(\tau, \theta) &\breve{\models} \top && \text{(true)} \\
f \in \tau[\theta(i)] &\Rightarrow (\tau, \theta) \breve{\models} f@i && \text{(action at timepoint)} \\
\theta(i) < \theta(j) &\Rightarrow (\tau, \theta) \breve{\models} i \prec j && \text{(timepoint ordering)} \\
\theta(i) = \theta(j) &\Rightarrow (\tau, \theta) \breve{\models} i \approx j && \text{(timepoint equality)} \\
\theta(t') =_E \theta(t') &\Rightarrow (\tau, \theta) \breve{\models} t \approx t' && \text{(term equality)} \\
\neg(\tau, \theta) \breve{\models} \phi &\Rightarrow (\tau, \theta) \breve{\models} \neg\phi && \text{(negation)} \\
\Big((\tau, \theta) \breve{\models} \phi\Big) \Rightarrow \Big((\tau, \theta) \breve{\models} \psi\Big) &\Rightarrow (\tau, \theta) \breve{\models} \phi \breve{\wedge} \psi && \text{(conjunction)} \\
(\tau, \theta) \breve{\models} \psi &\Rightarrow (\tau, \theta) \breve{\models} \phi \breve{\vee} \psi && \text{(disjunction)} \\
(\tau, \theta) \breve{\models} \phi &\Rightarrow (\tau, \theta) \breve{\models} \phi \breve{\vee} \psi && \text{(disjunction)} \\
\exists u. (\tau, \theta[x \mapsto u]) \breve{\models} \phi &\Rightarrow (\tau, \theta) \breve{\models} \exists x. \phi && \text{(existential quantification)} \\
\forall u. (\tau, \theta[x \mapsto u]) \breve{\models} \phi &\Rightarrow (\tau, \theta) \breve{\models} \forall x. \phi && \text{(universal quantification)}
\end{aligned}
$$

Note that in the last two constructors, u must be taken in the right domain: ground terms when x is a term and $\mathbb{Q}$ when x is a time variable. We also define implication as usual: $\phi \rightsquigarrow \psi$ means $\neg\phi \breve{\vee} \psi$, and equivalence $\phi \leftrightsquigarrow \psi$ means $\phi \rightsquigarrow \psi \wedge \psi \rightsquigarrow \phi$.

For a closed formula, we will write: $\tau \breve{\models} \phi$ if there exists a valuation θ such that $(\tau, \theta) \breve{\models} \phi$. $\tau \breve{\vdash} \phi$ denotes the syntactic statement that τ satisfies ϕ, and $\tau \breve{\models} \phi$ denotes the truth value of this statement.

Example 16. Most notations in trace formulas are rather intuitive, except maybe $f@t$. Writing $f@t$ can be loosely understood as the statement that, in the trace, the t-th rule has an action f. t is the "time" when f "happened".

In the theory of Example 13 we could encode a non-standard agreement property on the key as: "for all traces τ, $\tau \breve{\vdash} \forall x, t\colon Key_I(x)@t \rightsquigarrow \exists t'\colon Key_R(x)@t' \breve{\wedge} t' \prec t$". This can be interpreted as: each time the rule (1.6) is used, the rule (1.5) has been used *before*, and in both rules, re^e is the same value.

The trace denoting a normal execution of the protocol satisfies this formula, but the trace where the initiator's peer is the attacker does not because it contains a Key_I fact but not a Key_R one. Therefore, the above property does not hold.

Likewise, secrecy of the key from the point of view of the initiator can be expressed as: "for all traces τ, $\tau \breve{\vdash} \forall x, t\colon Key_I(x)@t \rightsquigarrow \neg(\exists t'\colon K(x)@t')$". This property does not hold either.

△

Definition 24 (Guarded formula)**.** A trace formula is guarded if all introductions of quantified variables are of the following form: $\forall t. Fact(\dots, t, \dots)@i \rightsquigarrow \phi$, or $\forall i. Fact(\dots)@i \rightsquigarrow \phi$, or $\exists t. Fact(\dots, t, \dots)@i \breve{\wedge} \phi$, or $\exists i. Fact(\dots)@i \breve{\wedge} \phi$.

Given a rewriting system R and a closed, guarded, trace formula ϕ, Tamarin can analyse the two following problems, called *lemmas*: $\exists \tau \in \text{trace}(\text{exec}(R)) : \tau \breve{\models} \phi$ and $\forall \tau \in \text{trace}(\text{exec}(R)) :$

$\tau \overset{\breve{}}{\models} \phi$. Tamarin is sound and complete, which means the answer is always correct, but it may not always terminate. We are mostly interested in the latter form of lemmas; let us introduce a notation for it.

Definition 25. We define the boolean $R \overset{\breve{}}{\|\models} \phi$ as $\forall \tau \in \text{trace}(\text{exec}(R)) : \tau \overset{\breve{}}{\models} \phi$. $R \overset{\breve{}}{\|\vdash} \phi$ is called a lemma and corresponds to the syntactic statement that $R \overset{\breve{}}{\|\models} \phi$ is true.

Example 17. For example, we can write statements like

$$\forall R, R', \text{exec}(R) \subseteq \text{exec}(R') \Rightarrow \forall \phi \in \Phi, R' \overset{\breve{}}{\|\models} \phi \Rightarrow R \overset{\breve{}}{\|\models} \phi.$$

△

2.1.2 Observational equivalence

Trace formulas are enough to encode trace properties, that is properties in the form "for all traces, this holds". Secrecy and agreement notably can be encoded in this way. However, anonymity is usually expressed as a hyperproperty, that is a statement relating several traces. If we want to prove that a role is anonymous in a protocol, we can express it by asserting that for all traces with Alice as this role, another valid trace exists where Bob plays this role instead and the adversary cannot tell the difference.

The category of statements: "for all trace of this set, there exists another trace of this second set such that these traces are indistinguishable to the adversary" is called observational equivalence. Tamarin is not able to deal with observational equivalence directly but instead can analyse a stronger property called dependency graph equivalence. Tamarin is thus correct but not complete with regard to observational equivalence.

We will need to reason about dependency graph equivalence, so we will present these notions slightly more formally in this paragraph. This is adapted from Basin et al. [4].

The terminology here is a bit different. The adversary is not the network, but the environment. Our goal is to prove that two sets of rules R_A and R_B called *systems* are indistinguishable as seen by this environment. For the sake of clarity, we will use Out_{sys} and In_{sys} instead of just Out and In for the system I/O and rules (1.1)–(1.4) are replaced by the set IF of *interface rules*:

$$Out_{sys}(x) \to In_{env}(x) \tag{2.1}$$
$$Out_{env}(x) \to In_{sys}(x) \tag{2.2}$$

which model how the environment and the system can communicate, and the set Env of *environment rules*:

$$In_{env}(x) \to K(x) \tag{2.3}$$
$$K(x) \xrightarrow{K(x)} Out_{env}(x) \tag{2.4}$$
$$Fr(x) \to K(x) \tag{2.5}$$
$$\forall f \in \Sigma, K(x_1, \dots) \to K(f(x_1, \dots)) \tag{2.6}$$
$$K(x), In_{env}(x) \to Out_{env}(\top) \tag{2.7}$$

The environment rules are the same as the usual Dolev Yao adversary, plus a new rule (2.7) which lets the adversary compare a value he knows and a value he has just received.

The adversary chooses how he instantiates rules. If we want to formalize that "the adversary cannot tell the difference", we need to keep track of how he derived the terms he knows. We do so with the concept of recipe.

Definition 26 (Recipe)**.** Let $l \xrightarrow{a} r$ be the ground instance of a rule y. We assume that l, r, a are ordered by some order relation. Let $F \in \Sigma_F$ be the name of a fact which appears at index k of r. The recipe for F is $\text{recipe}(F) = (y, k, \text{newvars}(F), (\text{recipe}(l_i))_i)$ where $\text{newvars}(F)$ denotes the list of variables appearing in F but not in l as pairs of variable name and instantiation.

Example 18. For the rule $r : [] \to B(\langle x, x\rangle)$ the recipe of B in the instance $[] \to B(\langle 1, 1\rangle)$ is $(r, 1, [(x, 1)], [])$. △

Similarly, the recipe of the whole rule instance is the list of the recipes of all its output facts, except for interface rules. These rules are supposed to be an opaque boundary between the environment and the system. So the recipe for instances of rule (2.1) is $((2.1), 1, [], z)$ where z is a new variable. We do similarly for (2.2). This effectively makes passing through the boundary oblivious to the original recipe of a fact: the adversary cannot see inside the system, only what the system explicitly outputs.

We now must expand the relation *steps* of a system to include the recipe. We write $S \xrightarrow[\rho]{a} S'$ when $(S, l \xrightarrow{a} r, S') \in \text{step}(R \cup IF \cup Env)$ and ρ is the recipe of $l \xrightarrow{a} r$.

Definition 27 (Observational equivalence)**.** Let R_A and R_B be two sets of multiset rewriting rules. Let $R'_A = R_A \cup IF \cup Env$ and $\mathcal{S}_A$ be the set of states of R'_A; similarly for B. The systems R_A and R_B are observationally equivalent for an environment Env if there is a relation $\mathcal{R}$ on $\mathcal{S}_A \times \mathcal{S}_B$ such that if $(S_A, S_B) \in R$, then if $S_A \xrightarrow[\rho]{a} S'_A$ then:

1. if $r \in IF \cup Env$ then there exists $S'_B \in \mathcal{S}_B$ and action facts a' such that $S_B \xrightarrow[\rho]{a'} S'_B$ and $(S'_A, S'_B) \in \mathcal{R}$.
2. else, there exists actions a_i and recipes ρ_i and $S'B \in \mathcal{S}_B$ such that $S_B \xrightarrow[\rho_1]{a_1} \dots \xrightarrow[\rho_n]{a_n} S'_B$ and $(S'_A, S'_B) \in \mathcal{R}$

and symmetrically if $S_B \xrightarrow[\rho]{a} S'_B$.

Colloquially, condition 1 expresses that the adversary is aware of his own choices of rules and recipes, so those must be the same in the two systems, whereas condition 2 expresses that what happens within the system is completely hidden to the adversary.

Then, anonymity of the initiator in a Noise-based protocol can be expressed as "The system where Alice is initiator is observationally equivalent to the system where Bob is the initiator".

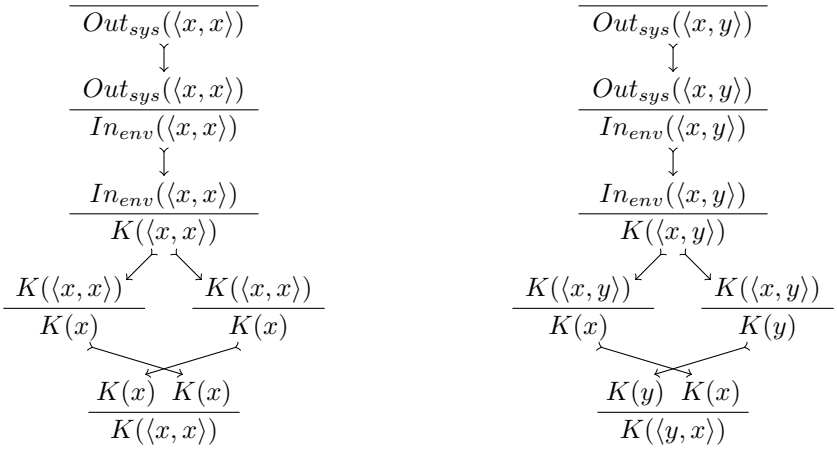
As mentioned above, Tamarin cannot analyse observational equivalence but only a strong property called dependency graph equivalence. In dependency graph equivalence, we restrict ourselves to so-called bi systems.

Definition 28 (Bi-system)**.** A bi-system is a set R of rules where terms can also be built with the function symbol diff of arity 2. The left-hand side system $\mathcal{L}(R)$ associated to R is the system identical to R but where $\text{diff}(x, y)$ is replaced by x and the right-hand side $\mathcal{R}(R)$ is the system identical to R but where $\text{diff}(x, y)$ is replaced by y.

Example 19. An example of bi-system is this one rule $[] \to Out_{sys}(\langle x, \text{diff}(x, y)\rangle)$. Its left system is $[] \to Out_{sys}(\langle x, x\rangle)$ and its right one is $[] \to Out_{sys}(\langle x, y\rangle)$

△

Definition 29 (Dependency graph)**.** A dependency graph dg is a pair (V, E) of a sequence of ground instances of rules V and a set of edges $E \subseteq \mathcal{P}(\mathbb{N}^2 \times \mathbb{N}^2)$ fulfilling the conditions below.



(a) Dependency graph of the left-hand side (b) Dependency graph of the right-hand side

Figure 2.1: An example dependency graphs of the left and right-hand side of the bi-system of Example 19.

Edges go from the conclusion fact of a rule to the premise fact from another. Premise (respectively conclusion) facts are thus designated by a pair of integers (i, u) such that i is the index of the rule in V and u is the index of the fact in the premises (respectively conclusion) of said rule. We write $(i, u) \rightarrowtail (j, v)$ if $((i, u), (j, v)) \in E$.

The conditions are:

- all edge indices must be in bounds
- if $(i, u) \rightarrowtail (j, v)$ then $i < j$ and the conclusion fact denoted by (i, u) is equal modulo $\mathcal{E}$ to the premise fact (j, v)
- every premise fact of dg has exactly one edge incoming
- every linear conclusion fact of dg has at most one outgoing edge

The set of dependency graphs of a system R is denoted as $\mathrm{dgs}(R)$.

Example 20. Figure 2.1 presents dependency graphs of the left and right-hand side of the bi-system of Example 19. △

Definition 30 (Mirror)**.** Let R be a bisystem, and Env an environment. Let $L = \mathcal{L}(R) \cup IF \cup Env$ and $R = \mathcal{R}(R) \cup IF \cup Env$. The mirrors of a dependency graph $dg = (V, E)$ over L are the set $\mathrm{mirrors}(dg) \subseteq \mathrm{dgs}(R)$ of the dependency graphs $dg' = (C', E')$ such that $|V| = |V'|$, $E = E'$ and for every rule index i in V, V_i is a ground instance of the same rule as V'_i with same instantiation of new variables.

The symmetric definition holds for the mirror of a right-hand-side dependency graph.

Example 21. The dependency graphes of Figure 2.1 are mirrors of one another. △

Definition 31 (Dependency graph equivalence)**.** A bi-system R is dependency graph equivalent given environment Env, denoted as $\mathcal{L}(R) \sim_{Env} \mathcal{R}(R)$, if all dependency graphs of the left side have at least one mirror and all new variable instantiations are present in at least one mirror, and the same holds for the right-hand side.

If the part of sentence "and all new variable instantiations are present in at least one mirror" is unclear to you, you can ignore it since it only applies when more variables are introduced in one side of a diff term (like $[] \to Foo(\text{diff}(1, x))$ for example) and we will not use such bi-systems.

Theorem 1 (Basin et al. [4])**.** *If the sides of a bi-system are dependency graph equivalent with environment Env then they are observationally equivalent for the same environment Env.*

Tamarin offers the possibility to restrict the behaviour of systems, and then investigate the dependency graph equivalence of such restricted systems. To formalize this, observe that a dependency graph encodes strictly more information than a trace (because of the recipes) so one can go from a dependency graph dg back to a trace denoted as $\tau(dg)$.

Definition 32 (Restriction)**.** A restriction is a closed trace formula. Let ψ be a trace formula.

The system R restricted to ψ, denoted as $R|_\psi$, is a system identical to R but with less dependency graphs:

$$\text{dgs}(R|_\psi) = \left\{ dg \in \text{dgs}(R) \mid \tau(dg) \breve{\vDash} \psi \right\}$$

A bi-system R restricted to ψ, also denoted as $R|_\psi$, is a bi-system whose left and right systems are restricted with ψ.

2.2 Security goals for Noise handshakes

Now we have the required background to define our security goals.

The goal of this work is to explore the security properties exhibited by any given handshake pattern. The main question is which security goals are worth investigating.

First, let us highlight once again the fact that, contrary to "usual" secure channel protocols, Noise allows sending payloads *before* the end of the handshake. These early payloads are opportunistically encrypted with a different key than after the handshake completes, or even not encrypted at all. See for instance Example 3. As a result, for a handshake pattern with two message patterns, we will need to verify the secrecy of the first four (two handshake messages, and two additional transport messages) exchanged payloads.

The Noise specification identifies the following types of properties:

Destination properties For a payload, whether confidentiality is guaranteed to its sender.

Source properties For a payload, whether its sender is correctly authenticated from the receiver's point of view.

Identity hiding properties For one role of a handshake, whether it is anonymous and/or unlinkable.

Notably, payload confidentiality from the point of view of the recipient are omitted. The specification gives a table which, for some selected handshakes, maps each payload to a level of destination guarantees (from 0, no guarantees, to 5, best guarantees) and sources guarantees (from 0 to 3); and likewise maps a selection of handshakes to their identity hiding level from 0 to 9.

Example 22 (Security properties from the specification). The second payload of KN has source level 0 (no guarantees) and destination level 3. Destination level 3 is described as follows:

> *Encryption to a known recipient, weak forward secrecy.* This payload is encrypted based on an ephemeral-ephemeral DH and also an ephemeral-static DH involving

> the recipient's static key pair. However, the binding between the recipient's alleged ephemeral public key and the recipient's static public key has not been verified by the sender, so the recipient's alleged ephemeral public key may have been forged by an active attacker. In this case, the attacker could later compromise the recipient's static private key to decrypt the payload.

From the point of view of the initiator, KN has identity hiding level 7 which is defined as follows:

> [The static public key of the initiator is] Not transmitted, but an active attacker who pretends to be the initiator without the initiator's static private key, then later learns a candidate for the initiator private key, can then check whether the candidate is correct.

Note how the description of these levels has more to do with the description of a possible attack than the description of what guarantees are provided and when they apply. △

Existing extensive analysis [11, 17] has focused on first formalizing a well-defined meaning for source and destination levels, and then machine-check which levels each payload satisfy. There are at least two downsides to this approach. First, since the specification overlooks properties like confidentiality to the recipient, the resulting formal analysis will also overlook them. Second, the descriptions for levels are arguably non-standard and non-trivial to formalize faithfully.

For these reasons, we chose to adopt a different approach. First, we will favour standard security goals: secrecy of payloads, and two of the four levels of agreement introduced by Lowe [12]. For identity hiding properties, we will focus on anonymity, but no standard definition really dominates. Then we will define a set of threat models to consider. For each of the security goals, we will determine in which threat models the property holds, and in which of them it does not hold. Only then we will compare our results to the levels in the specification. In the case of source and destination properties, we will establish that our results are more fine-grained than the specification, and that security levels can be described in terms of the aforementioned standard properties. This will enable us to provide a clearer and more unambiguous description for levels.

Example 23. Here is an example of property we consider: secrecy of the last handshake payload of XXpsk3 from the point of view of the initiator.

```
XXpsk3:
  -> e
  <- e, ee, s, es
  -> s, se, psk
```

The conclusions of our work on this specific property include, but are not limited to, the following. Let us consider, for the sake of simplicity, the set $\mathbf{A}$ of all threat models where the adversary is passive and keys (e, re, s, rs, psk) can be revealed or not. Let $t \in \mathbf{A}$. Secrecy of the payload holds under t unless t allows the psk to be revealed and either both e and re to be revealed as well, or one agent to have all their keys revealed. △

Note that a Noise-based protocol can be seen both as an authenticated key exchange and as a secure channel. Secrecy is preoccupied with payloads, and thus focuses on the secure channel aspect of Noise. But, as we will see later, agreement will be proved on both payloads and the hash $\sigma[h]$. It happens that $\sigma[h]$ is tightly coupled with encryption keys and used to authenticate encrypted messages, so agreement properties can be seen indirectly as properties of the authenticated key exchange as well.

2.2.1 Application layer protocol

As mentioned earlier, a Noise pattern does not represent a whole protocol. We need to provide an application layer to obtain a full protocol whose guarantees can be verified. This protocol will depend on what we need to prove: secrecy, agreement, or anonymity.

Definition 33 (Claim)**.** A security goal is secrecy, non-injective agreement, injective agreement or anonymity.

A claim is defined by a handshake pattern H, a goal and:

for secrecy: a role and a payload index: "secrecy of the i-th payload of H for role r"

for agreement: a payload index: "(non-)injective agreement on the i-th payload of H"

for anonymity: a role: "anonymity of role r in H"

The role of a claim is the role used in the above definition for anonymity and secrecy goals, or the role receiving the i-th payload for agreement claims.

An example of claim: secrecy of the 3$^{\text{rd}}$ payload of XX from the point of view of the initiator.

Our application layer depends on the claim we envision, but we avoid as much as possible to have it depend on the chosen threat model. This is implemented again with a number of channel-like rules.

Payloads The test payload for secrecy is a fresh value (*i.e.* a nonce). Other payloads are provided depending on the threat model via a fact `Payload(x)`. In some cases `x` will be a public value, otherwise it will be adversary-chosen. This is designed to be as permissive as possible with the adversary while still keeping meaningful results.

Prologue The prologue is also provided by a fact `Prologue(x)` depending on the threat model.

Needed keys Handshake patterns with `e` tokens in pre-messages are not meant to be executed in a stand-alone way, but when resuming from a former aborted handshake which had already generated an ephemeral key [14, section 10.4 for example]. This falls outside the scope of our analysis, although our Noise channel generator could be reused to model such resumable protocols. Therefore, we only focus on handshakes without `e` in pre-messages. Then, the keys needed to initialize the Noise channel are at most: a static key, provided by a fact[1] `!Pk(privatekey, 'g'^privatekey)`, an ephemeral key `Ek(privatekey, 'g'^privatekey)`, a pre-shared key `!Psk(key)` and the static public key of a peer `Peer(publickey)`. The origin of the `Peer` fact depends on the threat model. If the threat model assumes that the pre-message PKI is honest, then `Peer(publickey)` will be directly derived from an honest static key. On the other hand, if the pre-message PKI is not trusted, it will be an adversary-chosen value.

Actual layout The layout of the handshake phase is mandated by the specification. Messages must be alternating. However, transport mode messages can be exchanged in any order, and payload conveyed by different transport messages provide different guarantees.

Example 24 (Transport mode secrecy guarantees of KN). We are interested in the secrecy of various transport mode payloads for the responder of KN:

[1] The exclamation mark at the beginning of `!Pk` means that `Pk` is a persistent fact.

```
KN:
  -> s
  ...
  -> e
  <- e, ee, se
```

Intuitively (see Section 1.2) encryption keys of the second handshake message and all subsequent transport messages are derived from $\kappa = \langle g^{\sigma[e]\star\sigma[re]}, g^{\sigma[e]\star\sigma[rs]}\rangle$. But when sending the second handshake message, the responder has no proof that the initiator owns his private static key. The initiator could be an attacker using his own ephemeral key as first message. Secrecy guarantees are thus very weak. On the other hand, if the initiator has successfully sent at least one valid transport message to the responder, then the responder has a proof that this initiator could compute κ and is thus the rightful owner of their static key. For this reason, messages the responder sends after receiving this message can possibly have better security properties.

As a summary: transport messages sent by the responder before the first transport message from the initiator to the responder provide the same weak guarantees as the last handshake message, whereas later transport messages may provide stronger guarantees.

△

This reasoning helps us reduce the space of interesting message interleavings. Let r be the role sending the last (n-th) handshake message. Let κ be the tuple $(\sigma_n[k], \sigma_n[ck], \sigma_n[h])$. Observe that all subsequent transport messages and the payload of the last handshake are encrypted and authenticated with public, deterministic functions over κ. All transport messages sent by r before the first transport message sent by $\tilde{r}$ should thus have the same properties as the last handshake message as they are all encrypted by deterministic derivations from κ without receiving anything new from $\tilde{r}$. For this reason, we can dismiss them in our test protocol. Therefore, let the first transport message of our protocol be sent by $\tilde{r}$. $\tilde{r}$ has already received a valid message encrypted with κ, so receiving new ones does not bring any more knowledge. Therefore, further messages it could send would have the same guarantees. We can dismiss them as well. When receiving the message from $\tilde{r}$, r learns that $\tilde{r}$ knows κ, and thus this message may have different properties. Let our second transport message be from r to $\tilde{r}$. All further messages it may receive from $\tilde{r}$ do not bring in new information, and can be dismissed.

This leaves us with the following messages interleaving: $n + 2$ alternating messages, with n being the number of handshake messages.

The rules implementing the channel for a role r are:

$$\begin{aligned} &Fr(id), !Pk(s, g^s), !Peer(rs), Ek(e, g^e), !Psk(psk), Prologue(\pi) \\ &\qquad \xrightarrow{Create(r)} Init\_Handshake_r(id, \pi, s, rs, e, psk), State_{r,0}(id) \end{aligned} \tag{2.8}$$

omitting any unneeded key.

For all $1 \leq i \leq n$ where the i-th message is sent by r:

$$State_{r,i-1}(id), Payload(z_i) \to Handshake\_Snd_i(id, z_i), State_{r,i}(id) \tag{2.9}$$

or if the i-th message is received by r:

$$State_{r,i-1}(id), Handshake\_Recv_i(id, z_i) \to State_{r,i}(id) \tag{2.10}$$

Similarly, for all $n + 1 \leq i \leq n + 2$: if the i-th message is sent by r:

$$State_{r,i-1}(id), Payload(z_i) \to Snd_r(id, z_i), State_{r,i}(id) \tag{2.11}$$

or if the i-th message is received by r:

$$State_{r,i-1}(id), Recv_r(id, z_i) \to State_{r,i}(id) \tag{2.12}$$

The actual theory produced by our generator is slightly different because we do not generate ephemerals upfront but only on the message they are used, and because we store all known keys after message i inside $State_{r,i}$, and the session id in transport mode. The latter difference is needed to add some actions for lemmas, but makes formal description noticeably heavier.

2.2.2 Parametric theory

The parameters of our threat model can be summarized as follows:

- whether the adversary is passive or active;
- which honest private keys the adversary may have access to, and when;
- whether the pre-message PKI contains dishonestly generated keys;
- whether an agent assumes that his peer's static (or ephemeral) key is honestly generated.

Note that anonymity claims will only be tested on a very small subset of this search space, and with additional restrictions. A section will be dedicated to anonymity modelisation, so let us focus on secrecy and agreement claims.

Before we explain in full detail how our theory is parametrized, let us start with a simplified example.

Example 25 (Simplified active/passive parametrization). Let us focus on, say, secrecy of the third payload of XX for the initiator. Remember that the rules defining the low-level semantics of a handshake (see Section 1.4.3) send and receive network message through a channel `Send/Recv`. We define the channel as follows:

$$Send(x) \to Out(x), Receivable(x) \tag{2.13}$$

$$Receivable(x) \to Recv(x) \tag{2.14}$$

$$In(x) \xrightarrow{Active()} Recv(x) \tag{2.15}$$

With the rule (2.15), the adversary is active; without it he is passive.

Now, to encode secrecy, we add to the rule (2.9) or (2.11) sending the test payload z_3 an action fact $Secret(z_3)$. We consider the lemma

$$R \Vdash \forall x, t : Secret(x)@t \rightsquigarrow \neg(\exists t' : K(x)@t')$$

This lemma expresses that this payload is secret even if the adversary is active. On the other hand

$$R \Vdash \forall x, t : Secret(x)@t \rightsquigarrow \neg(\exists t' : K(x)@t') \veebar (\exists t' : Active()@t')$$

expresses that if the adversary does not use rule (2.15) *i.e.* is passive, then secrecy holds.

Following this template, we will encode secrecy and agreement lemmas as

$$ClaimFact(context) \rightsquigarrow Fulfilled(\dots) \veebar Invalid(\dots)$$

where $Invalid$ is a formula encoding what traces are excluded by the parameters of our current threat model. △

In the following we will define the actual "switches" like $Active()$ which enable us to restrict our lemmas to specific threat models.

Passive adversary As mentioned above, the channel described by rules (2.13)–(2.15) is used to distinguish if the adversary is passive. All network communications are done via this channel, *i.e.* `Recv` and `Send`. When the adversary is passive, he cannot replay or tamper with messages, but he can still drop or reorder them, since he is the network, and even non-hostile networks may drop and reorder messages.

The payload of each message (except the payload of the test message when the claim is a secrecy claim, see above) is chosen with a similar channel:

$$[] \rightarrow Payload(\$m) \tag{2.16}$$

$$In(m) \xrightarrow{Active()} Payload(m) \tag{2.17}$$

$\$values$ is the Tamarin syntax to denote public variables.

The fact that the active adversary can choose the (non secret) payload can be interpreted as the "worst case scenario" where, for some reason non encodable in the symbolic model, the adversary can influence the application layer. Another point of view is that it models the worst possible application layer protocol.

The prologue is chosen similarly:

$$[] \rightarrow Prologue(\text{`prologue'}) \tag{2.18}$$

$$In(m) \xrightarrow{Active()} Prologue(m) \tag{2.19}$$

Key generation Here are the rules for key generation:

$$Fr(s) \xrightarrow{HonestPk(g^s)} !Pk(s, g^s), Out(g^s) \tag{2.20}$$

$$!Pk(s, A) \xrightarrow{RevealLtk(A)} Out(s) \tag{2.21}$$

$$Fr(psk) \xrightarrow{HonestPsk(psk)} !Psk(psk) \tag{2.22}$$

$$!Psk(psk) \xrightarrow{RevealPsk(psk)} Out(psk) \tag{2.23}$$

$$Fr(e) \xrightarrow{HonestEk(g^e)} Ek(e), RevealableEk(e), Out(g^e) \tag{2.24}$$

$$RevealableEk(e) \xrightarrow{RevealEk(g^e)} Out(e) \tag{2.25}$$

Static keys (rules (2.20), (2.21)) are not tied to any form of identity because there is no other concept of identity in Noise protocols than the static key itself. From the perspective of the Noise layer, two different agents sharing the same static key behave exactly the same. Static keys can or cannot be revealed depending on whether the "switch" *RevealLtk* is allowed in traces.

Pre-shared symmetric keys (rules (2.22) and (2.23)) are not tied to an owner because the specification does not prohibit sharing one PSK among three or more partners for example. In this case, the purpose of a PSK is thus not to identify one's partner, but to restrict executability of the protocol to a set of (assumed honest) agents: those who know the PSK. When only two agents share the PSK, then an agent can indeed use it to identify their peer. Yet in this case static keys are arguably superfluous as one could just use NNpsk0 (a Diffie Hellman key exchange without static keys and authenticated by pre-shared key). This pattern would combine the simplicity of symmetric cryptography and the benefits of ephemeral keys: forward secrecy and lesser replayability. It seems more common to use PSK for a form of hardening for "the extremely paranoid" [6, Section V. B.]. Wireguard, for example, can optionally rely on a pre-shared key to "mitigate future advances in quantum computing" (*ibid.*). This discussion goes further on page 42. PSKs can or cannot be revealed depending on the "switch" *RevealPsk*.

Ephemeral keys (rules (2.24) and (2.25)) can be revealed before being used, for example before starting the session. This flexibility could be used to model (roughly) attacks based on flaws in a PRNG making ephemerals predictable, like [15]. Ephemeral key reveals can be enabled or disabled using the "switch" *RevealEk*.

Note that only fresh keys can be generated; although these keys can be revealed, the adversary cannot create arbitrary key facts with, for example, specially crafted content. This is what we mean when we say that keys are honestly generated. One important consequence is that the private keys of agents are always assumed to be honestly generated in our model. The reason for this choice is purely practical. Encoding Noise protocols for Tamarin leads to terms or the form b^a where a is the private key of an agent and b the public key of the peer. Tamarin needs to distinguish various cases like if b is g^1, or $g^{x^{-1}}$, or $g^{x \star y}$ and so on. The same holds for a, which leads to an exceedingly large number of cases, making proofs impossible to run to completion. Solving this problem led us to make these assumptions:

Honest private keys Honest agents only use honestly generated private keys, *i.e.* their private keys are always fresh values. In other words, we do not let the adversary tamper with the storage or generation mechanism of the keys. It is still possible to reveal these keys to the adversary—which could be a model for honestly but predictably generated keys.

Invalid point verification Agents' public keys are always based on the same generator g, and more importantly, when receiving a public key, agents check that this public key is indeed a valid group element, *i.e.* pattern match it against g^x. If this is not the case, then we assume that the agents ignore the message. This behaviour is not required by the specification: what happens in the presence of an invalid public key is implementation defined [14, Section 4.1]. Note that this assumption hides attacks relying on invalid points, which may be a real threat [2].

As a result, Diffie Hellman terms for honest agents are always of the form $g^{\sim x \star y}$ (the tilde denotes a fresh value) which keeps the number of cases in check and makes proofs manageable.

Key infrastructure Some handshakes use pre-messages. As said earlier, we only consider handshakes where only static keys are exchanged in pre-messages. There can be two different interpretations or uses of pre-messages: either they symbolize knowledge from a previous run of the protocol (for example in Noise Pipes, [14, section 10.4]), in which case only a moderate amount of trust can be put in them, or they symbolize a real (out-of-band) key exchange medium which may (or may not) be trusted. Here, trust mostly means that the adversary cannot convey dishonestly generated keys by this medium.

To distinguish these two cases we do not assume that pre-message static keys always come from the honest key facts. Instead, we define the *pre-message PKI* as follows:

$$!Pk(x, peer) \rightarrow Peer(peer) \tag{2.26}$$

$$In(peer) \xrightarrow{Active(), DishonestPki()} Peer(peer) \tag{2.27}$$

In case the second rule is used, we say that the pre-message PKI was dishonest.

Session identifier In section 11.2 of the Noise specification, an agent is allowed to use the last value of $\sigma[h]$ as a session identifier. For non PSK handshake, this value does not contain key

material (only public keys), and it is anyway heavily hashed. Since the specification does not forbid this value to be public, we give it to the adversary:

$$Sessionid(id, x) \to Out(x) \tag{2.28}$$

This will have consequences for anonymity modelling, see Section 2.2.6.

2.2.3 Secrecy and agreement claim encoding

With this in place, we can encode claims as suggested by Example 25. They will be parametrized by the formula (called *Invalid* in this example) which encodes our threat model.

Definition 34 (Secrecy lemma)**.** Let C be a secrecy claim for the role r and the i-th payload of the handshake pattern H. Let n be the number of message patterns of H. The set of rules $R(C)$ contains most of the aforementioned rules:

(1.1)–(1.4) attacker rules (they are built-in in Tamarin)

(1.7)–(1.14) low-level semantics of H for both roles

(2.8)–(2.12) application layer for both roles

(2.13)–(2.27) parametrization rules

(2.28) public session identifier

with the following modification:

- If $i \leq n$, then the i-th payload is a handshake payload. It sent by an instance of the rule (1.8). We add to this instance the action $Secret(z_i, \hat{\sigma}_i)$.
- Else, the i-th payload is sent by the rule (1.11). We add to this rule the action $Secret(z_i, \hat{\sigma}_n)$. This requires embedding the value $\hat{\sigma}_n$ in $!Transport_r$.
- In the application layer rule sending the test payload (an instance of (2.9) if $i \leq n$ or (2.11) otherwise) we replace $Payload(z_i)$ by $Fr(z_i)$.

Let $T \in \Phi^{\mathcal{T}^5 \times \Theta}$ be a mapping from a knowledge tuple and time point to a trace formula. The formula $\varphi(C, T)$ associated to C and T is

$$\forall x, \hat{\sigma}, t : Secret(x, \hat{\sigma})@t \rightsquigarrow T(\hat{\sigma}, t) \breve{\vee} \neg(\exists t' : K(x)@t')$$

Here, the purpose of T is the same as the formula *Invalid* in Example 25.

Example 26. Let us expand on the same situation as Example 25. Let C be the claim "secrecy of the third payload of XX for the initiator", $T_1 = (\hat{\sigma}, t) \mapsto \bot$ and $T_2 = (\hat{\sigma}, t) \mapsto \exists t' : Active()@t'$. The secrecy lemmas of example 25 are $R(C) \Vdash \varphi(C, T_1)$ for an active adversary and $R(C) \Vdash \varphi(C, T_2)$ for a passive adversary. △

Then we define similarly agreement lemmas:

Definition 35 (Agreement lemma)**.** Let C be an agreement claim for role r and payload i of the n-message handshake H. Let σ_x be the state of role x after message i if $i \leq n$ or the state of role x after message n if $i > n$ (see Definition 11). Let σ'_x be the state identical to σ_x except that keys which are unknown to x's peer are set to `empty`. Intuitively, $\hat{\sigma}'_x$ is the knowledge x expects to share with its peer. Also, if message patterns 1 to i do not contain the token `psk`, then we set $\sigma'_x[psk]$ to `empty`. This is because it is impossible on parties to agree on the pre-shared key before the first `psk` token.

The set of rules $R(C)$ contains all aforementioned rules:

(1.1)–(1.4) attacker rules (they are implicitly and automatically provided by Tamarin)

(1.7)–(1.14) low-level semantics of H for both roles

(2.8)–(2.12) application layer for both roles

(2.13)–(2.27) parametrization rules

(2.28) public session identifier

with the following modifications: add to the rule receiving this payload the action

$$Commit(\sigma'_r[e], \sigma'_r[s], \sigma'_r[re], \sigma'_r[rs], \sigma'_r[psk], \langle r, \tilde{r}, \langle \sigma'_r[h], z_i \rangle \rangle)$$

and to the rule sending it

$$Running(\sigma'_{\tilde{r}}[e], \sigma'_{\tilde{r}}[s], \sigma'_{\tilde{r}}[re], \sigma'_{\tilde{r}}[rs], \sigma'_{\tilde{r}}[psk], \langle \tilde{r}, r, \langle \sigma'_{\tilde{r}}[h], z_i \rangle \rangle)$$

Let $T \in \Phi^{\mathcal{T}^5 \times \Theta}$ be a mapping from a knowledge tuple and time point to a trace formula. The formula $\varphi(C, T)$ associated to C and T is

- for non-injective agreement:

$$\begin{aligned} &\forall m, \hat{\sigma}, t : Commit(\hat{\sigma}[e], \hat{\sigma}[s], \hat{\sigma}[re], \hat{\sigma}[rs], \hat{\sigma}[psk], \langle r, \tilde{r}, m \rangle)@t \\ &\quad \rightsquigarrow T(\hat{\sigma}, t) \veebar \exists t' : Running(\hat{\sigma}[re], \hat{\sigma}[rs], \hat{\sigma}[e], \hat{\sigma}[s], \hat{\sigma}[psk], \langle r, \tilde{r}, m \rangle)@t' \end{aligned}$$

- for injective agreement:

$$\begin{aligned} &\forall m, \hat{\sigma}, t : Commit(\hat{\sigma}[e], \hat{\sigma}[s], \hat{\sigma}[re], \hat{\sigma}[rs], \hat{\sigma}[psk], \langle r, \tilde{r}, m \rangle)@t \rightsquigarrow T(\hat{\sigma}, t) \\ &\quad \veebar (\exists t' : Running(\hat{\sigma}[re], \hat{\sigma}[rs], \hat{\sigma}[e], \hat{\sigma}[s], \hat{\sigma}[psk], \langle r, \tilde{r}, m \rangle)@t' \\ &\qquad \barwedge \forall e, s, re, rs, psk, t'' : Commit(e, s, re, rs, psk, \langle r, \tilde{r}, m \rangle)@t'' \rightsquigarrow t'' = t) \end{aligned}$$

Here, empty keys are implicitly omitted, and roles (which are not terms) are inserted as public constants `'initiator'` or `'responder'`.

These definitions are inspired from Lowe [12]. As agent name we use all keys this agent owns and his peer knows. The agreement bears on the payload, the pre-shared symmetric key, and the current hash. The reason to include the pre-shared key (PSK) in the agreement is that, for example in the case of NNpsk0 (a Diffie Hellman key exchange only authenticated by PSK), one can interpret the PSK as an identity. The reasons to include the hash in the agreed-upon term are multiple. First, agreement on the hash is expected to often hold since honest parties not agreeing on this value will not successfully execute the protocol. Then, the final hash of the handshake is exposed to the application layer, contrary to the other internal values k and ck of the state. The specification [14, section 11.2] allows parties to use this value as a (shared, unique) session identifier. For this reason, verifying that the protocol ensures agreement on this value makes sense. Finally, injective agreement solely on an attacker-chosen payload cannot hold, because an agent may send the same payload in two different, honest sessions. Therefore, we must include a unique value on which to agree: this hash is a perfect candidate for this.

Example 27 (Agreement lemma). We still consider XX:

```
XX:
  -> e
  <- e, ee, s, es
  -> s, se
```

Let C be the claim "Non-injective agreement of the second payload of XX". Implicitly, this claims belongs to the initiator, who receives the second payload. After the second message, the initiator knows all asymmetric keys. The responder knows all of them except the initiator's static key. In $\hat{\sigma}'_{\text{Initiator}}$, all keys are present except s and psk, which are empty. Therefore, they are omitted in the formulas. Let $T = (\hat{\sigma}, t) \mapsto \exists t' \colon \mathit{RevealLtk}(\hat{\sigma}[rs])@t'$. The formula $\varphi(C, T)$ is:

$$\begin{aligned}\forall m, e, re, rs, t \colon \mathit{Commit}(e, re, rs, \langle \texttt{'initiator'}, \texttt{'responder'}, m\rangle)@t \\ \rightsquigarrow (\exists t' \colon \mathit{RevealLtk}(rs)@t') \\ \dot{\vee} (\exists t' \colon \mathit{Running}(re, rs, e, \langle \texttt{'initiator'}, \texttt{'responder'}, m\rangle)@t')\end{aligned}$$

△

Anonymity claims cannot be encoded this way, because they are hyperproperties. We will formalize them later, in Section 2.2.6.

2.2.4 Parametric threat model

In Section 2.2.2, we defined a number of action facts like $\mathit{Active}()$ and $\mathit{DishonestPki}()$ which can mark whether the adversary compromised the pre-messages PKI, was active or passive, *etc.* This enables us to easily encode various threat models.

As suggested in Example 26, our threat models can be expressed with a family of suitably chosen formulas like $\hat{\sigma}, t \mapsto \exists t' : \mathit{Active}()@t'$. These formulas are cumbersome to manipulate, so we will hide them behind symbols called *exclusions*. For example, the exclusion for the above formula is called *active*, and we will write $[\![\mathit{active}]\!]_t = \hat{\sigma}, t \mapsto \exists t' : \mathit{Active}()@t'$. The set of threat models we will consider is the transitive closure of conjunction and disjunction of these exclusions.

Definition 36 (Exclusion)**.** The set of exclusions is the following set of symbols:

$$E = \{\mathit{active}, D_{pki}, D_{rs}, D_{re}, R_{psk}, R_{rs}, R_s, R_{rs}, R_e, R^{<}_{psk}, R^{<}_{rs}, R^{<}_{s}, R^{<}_{rs}, R^{<}_{e}\}$$

active denotes that the adversary is active. R is for "reveal": with R_{psk} the PSK is revealed, with $R^{<}_{psk}$ it is revealed before the claim of secrecy or agreement. D is for dishonest: D_{pki} means that the pre-message PKI contains a dishonestly generated key; D_{rs} means that my peer's static public key is dishonestly generated.

Definition 37 (Threat model)**.** For $F \subseteq E$, we denote its transitive closure by conjunction $\dot{\wedge}$ and disjunction $\dot{\vee}$ as $F^\star$. As usual, the empty disjunction is denoted as $\bot$ and the empty conjunction as $\top$.

A *threat model* is an element of $E^\star$.

For example, secrecy of a payload in the threat model $R_{rs} \dot{\vee} R_s \dot{\vee} \mathit{active}$ can be read as "secrecy holds unless my static key is revealed, or my peer's static key is revealed, or the adversary is active".

The definition of secrecy and agreement claims requires that we encore the threat model in an appropriate formula T yielding a suitable $\varphi(C, T)$, just like example 25.

Definition 38 (Trace semantics of exclusions)**.** For $x \in E^\star$, we inductively define the *trace semantics* of x, noted as $[\![x]\!]_t$, as the element of $\Phi^{\mathcal{T}^5 \times \Theta}$ (a function from a knowledge tuple and

time variable to a trace formula) such that:

$[\![\top]\!]_t = (\hat{\sigma}, t) \mapsto \top$	True
$[\![\bot]\!]_t = (\hat{\sigma}, t) \mapsto \bot$	False
$[\![active]\!]_t = (\hat{\sigma}, t) \mapsto \exists t' : Active()@t'$	Active adversary
$[\![D_{pki}]\!]_t = (\hat{\sigma}, t) \mapsto \exists t' : DishonestPki()@t'$	Dishonest pre-message PKI
$[\![D_{re}]\!]_t = (\hat{\sigma}, t) \mapsto \neg\exists t' : HonestEk(\hat{\sigma}[re])@t'$	My peer's ephemeral is dishonestly generated
$[\![D_{rs}]\!]_t = (\hat{\sigma}, t) \mapsto \neg\exists t' : HonestPk(\hat{\sigma}[rs])@t'$	My peer's static is dishonestly generated
$[\![R_e]\!]_t = (\hat{\sigma}, t) \mapsto \exists t' : RevealEk(\hat{\sigma}[e])@t'$	My ephemeral is revealed
$[\![R_{re}]\!]_t = (\hat{\sigma}, t) \mapsto \exists t' : RevealEk(\hat{\sigma}[re])@t'$	My peer's ephemeral is revealed
$[\![R_s]\!]_t = (\hat{\sigma}, t) \mapsto \exists t' : RevealLtk(\hat{\sigma}[s])@t'$	My static is revealed
$[\![R_{rs}]\!]_t = (\hat{\sigma}, t) \mapsto \exists t' : RevealLtk(\hat{\sigma}[rs])@t'$	My peer's static is revealed
$[\![R_{psk}]\!]_t = (\hat{\sigma}, t) \mapsto \exists t' : RevealPsk(\hat{\sigma}[psk])@t'$	The pre-shared key of this handshake is revealed
$[\![R_e^<]\!]_t = (\hat{\sigma}, t) \mapsto \exists t' : RevealEk(\hat{\sigma}[e])@t' \barwedge t' \prec t$	My ephemeral is revealed before the claim
$[\![R_{re}^<]\!]_t = (\hat{\sigma}, t) \mapsto \exists t' : RevealEk(\hat{\sigma}[re])@t' \barwedge t' \prec t$	My peer's ephemeral is revealed before the claim
$[\![R_s^<]\!]_t = (\hat{\sigma}, t) \mapsto \exists t' : RevealLtk(\hat{\sigma}[s])@t' \barwedge t' \prec t$	My static is revealed before the claim
$[\![R_{rs}^<]\!]_t = (\hat{\sigma}, t) \mapsto \exists t' : RevealLtk(\hat{\sigma}[rs])@t' \barwedge t' \prec t$	My peer's static is revealed before the claim
$[\![R_{psk}^<]\!]_t = (\hat{\sigma}, t) \mapsto \exists t' : RevealPsk(\hat{\sigma}[psk])@t' \barwedge t' \prec t$	The pre-shared key is revealed before the claim

and for $x, y \in E^\star$, we lift conjunction and disjunction to trace formulas:

$$[\![x \mathbin{\dot\wedge} y]\!]_t = \hat{\sigma}, t \mapsto [\![x]\!]_t \barwedge [\![y]\!]_t$$
$$[\![x \mathbin{\dot\vee} y]\!]_t = \hat{\sigma}, t \mapsto [\![x]\!]_t \barvee [\![y]\!]_t$$

Note that the t in $[\![\cdot]\!]_t$ is part of the notation and not a variable. It stands for "trace" in "trace semantics", as we will later introduce *diff semantics* noted as $[\![\cdot]\!]_d$.

Let us justify our choices of exclusions. *active* enables us to model a passive adversary—or rather, it excludes active adversaries. To build upon Examples 25 and 26, if C is a secrecy or agreement claim, then $R(C) \Vdash \varphi\,(C, [\![active]\!]_t)$ encodes a passive adversary while $R(C) \Vdash \varphi\,(C, [\![\bot]\!]_t)$ has an active adversary.

Similarly, D_{pki} models whether pre-messages provide keys by a trustable infrastructure or not, as explained when we introduced rule (2.27). The family of R_x exclusions, where x is a key, enables us to allow or disallow key reveals. For example, let C be a secrecy claim for XX.

```
XX:
  -> e
  <- e, ee, s, es
  -> s, se
```

Both agents have an ephemeral key and a static key. A naive threat model to encode "regular" secrecy is that the test payload is secret unless one key at least is revealed. This is encoded by the lemma

$$R(C) \Vdash \varphi\,(C, [\![R_e \mathbin{\dot\vee} R_{re} \mathbin{\dot\vee} R_s \mathbin{\dot\vee} R_{rs}]\!]_t) \tag{2.29}$$

A notable usage of R_{psk} is to model dummy pre-shared keys. This usage is described in section 11.1 of the Noise specification [14]. For example, Wireguard [6] uses the pattern IKpsk2 where parties need a pre-shared symmetric key. If parties do not have access to such a key, then

they may replace it with the value 0. This can be modelled by allowing pre-shared key reveals. Since pre-shared keys are not tied to an identity, all agents opting out can use the same revealed pre-shared key. The actual value will not be 0, but will be the same for all agents who opted out.

The family of $R_x^<$ enables us to encode forward secrecy. In the above example, perfect forward secrecy can be encoded by asserting secrecy unless an ephemeral key is revealed (at any time) or a static key is revealed before the secrecy claim, that is before the test payload is sent or received. This gives us the lemma:

$$R(C) \Vdash\!\!\!\!\breve{} \; \varphi\left(C, [\![R_e \mathbin{\dot\vee} R_{re} \mathbin{\dot\vee} R_s^< \mathbin{\dot\vee} R_{rs}^<]\!]_t\right)$$

Likewise, key compromise impersonation resistance (KCI) can be expressed within our set of threat models. Let C' be an agreement claim on XX from the point of view of the Initiator. Key-compromise impersonation resistant agreement can be defined as agreement unless an ephemeral key or the responder's static is revealed, that is, agreement must hold even though the static of initiator is revealed. This can be encoded as:

$$R(C') \Vdash\!\!\!\!\breve{} \; \varphi\left(C', [\![R_e \mathbin{\dot\vee} R_{re} \mathbin{\dot\vee} R_{rs}]\!]_t\right)$$

Now we still have to explain the purpose of D_{re} and D_{rs}. All handshakes which do not involve pre-messages are subject to an obvious man in the middle attack where Alice speaks to the protocol abiding adversary and this adversary relays the messages to Bob. This is, to some extent, expected. Because of this, lemma (2.29) is false, and more generally, all similar secrecy lemmas will be false as well. Nevertheless, these protocols still provide some amount of secrecy: one cannot be sure not to speak to the adversary, but if it is not the case, then no-one else can decrypt the conversation. Therefore, if we assume that the agent claiming secrecy does not belong to the adversary, then secrecy might hold. This threat model is weaker, but by considering it, we enrich the expressiveness of our results. This is what D_{rs} and D_{re} are for. These exclusions express that we assume that the (ephemeral or static) key of our peer was honestly generated (*i.e.* was generated by rule (2.20) or (2.24)) which implies that it does not belong to the adversary.

The broader question is "what guarantees to an agent that his peer is not the adversary ?". At the abstract level of a Noise pattern, an agent is defined by an ephemeral key, a static key and a PSK. The answer to the question is thus "because his peer's PSK does not belong to the adversary", or "because his peer's ephemeral does not belong to the adversary", or "because his peer's static does not belong to the adversary", and combinations thereof. If we elaborate on this, we can come up with the following scenarios:

PSK reliance The agent knows he does not speak to the adversary because his peer proves knowledge of the PSK, and the PSK is assumed not to be revealed. This only applies to handshakes with PSK. The corresponding threat models are of the form $T = R_{psk}^< \mathbin{\dot\vee} x, x \in E^\star$.

PKI reliance The agent knows he does not speak to the adversary because he received his peer's static key from the pre-message PKI, and we assume the adversary cannot register keys there and that the peer's static is not revealed. This is the "traditional" use of a PKI. The corresponding threat models are in the form $T = D_{pki} \mathbin{\dot\vee} R_{rs}^< \mathbin{\dot\vee} x, x \in E^\star$. This only applies when an agent receives their peer's static key in pre-messages.

TOFU The peer's static key could be the adversary's. If this is not the case, then the property we consider holds, though. Besides, if the agent re-establishes a later session with this same

peer (read: same static key) he will know it is the same agent and that it should still be honest. Hence the name TOFU (trust on first use). It is supposed to protect agents against man in the middle attacks attempted *after a first, unattacked session*. The corresponding threat models are in the form $T = D_{rs} \dot\vee R_{rs} \dot\vee x, x \in E^\star$. This only applies when the peer uses a static key.

Anonymous The peer has no static key, so the agent has no reliable way to distinguish a session with the protocol abiding attacker from a session with an honest peer. The best guarantees one can hope in this situation is "if the peer is honest (and their key non revealed) then…". This is modelled as $T = D_{re} \dot\vee R^<_{re} \dot\vee x, x \in E^\star$. This gives very weak guaranties. In the example of secrecy, the test payload is only known to the agent and their peer, but this peer could be anyone, including the attacker. For example, NN (unauthenticated Diffie Hellman) guarantees secrecy under $T = D_{re} \dot\vee R_{re} \dot\vee R_e$, despite this flaw. Still, this is better than plain text.

We distinguish D_{rs} and $R^<_{rs}$ because the former implies an active adversary when the latter does not. Besides, a revealed honest private key is always a nonce, whereas a dishonestly generated key can contain inverses, and thus enable the adversary to do more. There is no D_{psk} because for symmetric keys, the internal structure of the key does not matter. An attacker can do exactly as much with a revealed symmetric key as with an arbitrarily generated key.

Example 28. With this set of exclusions, we can encode some form of secrecy for a TOFU peer as follows:

$$T = R_e \dot\vee R_{re} \dot\vee R_s \dot\vee R_{rs} \dot\vee D_{rs}$$

which can be read as "secrecy unless the agent is speaking directly to the adversary (someone with the adversary's static) or any asymmetric key is revealed". It happens that contrary to (2.29), for C secrecy of the fifth payload of XX, the lemma $R(C) \Vdash \varphi(C, [\![T]\!]_t)$ holds. △

Definition 39 (Notation)**.** Let C be a secrecy or agreement claim, and T a threat model. We say that T guarantees C, denoted as $T \vDash C$, if $R(C) \Vdash\!\!\!\!\breve{}\;\; \varphi(C, [\![T]\!]_t)$. $T \vdash C$ is the syntactic statement $R(C) \Vdash\!\!\!\!\breve{}\;\; \varphi(C, [\![T]\!]_t)$ whose truth value is $T \vDash C$.

The attentive reader may have noticed that the threat models $T_1 = R_e$ (the local ephemeral key is revealed) and $T_2 = R^<_e \dot\vee R_e$ (the local ephemeral key is revealed before the claim or at any time) should intuitively be the same. We will give a formal meaning to this sentence.

Definition 40. Let $F \subseteq E$. We denote as F° the set of boolean formulae over the set of variables F with $\dot\wedge$ as conjunction, $\dot\vee$ as disjunction and $\neg$ as negation. We also denote usual entailment of boolean formulae as $\models$. For $A, B \in F^\circ$, we write $A \twoheadrightarrow B$ for $\neg A \dot\vee B$ and $A \twoheadleftarrow\!\!\!\!\twoheadrightarrow B$ for $(A \twoheadrightarrow B) \dot\wedge (B \twoheadrightarrow A)$.

Note that $F^\star \subseteq F^\circ$. The difference between the two is that F° contains negation. For example, we have $\models D_{rs} \dot\wedge R_e \twoheadleftarrow\!\!\!\!\twoheadrightarrow R_e \dot\wedge D_{rs}$, $\models R_e \twoheadrightarrow R_e \dot\wedge R_s$, and $R^<_e \twoheadrightarrow R_e \models R_e \dot\wedge R^<_e \twoheadleftarrow\!\!\!\!\twoheadrightarrow R_e$.

Definition 41 (Subsumption)**.** Let $A, B \in E^\circ$. We say that A subsumes B (denoted as $A \rightharpoonup B$) if $\mathcal{K} \models A \twoheadrightarrow B$ where

$$\begin{aligned}\mathcal{K} = (R^<_e \twoheadrightarrow R_e) \dot\wedge (R^<_{re} \twoheadrightarrow R_{re}) \dot\wedge (R^<_s \twoheadrightarrow R_s) \dot\wedge (R^<_{rs} \twoheadrightarrow R_{rs}) \dot\wedge (R^<_{psk} \twoheadrightarrow R_{psk}) \dot\wedge \\ (D_{rs} \twoheadrightarrow \mathit{active}) \dot\wedge (D_{re} \twoheadrightarrow \mathit{active}) \dot\wedge (D_{pki} \twoheadrightarrow \mathit{active}) \dot\wedge \\ (D_{rs} \twoheadrightarrow D_{pki})\end{aligned}$$

We write $A \rightleftharpoons B$ when $A \rightharpoonup B$ and $B \rightharpoonup A$.

This definition expresses the intuitive fact that $R_e^<$ (the agent's ephemeral is revealed before the claim) implies R_e (the agent's ephemeral is revealed, at any time), for example. It also encompasses "normal" implication on $E^\star$ viewed as a set of boolean formulas over positive literals of E: $active \rightharpoonup active \dot{\vee} R_e$ and $R_e \dot{\wedge} R_s \rightharpoonup R_e$.

Definition 42 (Distinct threat models)**.** Let $F \subseteq E$. We denote as $\bar{F}$ quotient of $F^\star$ by $\rightleftharpoons$.

Intuitively, $\bar{F}$ is the set of *distinct* threat models one can obtain by combinations of the exclusions in E. This enables us to say that $R_e \dot{\wedge} R_e^<$ and $R_e^<$ are the same in $\bar{E}$ and thus match intuition.

Lemma 1. *Let $F \subseteq E$. $\rightharpoonup$ is well-defined on $\bar{F}$ and it induces a partial order on $\bar{F}$.*

Proof. $\rightharpoonup$ is reflexive and transitive on $F^\star$. Therefore, it is a preorder. This proves that the construction of $\bar{F}$ leads to a partial order. □

For this reason, we will not introduce a notation to distinguish, say, R_e and the equivalence class of R_e.

Remark (Number of distinct threat models). Let $E_0 = \{R_s, R_{rs}, R_{psk}, R_{re}, R_e, D_{rs}, D_{re}\}$. It is easy to see that on $E_0^\star$, subsumption and implication coincide, *i.e.*

$$\forall T, T' \in E_0^\star, (T \rightleftharpoons T') \Leftrightarrow (\Vdash T \leftrightarrow T').$$

Therefore $\bar{E}_0$ is in bijection with the quotient of $E_0^\star$ by standard boolean formula equivalence. This quotient is in fact the free distributive lattice generated by E_0, and its cardinal is the 7^{th} number of Dedekind $M(7) = 2414682040998$ [1]. We can deduce from this that $|\bar{E}| \geq |\bar{E}_0| = M(7)$. This justifies our claim that we consider up to 10^{12} distinct threat models. △

2.2.5 Real Setup

Some exclusions rely on channels: *active* and D_{pki}. We had problems finding a proof heuristic which performs well on all claims and all handshake in the presence of these channels. To perform well here means mostly to prevent cycles and thus to ensure termination.

We therefore decided to get rid of these channels by splitting theories in two parts. For a claim C we have a theory $R_1(C)$ where we remove the $Active()$ action fact on rule (2.15) and get rid on the (now redundant) rules (2.14), (2.16), and (2.18). This theory always models an active adversary. Then we have a theory $R_2(C)$ where we remove all rules with an action $Active()$. This theory always models a passive adversary. Then we observe that

$$\forall T \in (E \setminus \{active\})^\star, R(C) \mid\!\breve{=} \varphi(C, [\![T]\!]_t) \Leftrightarrow R_2(C) \mid\!\breve{=} \varphi(C, [\![T]\!]_t)$$

and

$$\forall T \in (E \setminus \{active\})^\star, R(C) \mid\!\breve{=} \varphi(C, [\![T \dot{\vee} active]\!]_t) \Leftrightarrow R_1(C) \mid\!\breve{=} \varphi(C, [\![T]\!]_t)$$

This observation enables us to operate on simplified theories $R_1(C)$ and $R_2(C)$ instead of $R(C)$. We also repeat the same operation to get rid of $DishonestPki()$, and then we can actually write heuristics which make proofs terminate in a reasonable time.

This is an implementation detail, and for the remainder of this thesis, we will only deal with lemmas in the form $R(C) \mid\!\breve{\vdash} \ldots$.

2.2.6 Anonymity claims

In this section, we explain how we encode the claim "Anonymity of role r in the handshake pattern H."

From a high level, anonymity proofs deal with three types of agents: Alices, Bobs and Charlies. There can be several of each, but let us ignore this for a moment. The situation is as follows: diff(Alice, Bob) talks to Charlie. The task of the attacker is to determine which of Alice and Bob was present—effectively, which of Alice's or Bob's public key is the one used in the transcript. Charlie is only present because we need an honest peer to execute the protocol with Alice or Bob. diff(Alice, Bob) can only act as role r, but Charlie can be both roles, allowing "normal sessions".

Key facts The different key facts are used as follows:

- $!PeerPk(s, g^s)$ exists only if s Charlie's static key;
- $!TestPk(s, g^s)$ exists if s is owned by Charlie and diff(Alice, Bob);
- $!Pk(s, g^s)$ exists if s is owned by Alice, Bob, and Charlie, *i.e.* all honest parties.

Here is how one creates diff(Alice, Bob):

$$\begin{aligned} Fr(a), Fr(b) \to\ & !TestPk(\mathrm{diff}(a,b), g^{\mathrm{diff}(a,b)}), Out_{sys}(g^a), \\ & Out_{sys}(g^b), !TestAlternatives(a+b) \end{aligned} \tag{2.30}$$

$$!TestAlternatives(a+b) \to !Pk(a, g^a), !Pk(b, g^b) \tag{2.31}$$

$$!TestAlternatives(a+b) \xrightarrow{RevealTest()} Out_{sys}(a), Out_{sys}(b) \tag{2.32}$$

and Charlie:

$$Fr(c) \to !TestPk(g, g^c), !PeerPk(g, g^c), !Pk(c, g^c) \tag{2.33}$$

$$!PeerPk(c, g^c) \xrightarrow{RevealPeer()} Out_{sys}(c) \tag{2.34}$$

Rules (2.32) and (2.34) are only needed if we want to reveal diff(Alice, Bob) and Charlie respectively.

The notation $a + b$ denotes the set containing a and b, which is a commutative construction. For our purpose, this can be modelled as adding the equation $a + b = b + a$ to our equational theory. The indirection through the fact $!TestAlternatives$ is only there to prevent spurious falsifications (remember, Tamarin is correct but not complete with observational equivalence proofs).

Exclusions The observational equivalence mode of Tamarin has a number of limitations. It is considerably more computationally intensive, does not support induction, *etc.* For this reason, we will restrict the set of threat models we consider to the closure $E_a^\star$ of $E_a = \{R_s, R_{rs}, R_{psk}, active\}$. Then we need to give new, adapted semantics to exclusions in E_a.

Definition 43 (Diff semantics of exclusions)**.** For $x \in E_a^\star$, we inductively define the *diff semantics*

of x, noted as $[\![x]\!]_d$, as the closed trace formula such that:

$$
\begin{aligned}
[\![\bot]\!]_d &= \bot && \text{False}\\
[\![\top]\!]_d &= \top && \text{True}\\
[\![active]\!]_d &= \exists t : Active()@t && \text{Active adversary}\\
[\![R_s]\!]_d &= \exists t : RevealTest()@t && \text{Alice's static is revealed}\\
[\![R_{rs}]\!]_d &= \exists t : RevealPeer()@t && \text{Charlie's static is revealed}\\
[\![R_{psk}]\!]_d &= \exists k, t : RevealPsk(k)@t && \text{Any pre-shared key is revealed}
\end{aligned}
$$

and for $x, y \in E_a^\star$, we lift conjunction and disjunction to trace formulas:

$$
\begin{aligned}
[\![x \mathbin{\dot\wedge} y]\!]_d &= \hat\sigma, t \mapsto [\![x]\!]_d \mathbin{\breve\wedge} [\![y]\!]_d\\
[\![x \mathbin{\dot\vee} y]\!]_d &= \hat\sigma, t \mapsto [\![x]\!]_d \breve\vee [\![y]\!]_d
\end{aligned}
$$

This semantics is very close to trace semantics, but global. R_{psk} forbids reveals of *any* pre-shared key, R_s forbids reveals of the static key of *any* Charlie, *etc.* With trace properties, R_{psk} would only denote that if a pre-shared key is used to convey a payload whose secrecy is claimed (for example), then and only then is must not be revealed. The reason for this difference is that anonymity is a global property; not restricted to, say, the 3$^{\text{rd}}$ payload.

Assumptions made in the model In its Section 7.8 [14], the Noise specification presents the identity hiding properties of various patterns, and makes two assumptions: ephemeral private keys are not revealed, and "parties abort the handshake if they receive a static public key from the other party which they don't trust". We keep the former assumption for performance reasons. The latter is more interesting: if diff(Alice, Bob) does not receive Charlie's static key in pre-messages and the adversary is active, then anonymity does not hold. The reason is that the adversary can try to start a session with diff(Alice, Bob) assuming this is Alice, and see if the handshake is aborted. This could be formalized in a very similar fashion to Lemma 16. Therefore, only threat models where agents only accept to start sessions with honest peers (read: peers with an honestly generated static) may guarantee anonymity and are worth exploration. Hence the use of the key facts: role r is created with a $!TestPk$ fact and only accepts $!PeerPk$ keys as peer, and role $\tilde r$ is created with a $!PeerPk$.

Note that we should let $\tilde r$ accept any key, but unfortunately this makes proofs computationally infeasible. Therefore, we only let $\tilde r$ accept $!Pk$ peers. This is a shortcoming of our model.

An important divergence from the specification is the fact that public keys are public. The specification distinguishes the cases where the attacker has or does not have access to a list of candidate public keys. The attack mentioned above (for each possible key, the attacker starts a session with the target and checks whether the handshake completes) is only possible with a reasonably short list of candidates. With a sufficiently large list, this attack may be dismissed as infeasible. Nevertheless, in a symbolic model, either public keys are not public at all, or the attacker can test all public keys in the world. There is no middle ground. Let us call these two cases *hidden public keys* and *accessible public keys*, respectively.

Now, observe that hidden public keys cannot be modelled as a bi-system. If one removes the Out_{sys} facts from rule (2.30), the left and right versions of the rule become rigorously identical, so observational equivalence will always hold, even for handshakes where diff(Alice, Bob)'s static key is sent in clear text. In this case, the challenge we pose to the adversary is of a different

nature. We do not ask him compare two systems, but rather to *produce* the value of the static key given a transcript. This makes modelling the private public keys case a completely different endeavour. We have not explored it, for lack of time.

To sum up, we assume that public keys are really public, which implies that the adversary has access to the list of all public keys in existence and can test them one by one until he finds the right one because our modelling is symbolic and does not limit the adversary to "reasonable" computing power[2]. This makes our modelling of anonymity quite strong.

Definition 44 (Ideal anonymity lemma)**.** Let C be an anonymity claim for role r in the handshake H. The bi-system $R_0(C)$ is composed of the following rules:

Adversary rules rules (2.3)–(2.7), built-in in Tamarin

Low-level semantics of H rules (1.7)–(1.14), for both roles

Application layer rules (2.8)–(2.12), for both roles

Message channel rules (2.13)–(2.15)

Payload channel rule (2.16): payloads are public values

Prologue channel rule (2.18): the prologue is a public constant

Ephemeral key generation rule (2.24): ephemerals are not revealable

Pre-shared key generation rule (2.22) and (2.23)

Static keys rules (2.30)–(2.34)

with the following modifications:

- The initialisation rule (2.8) does not input $!Pk(s, g^s)$ but $!TestPk(s, g^s)$ for role r and $!PeerPk(s, g^s)$ for role $\tilde{r}$.
- The initialisation rule (2.8) does not input $!Peer(rs)$ but $!PeerPk(w, rs)$ for role r and $!TestPk(w, rs)$ for role $\tilde{r}$.
- If role r learns rs at the i-th message, then the rule (1.9) receiving this message gains the additional premise $!PeerPk(w, \sigma_i[rs])$.
- If role $\tilde{r}$ learns rs at the i-th message, then the rule (1.9) receiving this message gains the additional premise $!Pk(w, \sigma_i[rs])$.

Additionally, this bi-system is restricted by the trace formula:

$$\forall x, t_1, t_2 : Create(x)@t_1 \mathbin{\dot\wedge} Create(x)@t_2 \rightsquigarrow t_1 = t_2 \tag{2.35}$$

Note that the $Create$ action fact is to be found on rule (2.8). The restriction (2.35) expressed that at most one initiator and one responder may be created. The need for this restriction—which is not attack preserving—comes from the fact that Tamarin does not support inductive reasoning in observational equivalence mode.

Remark. For the interested reader, here is a more precise justification for the necessity of restriction (2.35). Let us consider for example our strictest threat model $T = R_s \mathbin{\dot\vee} R_{rs} \mathbin{\dot\vee} active \mathbin{\dot\vee} R_{psk}$, and the anonymity of the initiator in the handshake XX:

[2]Of course we assume that cryptographic primitives cannot be broken by this attacker.

```
XX:
  -> e
  <- e, ee, s, es
  -> s, se
```

For an integer $n \in \mathbb{N}$, we consider n responders $C_1, \ldots, C_n$ with ephemeral keys $e_1, \ldots, e_n$ and one initiator with ephemeral key e_I. Note that the responder expects a first message in the form $\langle g^e, x\rangle$ where x could be anything—in a non symbolic setup, any bytestring below the maximum message size imposed by the specification. The initiator sends C_1 the first message $\langle g^{e_I}, z_I\rangle$ where e_I is his ephemeral key and z_I is a payload. Then for $1 \leq i < n$, C_i answers with $\langle g^{e_i}, w_i\rangle$ where w_i is a term including C_i's static and various other components. This message is not addressed to the initiator, but to C_{i+1} who expects his own first message.

This construction proves that for any size n we can create a valid dependency graph greater than this size. This implies that Tamarin will not terminate; this type of proofs would require inductive reasoning which Tamarin supports only for trace properties. Therefore, we need to bound the number of agents.

The argument above works as soon as the first message of the handshake is `-> e`. Similar infinite chains have been also empirically observed when the first message of the handshake is `-> e, s`, but in some cases only. A slightly different argument, but still based on "a family of dependency graphs of arbitrary big size exists" can be made when the adversary is active, for any handshake. These cases already represent a large proportion of those we consider, and trying to find a list of cases where this restriction is not needed turned out to be a wild goose chase. For these reasons, we apply the restriction (2.35) uniformly on all anonymity lemmas.

Why at most only one agent per role and not, say, two? We wanted to have results for at least all handshakes patterns which have a security level in the specification for comparison. Even with a bound of two, roughly only patterns where keys are exchanged in pre-messages would have reasonable proof times. Nevertheless, if someone is only interested in such a pattern, or only in a passive adversary, it is reasonable to expect that a higher bound can be used. Our Tamarin theory generator is configurable in this respect. △

A last detail worth mentioning is that, contrary to agreement and secrecy lemmas, we do not make the session identifier public (rule (2.28)). The reason is that this makes anonymity false for all non PSK handshakes. Observe that if a handshake does not contain the `psk` token, then the session identifier ($\sigma_n[h]$ where n is the number of messages) only contains public values. We can write $\sigma_n[h] = f(\text{diff}(s_A, s_B))$ where s_A and s_B are the static key of Alice and Bob, and f is a function computable by the attacker. A passive eavesdropper can thus compare the value of $\sigma_n[h]$ to $f(s_A)$ and $f(s_B)$ and discover the identity of diff(Alice, Bob). This is another instance of attack relying on a directory of all existing public keys; it may be unrealistic in some cases.

Now we can extend the notation $T \vDash C$ to anonymity lemmas:

Definition 45 (Notation)**.** Let C be an anonymity claim, and $T \in E_a^\star$ a threat model. We say that T guarantees C if $\mathcal{L}(R_0(C)|_{\neg[\![T]\!]_d}) \sim_{Env} \mathcal{R}(R_0(C)|_{\neg[\![T]\!]_d})$, and then we write $T \vDash C$. The corresponding syntactic statement is $T \vdash C$.

Real anonymity lemma In the rest of this section we discuss further performance optimisations. This is not essential and can be left for a second reading.

The lemma defined in Definition 45 is hard to prove for Tamarin. We now define a different lemma which takes into account the chosen threat model for optimal performance. We then prove that they are equivalent.

Definition 46 (Anonymity lemma)**.** Let C be an anonymity claim for role r in the handshake H. Let $T \in E_a^\star$ be a disjunction of exclusions in E_a. The bi-system $R(C, T)$ is composed of the following rules:

Adversary rules rules (2.3)–(2.7), implicitly added by Tamarin;

Low-level semantics of H rules (1.7)–(1.14), for both roles;

Application layer rules (2.8)–(2.12), for both roles;

Message channel If $active \rightharpoonup T$ then rules (2.13) and (2.14) else (2.15) and

$$Send(x) \to Out_{sys}(x) \tag{2.36}$$

Payload channel rule (2.16) only;

Prologue channel rule (2.18) only;

Ephemeral key generation rule (2.24) only;

Pre-shared key generation rule (2.22) and, if $R_{psk} \not\rightharpoonup T$ then (2.23);

Peer static key generation rule (2.33), and if $R_{rs} \not\rightharpoonup T$ (2.34);

Test agent static key generation rules (2.30), (2.31), and if $R_{rs} \not\rightharpoonup T$

$$\begin{aligned} Fr(a), Fr(b) \to {!}TestPk(\mathrm{diff}(a,b), g^{\mathrm{diff}(a,b)}), Out_{sys}(a), \\ Out_{sys}(b), {!}TestAlternatives(a+b) \end{aligned} \tag{2.37}$$

with the following modifications:

- The initialisation rule (2.8) does not input $!Pk(s, g^s)$ but $!TestPk(s, g^s)$ for role r and $!PeerPk(s, g^s)$ for role $\tilde{r}$.
- The initialisation rule (2.8) does not input $!Peer(rs)$ but $!PeerPk(w, rs)$ for role r and $!TestPk(w, rs)$ for role $\tilde{r}$.
- If role r learns rs at the i-th message, then the rule (1.9) receiving this message gains the additional premise $!PeerPk(w, \sigma_i[rs])$.
- If role $\tilde{r}$ learns rs at the i-th message, then the rule (1.9) receiving this message gains the additional premise $!Pk(w, \sigma_i[rs])$.

Additionally, this bi-system is restricted by restriction (2.35).

Lemma 2. *Let C be an anonymity claim. Let $T \in E_a^\star$ a disjunction of exclusions.*

$$T \vDash C \Leftrightarrow \mathcal{L}(R(C,T)) \sim_{Env} \mathcal{R}(R(C,T))$$

Proof. This is quite tedious, so we will leave some details to the reader. Let $A = R(C,T)$ and $B = R_0(C)|_{\neg[\![T]\!]_d}$. It is enough to exhibit an invertible mapping f from $\mathrm{dgs}(\mathcal{L}(A))$ to $\mathrm{dgs}(\mathcal{L}(B))$ and $\mathrm{dgs}(\mathcal{R}(A))$ to $\mathrm{dgs}(\mathcal{R}(B))$ such that $dg' \in \mathrm{mirrors}(dg) \Leftrightarrow f(dg') \in \mathrm{mirrors}(f(dg))$, that is, f preserves mirrorability.

The difference between $R_0(C)$ and $R(C,T)$ are:

1. The message channel: $R_0(C)$ has the rules (2.13)–(2.15) when $R(C,T)$ has (2.13) and (2.14) if T contains active and (2.36) and (2.15) otherwise.
2. Static reveals: identical if R_s is in T, else $R_0(C)$ has (2.30) and (2.32) when $R(C,T)$ has rule (2.37)

3. $R_0(C)$ has (2.23) when $R(C,T)$ has it only if T does not contain R_{psk}

4. $R_0(C)$ has (2.34) when $R(C,T)$ has it only if T does not contain R_{rs}

The last two points are not real differences. For example, regarding R_{psk}, if R_{psk} is missing then indeed a rule is missing from $R(C,T)$ but this rule is unusable in $R_0(C)|_{\neg[\![T]\!]_d}$, so this does not make the set of dependency graphs differ.

Therefore, we can define f to be the identity on all rules except when converting the rules which differ: (2.13)–(2.15), (2.36), (2.32), (2.37).

Let us start with the message channel when the adversary is active. f maps instance of rule (2.36) like:

$$\frac{Send(x)}{Out_{sys}(x)}\,(2.36)$$

to the corresponding instance of rule (2.13):

$$\frac{Send(x)}{Out_{sys}(x) \quad Receivable(x)}\,(2.13)$$

Going back is always possible because no rule in B consumes a *Receivable* fact. Therefore, there can be no outgoing edge from the *Receivable* fact.

Now we need an argument justifying that this transformation preserves mirrorability. Observe that in both bi-systems, the only rule with a diff term is of the form (simplified):

$$\frac{Fr(a) \quad Fr(b)}{\begin{array}{cccc} !TestPk(\mathrm{diff}(a,b)) & !TestAlternatives(a+b) & Out_{sys}(a) & Out_{sys}(b) \\ \vdots & \vdots & \vdots & \\ s_1 & s_2 & s_3 & \end{array}}$$

The output facts of this rule give rise to subtrees s_1, s_2, s_3 as show above, which may intersect. We argue that if a rule we modify does not input at least one premise from s_1 and one from s_3 then our mapping preserves mirrorability. Reason is that if it is completely in s_3 then this part of the dependency graph is left untouched by mirroring; if it is completely in s_1 then all occurrences of a (which is a nonce, and thus "atomic") are replaced by b and *vice versa*, so we cannot break mirrorability again. Finally, rules in s_2 can be chosen to not change when mixing with s_3 or change fully when mixing with s_1 since $a+b=b+a$.

The mapping outlined above only modifies rules with one premise, so mirrorability is preserved.

Dealing with static reveals is more tricky.

Table 2.1 presents how to convert usages of $Out_{sys}(x)$ outputs of rule (2.37) (two first lines) and how to come back, even if mirroring mirrors $a+b$ into $b+a$ (two last lines). This preserve mirrorability because it only involves subtrees s_2 and s_3, and we can add as many instances of rule (2.32) since $!TestAlternatives$ is persistent. Technically speaking, this mapping is not injective, but we can choose arbitrarily in one direction, label the modified rule with the choice we made, and when going back undo the right choice. This is always possible because mirroring preserves rules, and thus our label.

The last case to tackle is to convert the use of the $Out_{sys}(g^x)$ of rule (2.30). The Out_{sys} fact can only be used in rule (2.1) to get a In_{env} but then two usages are possible: converting this in knowledge for message deduction rules like (2.6), or using it as a comparison: (2.7). This comparison needs a $K(g^a)$ premise; it comes necessarily[3] from the interface with $In_{env}(g^a)$ and

[3]The inquisitive reader is refered to table 7 of the full version of [4] which contains the list of environment rules

$R(C,T)$	$R_0(C)\vert_{\neg[\![T]\!]_d}$
$\dfrac{Fr(a) \quad Fr(b)}{\ldots \quad \dfrac{Out_{sys}(a)}{\ldots}(2.1) \quad Out_{sys}(b)}(2.37)$	$\dfrac{Fr(a) \quad Fr(b)}{\ldots \quad \dfrac{!TestAlternatives(a+b)}{\ldots \quad \dfrac{Out_{sys}(a)}{\ldots}(2.1) \quad Out_{sys}(b)}(2.32)}(2.30)$
$\dfrac{Fr(a) \quad Fr(b)}{\ldots \quad Out_{sys}(a) \quad \dfrac{Out_{sys}(b)}{\ldots}(2.1)}(2.37)$	$\dfrac{Fr(a) \quad Fr(b)}{\ldots \quad \dfrac{!TestAlternatives(a+b)}{\ldots \quad Out_{sys}(a) \quad \dfrac{Out_{sys}(b)}{\ldots}(2.1)}(2.32)}(2.30)$
$\dfrac{Fr(a) \quad Fr(b)}{\ldots \quad Out_{sys}(a) \quad \dfrac{Out_{sys}(b)}{\ldots}(2.1)}(2.37)$	$\dfrac{Fr(a) \quad Fr(b)}{\ldots \quad \dfrac{!TestAlternatives(a+b)}{\dfrac{!TestAlternatives(b+a)}{\ldots \quad \dfrac{Out_{sys}(b)}{\ldots}(2.1) \quad Out_{sys}(a)}(2.32)}=_{\varepsilon}}(2.30)$
$\dfrac{Fr(a) \quad Fr(b)}{\ldots \quad \dfrac{Out_{sys}(a)}{\ldots}(2.1) \quad Out_{sys}(b)}(2.37)$	$\dfrac{Fr(a) \quad Fr(b)}{\ldots \quad \dfrac{!TestAlternatives(a+b)}{\dfrac{!TestAlternatives(b+a)}{\ldots \quad Out_{sys}(b) \quad \dfrac{Out_{sys}(a)}{\ldots}(2.1)}(2.32)}=_{\varepsilon}}(2.30)$

Table 2.1: Mapping of patterns between $R(C,T)$ and $R_0(C)\vert_{\neg[\![T]\!]_d}$ for the proof of Lemma 2

$R(C,T)$	$R_0(C)\vert_{\neg[\![T]\!]_d}$
$\dfrac{\dfrac{\dots \quad \dfrac{\dfrac{Fr(a) \quad Fr(b)}{Out_{sys}(a)}(2.1) \quad Out_{sys}(b)}{\dfrac{In_{env}(a)}{K(a)}(2.3)}(2.37)}{\dfrac{K(g) \quad K(a)}{K(g^a)}(2.6)}}{\dfrac{In_{env}(g^a) \quad \dfrac{\dfrac{Fr(a) \quad Fr(b)}{Out_{sys}(a)}(2.1) \quad \dots}{\dfrac{In_{env}(a)}{K(a)}(2.3)}(2.37)}{K(g^a)}(2.6)}{Out_{env}(\top)}(2.7)$	$\dfrac{\dfrac{\dots \quad \dfrac{Fr(a) \quad Fr(b)}{Out_{sys}(g^a)}(2.1) \quad Out_{sys}(g^b)}{\dfrac{In_{env}(g^a)}{K(g^a)}(2.3)}(2.30)}{\dfrac{\dfrac{In_{env}(g^a)}{K(g^a)}(2.3) \quad \dfrac{\dfrac{Fr(a) \quad Fr(b)}{Out_{sys}(g^a)}(2.1) \quad \dots}{In_{env}(g^a)}(2.30)}{Out_{env}(\top)}(2.7)$
$\dfrac{K(g), \quad \dfrac{K(a) \quad \dfrac{\dfrac{Fr(a) \quad Fr(b)}{Out_{sys}(a)}(2.1) \quad \dots}{In_{env}(a)}(2.37)}{Out_{env}(\top)}(2.7)$	$\dfrac{\dfrac{K(g) \quad K(a)}{K(g^a)}(2.3) \quad \dfrac{\dfrac{Fr(a) \quad Fr(b)}{Out_{sys}(g^a)}(2.1) \quad \dots}{In_{env}(g^a)}(2.30)}{Out_{env}(\top)}(2.7)$

Analog cases with b are omitted.

Table 2.2: Continued from table 2.1, other cases

Role r	i	Best threat model for secrecy of the i-th payload for role r
Initiator	1	$\top$
Initiator	2	$(R_{re} \dot\wedge R_{rs}) \dot\vee R_e \dot\vee (D_{re} \dot\wedge R^<_{rs}) \dot\vee (D_{rs} \dot\wedge R^<_{re}) \dot\vee (D_{re} \dot\wedge D_{rs})$
Initiator	3	$(R_{re} \dot\wedge R_{rs} \dot\wedge R_{psk}) \dot\vee (R_e \dot\wedge R_{re} \dot\wedge R_{psk}) \dot\vee (R_e \dot\wedge R_s \dot\wedge R_{psk}) \dot\vee (D_{re} \dot\wedge R^<_{rs} \dot\wedge R_{psk}) \dot\vee (D_{re} \dot\wedge R^<_e \dot\wedge R_{psk}) \dot\vee (D_{rs} \dot\wedge R^<_{re} \dot\wedge R_{psk}) \dot\vee (D_{re} \dot\wedge D_{rs} \dot\wedge R_{psk})$
Initiator	4	$(R_{re} \dot\wedge R_{rs} \dot\wedge R_{psk}) \dot\vee (R_e \dot\wedge R_{re} \dot\wedge R_{psk}) \dot\vee (R_e \dot\wedge R_s \dot\wedge R_{psk}) \dot\vee (D_{re} \dot\wedge R^<_{rs} \dot\wedge R^<_{psk}) \dot\vee (D_{re} \dot\wedge R^<_e \dot\wedge R^<_{psk}) \dot\vee (D_{rs} \dot\wedge R^<_{re} \dot\wedge R^<_{psk}) \dot\vee (D_{re} \dot\wedge D_{rs} \dot\wedge R^<_{psk})$
Initiator	5	$(R_{re} \dot\wedge R_{rs} \dot\wedge R_{psk}) \dot\vee (R_e \dot\wedge R_{re} \dot\wedge R_{psk}) \dot\vee (R_e \dot\wedge R_s \dot\wedge R_{psk}) \dot\vee (D_{re} \dot\wedge R^<_{rs} \dot\wedge R^<_{psk}) \dot\vee (D_{re} \dot\wedge R^<_e \dot\wedge R^<_{psk}) \dot\vee (D_{rs} \dot\wedge R^<_{re} \dot\wedge R^<_{psk}) \dot\vee (D_{re} \dot\wedge D_{rs} \dot\wedge R^<_{psk})$
Responder	1	$\top$
Responder	2	$R_{re} \dot\vee (R_e \dot\wedge R_s) \dot\vee D_{re}$
Responder	3	$(R_{re} \dot\wedge R_{rs} \dot\wedge R_{psk}) \dot\vee (R_e \dot\wedge R_{re} \dot\wedge R_{psk}) \dot\vee (R_e \dot\wedge R_s \dot\wedge R_{psk}) \dot\vee (D_{re} \dot\wedge R^<_{rs} \dot\wedge R^<_{psk}) \dot\vee (D_{re} \dot\wedge R^<_e \dot\wedge R^<_{psk}) \dot\vee (D_{rs} \dot\wedge R^<_{re} \dot\wedge R^<_{psk}) \dot\vee (D_{re} \dot\wedge D_{rs} \dot\wedge R^<_{psk})$
Responder	4	$(R_{re} \dot\wedge R_{rs} \dot\wedge R_{psk}) \dot\vee (R_e \dot\wedge R_{re} \dot\wedge R_{psk}) \dot\vee (R_e \dot\wedge R_s \dot\wedge R_{psk}) \dot\vee (D_{re} \dot\wedge R^<_{rs} \dot\wedge R^<_{psk}) \dot\vee (D_{re} \dot\wedge R^<_e \dot\wedge R^<_{psk}) \dot\vee (D_{rs} \dot\wedge R^<_{re} \dot\wedge R^<_{psk}) \dot\vee (D_{re} \dot\wedge D_{rs} \dot\wedge R^<_{psk})$
Responder	5	$(R_{re} \dot\wedge R_{rs} \dot\wedge R_{psk}) \dot\vee (R_e \dot\wedge R_{re} \dot\wedge R_{psk}) \dot\vee (R_e \dot\wedge R_s \dot\wedge R_{psk}) \dot\vee (D_{re} \dot\wedge R^<_{rs} \dot\wedge R^<_{psk}) \dot\vee (D_{re} \dot\wedge R^<_e \dot\wedge R^<_{psk}) \dot\vee (D_{rs} \dot\wedge R^<_{re} \dot\wedge R^<_{psk}) \dot\vee (D_{re} \dot\wedge D_{rs} \dot\wedge R^<_{psk})$

In plain English, one can for example deduce from the 7$^{\text{th}}$ row of this table that secrecy of the 2$^{\text{nd}}$ payload of XXpsk3 from the point of view of the responder holds unless both the responder's asymmetric keys are revealed, or the initiator's ephemeral key is either dishonestly generated or revealed.

Table 2.3: Best threat model for secrecy claims relative to XXpsk3

the coerce rule, or from $K(g)$ (g is public) and $K(a)$. These three cases are presented in table 2.2. The fact that the first line preserves mirrorability comes from the same argument as before. The two other need special care. Since they directly contain the comparison rule (2.7), and each time, one premise is unaffected by mirroring. If one side is mirrorable, then all premises are unaffected by mirroring. □

2.3 Best threat model

In the previous section, we described what application layer protocol, properties, and threat models we consider. Let us consider, for example, secrecy claims, and a handshake like XXpsk3:

The secrecy properties we consider are secrecy of the 1$^{\text{st}}$, 2$^{\text{nd}}$, 3$^{\text{rd}}$, 4$^{\text{th}}$, and 5$^{\text{th}}$ payloads, from the point of view of both the initiator and responder, and under more than 10^{12} distinct threat models. Leaving the problem of computation time aside for a moment, we still have to solve the problem of presenting the results. A table presenting for each of these properties, whether it holds with XXpsk3 would have billions of rows. We will take advantage of the structure of $\bar{E}$ to condense the results. For a given claim, we will only give the so-called *best threat model* under which this claim still holds, like in table 2.3. Best threat model roughly means most permissive threat model under which the claim holds. We will then show that this information is enough to retrieve whether a claim holds under *any* threat model. This construction is very close to the notion of protocol-security hierarchy developed by Basin and Cremers [3].

2.3.1 Construction

Lemma 3. *For any agreement or secrecy claim C and threat models $A, B \in E^\star$,*

$$A \vDash C \wedge B \vDash C \Leftrightarrow A \dot{\wedge} B \vDash C$$

Proof. Observe that for all agreement or secrecy claims C, $\varphi(C, [\![T]\!]_t)$ can be written as

$$\forall \hat{\sigma}, t, u_1, \ldots, u_p : A_C(\hat{\sigma}, u_1, \ldots, u_p)@t \rightsquigarrow B_C(\hat{\sigma}, u_1, \ldots, u_p, t) \breve{\vee} [\![T]\!]_t(\hat{\sigma}, t)$$

With these notations, the lemma reads as:

$$\begin{aligned}
&\Big(R(C) \mid\!\breve{\models} \forall \hat{\sigma}, t, u_1, \cdots : A_C(\hat{\sigma}, u_1, \ldots)@t \rightsquigarrow B_C(\hat{\sigma}, u_1, \ldots, t) \breve{\vee} [\![A]\!]_t(\hat{\sigma}, t)\Big) \\
\wedge &\Big(R(C) \mid\!\breve{\models} \forall \hat{\sigma}, t, u_1, \cdots : A_C(\hat{\sigma}, u_1, \ldots)@t \rightsquigarrow B_C(\hat{\sigma}, u_1, \ldots, u_p, t) \breve{\vee} [\![B]\!]_t(\hat{\sigma}, t)\Big) \\
\Leftrightarrow &\big(R(C) \mid\!\breve{\models} \forall \hat{\sigma}, t, u_1, \cdots : A_C(\hat{\sigma}, u_1, \ldots)@t \\
&\qquad\qquad \rightsquigarrow B_C(\hat{\sigma}, u_1, \ldots, t) \breve{\vee} \left([\![A]\!]_t(\hat{\sigma}, t) \breve{\wedge} [\![B]\!]_t(\hat{\sigma}, t)\right)\big)
\end{aligned}$$

which is obvious. □

Lemma 4. *If $A \rightharpoonup B$, then for any secrecy and agreement claim C,*

$$A \vDash C \Rightarrow B \vDash C$$

Proof. Let C be a secrecy or agreement claim. Let $A, B \in E^\star$ such that $A \rightharpoonup B$ and $R(C) \mid\!\breve{\models} \varphi(C, [\![A]\!]_t)$.

Let x be a key. It is obvious that

$$R(C) \mid\!\breve{\models} \varphi(C, [\![R_x^<]\!]_t \rightsquigarrow [\![R_x]\!]_t)$$

once we unfold its definition:

$$\begin{aligned}
&R(C) \mid\!\breve{\models} \forall \hat{\sigma}, t : A_C(\ldots)@t \rightsquigarrow B_C(\ldots) \\
&\qquad \breve{\vee}((\exists t' : Reveal \ldots (\sigma_k[x])@t' \breve{\wedge} t' \prec t) \rightsquigarrow (\exists t' : Reveal \ldots (\sigma_k[x])@t'))
\end{aligned}$$

Let us now justify that $R(C) \mid\!\breve{\models} \varphi(C, [\![A]\!]_t \breve{\wedge} ([\![D_{rs}]\!]_t \rightsquigarrow [\![D_{pki}]\!]_t))$. If the role of the claim C does not receive their peer's static in pre-messages (that is $\sigma_0[rs] = \mathsf{empty}$) then the presence or absence of $DishonnestPki()$ in a trace has no influence on the validity of any lemma we could consider for this claim. Therefore, $R(C) \mid\!\breve{\models} \varphi(C, [\![A \dot{\wedge} D_{pki}]\!]_t)$ and thus $R(C) \mid\!\breve{\models} \varphi(C, [\![A]\!]_t \breve{\wedge} ([\![D_{rs}]\!]_t \rightsquigarrow [\![D_{pki}]\!]_t))$.

Otherwise, observe that $\hat{\sigma}[rs]$ is a key which went through rules (2.26) or (2.27). If $[\![D_{rs}]\!]_t(\hat{\sigma}, \cdot)$ holds then it could not have been (2.26) and thus $[\![D_{pki}]\!]_t(\cdot, \cdot)$ holds. Thus, $R(C) \mid\!\breve{\models} \varphi(C, [\![A]\!]_t \breve{\wedge} ([\![D_{rs}]\!]_t \rightsquigarrow [\![D_{pki}]\!]_t))$.

By similar arguments, one can also prove that for $x = re, rs, pki$, $R(C) \mid\!\breve{\models} \varphi(C, [\![A]\!]_t \breve{\wedge} ([\![D_x]\!]_t \rightsquigarrow [\![active]\!]_t))$: honest agents only send honest keys so if an agent received a dishonest key then the adversary was active.

Let $X = ([\![R_e^<]\!]_t \rightsquigarrow [\![R_e]\!]_t) \breve{\wedge} ([\![R_{re}^<]\!]_t \rightsquigarrow [\![R_{re}]\!]_t) \breve{\wedge} ([\![R_s^<]\!]_t \rightsquigarrow [\![R_s]\!]_t) \breve{\wedge} ([\![R_{rs}^<]\!]_t \rightsquigarrow [\![R_{rs}]\!]_t) \breve{\wedge} ([\![R_{psk}^<]\!]_t \rightsquigarrow [\![R_{psk}]\!]_t) \breve{\wedge} ([\![D_{rs}]\!]_t \rightsquigarrow [\![active]\!]_t) \breve{\wedge} ([\![D_{re}]\!]_t \rightsquigarrow [\![active]\!]_t) \breve{\wedge} ([\![D_{pki}]\!]_t \rightsquigarrow [\![active]\!]_t) \breve{\wedge} ([\![D_{rs}]\!]_t \rightsquigarrow [\![D_{pki}]\!]_t)$. By Proposition 2 we can summarize our hypothesis and previous observations as:

$$R(C) \mid\!\breve{\models} \varphi(C, [\![A]\!]_t \breve{\wedge} [\![X]\!]_t \breve{\wedge} ([\![X]\!]_t \rightsquigarrow [\![A]\!]_t \rightsquigarrow [\![B]\!]_t))$$

and thus $R(C) \mid\!\breve{\models} \varphi(C, [\![B]\!]_t)$. □

Now we need an equivalent of the previous lemmas for anonymity claims.

Lemma 5. *Let C be an anonymity claim, and $T \in E_a^\star$ a threat model.*

If $dg \in \mathrm{dgs}\left(\mathcal{L}\left(R_0(C)|_{\neg[\![T]\!]_t}\right)\right)$ has a mirror $dg' \in \mathrm{dgs}\left(\mathcal{R}\left(R_0(C)\right)\right)$, then we have $dg' \in \mathrm{dgs}\left(\mathcal{R}\left(R_0(C)|_{\neg[\![T]\!]_t}\right)\right)$. This also holds when replacing $\mathcal{L}$ by $\mathcal{R}$ and vice versa.

Proof. Observe that whether $\tau(dg') \breve{\models} \neg[\![T]\!]_d$ depends only on which rules are present in dg'. For example, when $T = R_{psk}$, this is the case if, and only if, dg' does not contain an instance of rule (2.23). dg and dg' are mirrors of one another so their nodes are instances of the same rules. $\tau(dg) \breve{\models} \neg[\![T]\!]_d$ so dg cannot use the "forbidden rules" which implies that dg' cannot use them either. Hence, $\tau(dg') \breve{\models} \neg[\![T]\!]_d$, which gives the expected result. □

Lemma 6. *Let $A, B \in E_a^\star$. If $A \rightharpoonup B$, then for any anonymity claim C,*

$$A \vDash C \Rightarrow B \vDash C$$

Proof. Let C be an anonymity claim, and $A, B \in E_a^\star$ such that $A \rightharpoonup B$ and $A \vDash C$. We have to show that $\mathcal{L}\left(R_0(C)|_{\neg[\![B]\!]_d}\right) \sim_{Env} \mathcal{R}\left(R_0(C)|_{\neg[\![B]\!]_d}\right)$.

Let $dg \in \mathrm{dgs}\left(\mathcal{L}\left(R_0(C)|_{\neg[\![B]\!]_d}\right)\right)$. This means that $dg \in \mathrm{dgs}\left(\mathcal{L}\left(R_0(C)\right)\right)$ and $\tau(dg) \breve{\models} \neg[\![B]\!]_t$. Note that on $E_a^\star$, $\rightharpoonup$ and $\twoheadrightarrow$ coincide, so $A \rightharpoonup B$ implies that $\tau(dg) \breve{\models} \neg[\![A]\!]_t$. Therefore, $dg \in \mathcal{L}\left(R_0(C)|_{\neg[\![A]\!]_d}\right)$. By hypothesis, this proves that dg has a mirror $dg' \in \mathrm{dgs}\left(\mathcal{R}\left(R_0(C)|_{\neg[\![A]\!]_d}\right)\right)$. dg' is a mirror of dg in $R_0(C)$ and by Lemma 5, $dg' \in \mathrm{dgs}\left(\mathcal{R}\left(R_0(C)|_{\neg[\![B]\!]_d}\right)\right)$. □

We can deduce from lemmas 4 and 6 the following statement:

Proposition 1. *Let C be a claim. Let $F \subseteq E_a$ if C is an anonymity claim, or $F \subseteq E$ otherwise. Let $A, B \in F^\star$. If $A \rightharpoonup B$, then*

$$A \vDash C \Rightarrow B \vDash C$$

Corollary 1. *Let C be a claim. Let $F \subseteq E_a$ if C is an anonymity claim, or $F \subseteq E$ otherwise. Let $A, B \in F^\star$. If $A \rightleftharpoons B$ then $A \vDash C \Leftrightarrow B \vDash C$. As a result, we can lift the operator $\cdot \vDash C$ to $\bar{F}$ the quotient of $F^\star$ by $\rightleftharpoons$. We can thus write:*

$$\forall T, T' \in \bar{F}, T \rightharpoonup T' \Rightarrow T \vDash C \Rightarrow T' \vDash C$$

We also extend Lemma 3:

Lemma 7. *For any anonymity claim C and threat models $A, B \in E_a^\star$,*

$$A \vDash C \wedge B \vDash C \Leftrightarrow A \dot{\wedge} B \vDash C$$

Proof. The implication ($\Leftarrow$) can be immediately derived from Proposition 1. For the implication ($\Rightarrow$), assume $A \vDash C$ and $B \vDash C$. Let $dg \in \mathrm{dgs}\left(\mathcal{L}\left(R_0(C)|_{\neg[\![A \dot{\wedge} B]\!]_d}\right)\right)$. $\neg[\![A \dot{\wedge} B]\!]_d = \neg[\![A]\!]_d \breve{\vee} \neg[\![B]\!]_d$ so either $dg \in \mathrm{dgs}\left(\mathcal{L}\left(R_0(C)|_{\neg[\![A]\!]_d}\right)\right)$ or the same for B. By symmetry, we only consider the first case. By hypothesis, dg has a mirror, and by Lemma 5, this mirror is valid in $dg \in \mathrm{dgs}\left(\mathcal{R}\left(R_0(C)|_{\neg[\![A \dot{\wedge} B]\!]_d}\right)\right)$.

The same reasoning works from the right-hand side to the left-hand side. □

From lemmas 3 and 7 we derive:

Proposition 2. *Let C be a claim. Let $F \subseteq E_a$ if C is an anonymity claim, or $F \subseteq E$ otherwise. Let $A, B \in F^\star$.*

$$A \vDash C \wedge B \vDash C \Leftrightarrow A \mathbin{\dot\wedge} B \vDash C$$

This enables us to define the best threat model:

Lemma 8 (Definition of the best threat model)**.** *Let C be a claim. Let F be a set of exclusions. There exists a unique threat model $B_F(C) \in \bar{F}$ such that $B_F(C) \vDash C$ and $\forall T \in \bar{F}, T \vDash C \Rightarrow B_F(C) \rightharpoonup T$. We call $B_F(C)$ the best threat model for C. By convention, $B(C)$ is a shortcut for $B_E(C)$ for secrecy and agreement claims, or $B_{E_a}(C)$ for anonymity claims.*

Proof. We first prove existence of a representative X of the best threat model in $F^\star$ and then uniqueness of this representative up to $\rightleftharpoons$.

Existence Let $\mathbf{C} \subset F^\star$ be the set of all positive clauses of F atoms (modulo commutativity of disjunction). Let $\mathbf{C}^+ = \{T \in \mathbf{C} \mid T \vDash C\}$. Let $X = \dot\bigwedge_{T \in \mathbf{C}^+} T$. By Proposition 2 we have $X \vDash C$.

Let $T_0 \in F^\star$ such that $T_0 \vDash C$. We write T_0 in conjunctive normal form. Remember that $F^\star$ is the closure of F by $\dot\wedge$ and $\dot\vee$, without negation. By de Morgan's law, we can write T_0 as a conjunction of disjunctions of elements of F without negations. Since the CNF is unique, the CNF of T_0 cannot involve negations. Therefore, there exists a set $S \subseteq \mathbf{C}$ such that $T_0 = \dot\bigwedge_{T \in S} T$. By Proposition 2 again, all $T \in S$ satisfy $T \vDash C$ which implies $S \subseteq \mathbf{C}^+$ and then $X \rightharpoonup T_0$.

Uniqueness If A and B satisfy the definition of a representative of the best threat model, then

- $B \vDash C$ so $A \rightharpoonup B$.
- $A \vDash C$ so $B \rightharpoonup A$.

Therefore $A \rightleftharpoons B$.

The equivalence class $B_F(C) \in \bar{F}$ of all the $X \in F^\star$ such that $X \vDash C$ and $\forall T \in F^\star, T \vDash C \Rightarrow X \rightharpoonup T$ is thus non-empty and unique.

□

Proposition 3. *Let C be a claim and $T \in \bar{F}$ a threat model.*

$$T \vDash C \Leftrightarrow B_F(C) \rightharpoonup T$$

Proof. The direct implication is given by Lemma 8. For the other direction: if $B_F(C) \rightharpoonup T$ then we observe that by definition of the best threat model $B_F(C) \vDash C$ and by Proposition 1 we get that $T \vDash C$. □

This shows that the sole value $B(C)$ accurately describes in which threat models a claim C holds, and more generally, whether C holds "often" or not.

Example 29 (Best threat model for secrecy of the 4$^{\text{th}}$ payload of IK for the initiator). Let C be the claim "secrecy of the 4$^{\text{th}}$ payload of IK for the initiator". $B(C) = (D_{re} \mathbin{\dot\wedge} D_{rs} \mathbin{\dot\wedge} \mathit{active} \mathbin{\dot\wedge} D_{pki}) \dot\vee (D_{rs} \mathbin{\dot\wedge} \mathit{active} \mathbin{\dot\wedge} D_{pki} \mathbin{\dot\wedge} R_{re}^{<}) \dot\vee (R_e \mathbin{\dot\wedge} R_s) \dot\vee (R_{re} \mathbin{\dot\wedge} R_{rs}) \dot\vee (D_{re} \mathbin{\dot\wedge} \mathit{active} \mathbin{\dot\wedge} R_{rs}^{<})$. This means that threat models which are not weaker than $B(C)$, like $T = R_e \dot\vee R_{re} \dot\vee R_s \dot\vee R_{rs}$ the naive secrecy threat model of equation 2.29, do not guarantee secrecy. By not weaker we mean $B(C) \not\rightharpoonup T$. The threat models guaranteeing secrecy are exactly those subsumed by $B(C)$, like non-naïve TOFU secrecy $T' = D_{rs} \dot\vee T$. △

An intuitive way of reading the best threat model of a claim is that each conjunction describes minimal assumption for one of the possible attacks. The clause $D_{re} \dot\wedge D_{rs} \dot\wedge active \dot\wedge D_{pki}$ expresses for example that if we talk directly to the adversary (this requires an active adversary which registers his own keys in the pre-message PKI *etc.*) then secrecy is voided. The clause $R_e \dot\wedge R_s$ expresses that fully revealing the initiator, even long after the end of the session, is enough to void secrecy. We easily see that this attack is passive: the exclusion *active* is not conjoined here. The clause $D_{re} \dot\wedge active \dot\wedge R^{<}_{rs}$ expresses that if the attacker has access to the responder's static key before the claim, and does some trickery involving being active and sending his own ephemeral key to the initiator, then secrecy is voided. This information is enough to let us guess the exact attack: the responder's static is compromised before the session, and the attacker then impersonates the (honest) responder. More importantly, we know that this attack is minimal, that is, if we remove any of these ingredients (D_{re}, *active*, $R^{<}_{rs}$), then the attack becomes impossible.

Example 29 (continuing from p. 56). With these observations in mind, here is an interpretation of our example best threat model: secrecy for the initiator at the 4^{th} payload of IK holds unless either party is fully revealed or the initiator is directly speaking to the adversary or the adversary is active and reveals at least one key before the claim. This corresponds to perfect forward secrecy. △

2.3.2 Computation

The existence proof of the best threat model is constructive. The algorithm to compute it is as follows: determine with Tamarin the set $\mathbf{C}^+$ of positive clauses T in $E^\star$ such that $T \vDash C$, and then $B(C)$ is the equivalence class of $\dot\bigwedge_{T \in \mathbf{C}^+} T$.

Naively this suggests that $2^{|E|}$ lemmas must be submitted to Tamarin. One can refine this a bit. The remainder of this section is dedicated to building sufficient conditions for us to omit submitting a clause to Tamarin.

Implied exclusions Some exclusions in E imply each other. We have $R^{<}_i \rightharpoonup R_i$, $D_* \rightharpoonup active$ and $D_{rs} \rightharpoonup D_{pki}$. Since we quotient $E^\star$ with $\rightleftharpoons$, this makes the set of clauses to consider smaller. More specifically there are $2^{|E|} = 16384$ clauses over E, but when taking account of the quotient by $\rightleftharpoons$, there are only $2^{3^5(1+3\times 2)} = 1701$ distinct clauses.

Absent keys When the claim is at a point in the handshake where the initiator has no ephemeral key yet (for example), then we can omit all clauses mentioning the exclusion "reveal of this key". Formally, we can compute the best threat model on a strict subset F of E which drops these unneeded exclusions.

Similarly, when the handshake involves no pre-messages, we can discard D_{pki}.

Passive adversary When the adversary is passive, its influence on the transcript of the test session is limited to the ordering of messages. The knowledge of the adversary cannot be the cause of any event besides adversary knowledge. Therefore, we can freely reorder reveals in traces, and the knowledge of the adversary cannot influence agreement claims.

Lemma 9. *For any (injective or non-injective) agreement claim C, positive clause T over E and key k, $T \dot\vee active \dot\vee R_k \vDash C$ if, and only if, $T \dot\vee active \vDash C$*

Proof. The implication ($\Leftarrow$) can be derived from Proposition 1. For the implication ($\Rightarrow$), by contrapositive, let us show that if there is a counterexample trace $t \in \mathrm{trace}(\mathrm{exec}(R(C))$ such that $t \not\breve\vDash \varphi\,(C, [\![T \dot\vee active]\!]_t)$ then we can exhibit a counterexample trace to $\varphi\,(C, [\![T \dot\vee active \dot\vee R^{<}_k]\!]_t)$.

Our trace t contains a Commit without matching Running (non-injective agreement) or two Commits for the same Running (injective agreement) and makes $[\![T]\!]_t$ and $[\![\mathit{active}]\!]_t$ false. If it does not make $[\![R_k]\!]_t$ true, then t is also a counterexample to $\varphi\,(C, [\![T \dot\vee \mathit{active} \dot\vee R_k]\!]_t)$. Otherwise, our trace contains a reveal action for k. Observe that given that the adversary is passive, our theory does not contain any usable `In` fact; so no rule instance except adversary knowledge depends on this reveal. Also, the agreement claim does not depend directly on the knowledge of the adversary. Therefore, the trace t' obtained by removing this reveal fact and its dependents is still a valid trace for our theory and still violates agreement. $[\![T]\!]_t$ and $[\![\mathit{active}]\!]_t$ are now still false. Therefore, t' is a counterexample for $\varphi\,(C, [\![T \dot\vee \mathit{active}]\!]_t)$. □

Lemma 10. *For any secrecy or agreement claim C, positive clause T over E and key k, $T \dot\vee \mathit{active} \dot\vee R_k^< \vDash C$ if, and only if, $T \dot\vee \mathit{active} \vDash C$.*

Proof. The implication ($\Leftarrow$) comes from Proposition 1. For the implication ($\Rightarrow$): in the case of agreement, by Proposition 2, we have that $R(C) \mathrel{\breve{\models}} \varphi\,(C, [\![T \dot\vee \mathit{active} \dot\vee R_k^<]\!]_t)$ implies $R(C) \mathrel{\breve{\models}} \varphi\,(C, [\![T \dot\vee \mathit{active} \dot\vee R_k]\!]_t)$. By Lemma 9 this implies $R(C) \mathrel{\breve{\models}} \varphi\,(C, [\![T \dot\vee \mathit{active}]\!]_t)$. For secrecy, by contrapositive, let us show that if there is a counterexample trace t to $R(C) \mathrel{\breve{\models}} \varphi\,(C, [\![T \dot\vee \mathit{active}]\!]_t)$ then we can exhibit a counterexample trace to $\varphi\,(C, [\![T \dot\vee \mathit{active} \dot\vee R_k^<]\!]_t)$.

Our trace t contains $K(p)$ where p is the secret payload and makes $[\![T]\!]_t$ and $[\![\mathit{active}]\!]_t$ false. Let us show that it cannot make $R_k^<$ true, which implies that t is also a counterexample to $\varphi\,(C, [\![T \dot\vee \mathit{active} \dot\vee R_k^<]\!]_t)$. Assume there is a reveal of k before the secrecy claim. Given that *active* is false, no `In` fact is usable in the theory. Therefore, no rule except adversary knowledge rules can depend on this reveal. Therefore, the secrecy claim and the reveal fact are not ordered, which is a contradiction. □

These two assertions allow us to omit

- all secrecy lemmas with a combination of a passive adversary and $R_k^<$ for some k;
- and all agreement lemmas with a combination of a passive adversary and any reveal.

Perfect forward agreement $R_k^<$ exclusions have been introduced in E with the explicit goal of modelling perfect forward secrecy. But there is no such thing as perfect forward agreement: these exclusions do not model interesting threat models in the case of agreement. Therefore, there is little value in keeping both $R_k^<$ and R_k in E in the case of agreement. We can also formally justify dropping R_k in the case of non-injective agreement.

Lemma 11. *For any non-injective agreement claim C, positive clause T over E and key k, $R(C) \mathrel{\breve{\models}} \varphi\,(C, [\![T \vee R_k]\!]_t)$ and $R(C) \mathrel{\breve{\models}} \varphi\,(C, [\![T \vee R_k^<]\!]_t)$ are equivalent.*

Proof. The implication ($\Leftarrow$) is obvious. For the implication ($\Rightarrow$), by contrapositive, assume we have a counterexample trace t for $\varphi\,(C, [\![T \vee R_k^<]\!]_t)$. Let us show we can exhibit a counterexample trace for $\varphi\,(C, [\![T \vee R_k]\!]_t)$ as well.

The trace t contains a Commit and no corresponding Running, and keeps $[\![T]\!]_t$ false. If it contains a reveal fact, then we can arbitrarily order it after the claim since $R_k^<$ is false. We cut the trace t after the claim. This leaves us with a still valid trace t' which still does not make $[\![T]\!]_t$ true and falsifies non-injective agreement, but does not contain a reveal of k any more. □

Note that this argument does not trivially extend to injective agreement. Falsifying injective agreement can involve two Commits and one Running where the second commit depends on the reveal we want to get rid of. In our experiments, it never happened, though.

This result allows us to omit all non-injective agreement lemmas with a R_k for some k.

Obviously false properties Noise is based on Diffie Hellman key exchanges, and yet the above observations do not take into account which DH tokens a pattern contains. We can take advantage of this knowledge. Let us start with a simplistic example.

Example 30 (Obviously false property). Considering XX:

```
XX:
  -> e
  <- e, ee, s, es
  -> s, se
```

and the secrecy of the third payload for the initiator under the threat model $T = R_s \dot{\vee} R_e \dot{\vee} D_{rs}$. This property does not hold because the adversary can reveal the responder's key and use his own ephemeral key to start a session with the initiator. There is nothing interesting in this claim; this threat model is so permissive that the attacker can perform straightforward impersonation. Colloquially, this property is obviously false. △

If we know that a property is false, then we can omit handing it to Tamarin. We will generalize the observation above into a sufficient condition to conclude that a property is false. It will only detect straightforward attacks but still will enable us to reduce the number of lemmas submitted to Tamarin.

Definition 47 (Usable key)**.** Let T be a positive clause over E. A key k (*i.e.* e, re, s, rs, or psk) is *usable* in threat model T if $R_k^< \not\triangleq T$ or k is re or rs and $D_k \not\triangleq T$.

In other words, the attacker can reveal the corresponding private key before the start of the session, or (for remote asymmetric keys only) r will accept a dishonestly generated key as k.

Example 31. With the threat model $T = D_{rs} \dot{\vee} R_e^< \dot{\vee} R_{re} \dot{\vee} R_{rs}^<$, usable keys are re, s and psk, "because" the attacker can reveal psk and s before the start of the session, and can use his own, dishonest re.

With the threat model $T' = T \dot{\vee} active$, re is not usable any more because a passive attacker can only use revealed keys.

△

The intuition is that if too many keys are usable by the attacker, then no claim will hold. We formalize the notion of "too many" as follows:

Definition 48 (Transparency)**.** A DH token is said to *refer* to two keys, one for the initiator and one for the responder. The first letter of the token denotes whether it refers to initiator's static (`s`) or the initiator's ephemeral (`e`) and likewise for the second letter and the responder.

Let H be a handshake pattern and r a role. Let T be a positive clause over E. H is said to be *transparent* to an attacker targeting r under threat model T if

- for all DH tokens t in H, t refer to at least one usable key, and
- if the `psk` token is present in H then psk is usable.

Example 32. Let us consider the handshake:

```
XX:
  -> e
  <- e, ee, s, es
  -> s, se
```

For an attacker targeting the initiator and under $T = D_{rs} \dot{\vee} R_s \dot{\vee} R_{rs}$ this handshake is transparent. He can use e and re which is enough to control `ee` (both keys), `es` (one key) and `se` (one key).

△

Lemma 12. *Let T a positive clause over E. Let H be a transparent handshake against r under threat model T. Let C be a claim on H. If C is an anonymity claim, then there exists a trace $\tau \in \text{trace}\left(\text{exec}\left(\mathcal{L}\left(R_0\left(C\right)|_{\neg[\![T]\!]_d}\right)\right)\right)$ such that*

$$\tau \breve{\models} \forall \sigma, t\colon State(r,\sigma)@t \rightsquigarrow \exists t'\colon K(\sigma[k])@t' \barwedge \exists t'\colon K(\sigma[ck])@t' \barwedge \exists t'\colon K(\sigma[h])@t'$$

If C is a secrecy or agreement claim, then there exists a trace $\tau \in \text{trace}(\text{exec}(R(C)))$ such that

$$\begin{aligned}\tau \breve{\models} \forall \sigma, t, t''\colon State(r,\sigma)@t \rightsquigarrow \neg[\![T]\!]_t(\sigma, t'')\\ \barwedge \exists t'\colon K(\sigma[k])@t' \barwedge \exists t'\colon K(\sigma[ck])@t' \barwedge \exists t'\colon K(\sigma[h])@t'\end{aligned}$$

Note that *State* facts are created in rules (1.7)–(1.9).

Proof. The core of the proof is that for a state σ, the values of interest, that is $\sigma[k]$, $\sigma[ck]$ and $\sigma[h]$, only contain public values, previously sent messages, the pre-shared key and DH terms. The adversary knows previously sent messages and public values. By hypothesis, we can include in our trace $RevealPsk(\sigma[psk])$ and still have $\neg[\![T]\!]$. We only have to craft a trace such that all DH terms are known to the attacker.

We consider two cases. If $active \rightharpoonup T$, then let τ be the trace of an honest execution of the protocol between two agents. $D_{re} \rightharpoonup active \rightharpoonup T$ and $D_{rs} \rightharpoonup T$ as well, so we know that for every DH token at least one can safely be revealed. By using the other public key we can compute the corresponding DH term.

If $active \not\rightharpoonup T$, then let τ be the trace of the attacker completing a handshake with r while abiding by the protocol. The attacker needs keys: remote keys are public, the pre-shared key can be revealed, but he also need private keys. Let k be a need private key. If $R_k^{<} \not\rightharpoonup T$ then the adversary uses an honest, revealed key. If $D_k \not\rightharpoonup T$ then the adversary uses 1 as private key. Else we claim that the attacker does not need a private key for k and uses a public honest key instead. Let us consider a DH token between r's key a and the attacker key b: the attacker must be able to compute $g^{a\star b}$. If the attacker knows the private key b, then he knows $g^{a\star b}$. Else, b is not usable and by hypothesis a is; and since a is r's key, this means that a can be revealed. Therefore, the attacker can compute $g^{a\star b}$. This proves that τ is indeed a valid trace for $R(C)$ or $\mathcal{L}\left(R_0\left(C\right)|_{\neg[\![T]\!]_d}\right)$ and that is satisfies the desired property. □

Example 33. Let us again consider the handshake:

```
XX:
  -> e
  <- e, ee, s, es
  -> s, se
```

As said in the previous example, for an attacker targeting the initiator and under $T = D_{rs} \dot{\vee} R_s \dot{\vee} R_{rs}$ this handshake is transparent. He can use e and re. Let us explicit the trace τ falsifying the secrecy of the 3rd payload for the initiator.

The attacker starts by revealing the ephemeral private key e of the initiator. He chooses 1 as his own private ephemeral. As static key, the attacker uses any public key $g^{s'}$ belonging to an

honest third party, whose private key s' he does not need to know. Let g^s be the static key of the initiator.

The attacker receives the first message g^e. It is not encrypted.

To send the second one, the attacker only needs[4] to be able to compute $g^{e\star 1}$ and $g^{e\star s'}$ which is possible given that e is known to the attacker.

To decrypt the next message, the attacker needs to know $g^{s\star 1}$. Here again, this is easy.

To sum up, this threat model is so permissive that the security of the whole handshake is voided.

△

Voiding secrecy needs only that the handshake be transparent "until" the payload we are interested in. To formalize that, we will consider truncated handshakes.

Definition 49 (Truncated handshake)**.** If H is a decorated handshake pattern, a truncation of H is a (decorated) handshake pattern H' where the list of message patterns is a prefix of that of H, and optionally the list of message tokens of the last message pattern of H' is a prefix of that of H.

Additionally, if H contains the message token `psk`, in the definition 11 of high-level semantics, the value of p is true for H' even if the token `psk` has been truncated away.

Example 34. Considering the undecorated pattern

```
NN:
  -> e
  <- e, ee
```

here are some truncations:

```
-> e, msg
```

and

```
-> e, msg
<- e
```

△

Here is an observation that truncating a handshake respects its high-level semantics.

Lemma 13 (High-level semantics of truncations)**.** *If H' is a truncation of H, and we denote their high-level semantics as $v' = (\sigma'_0, (\sigma'_1, c'_1), \ldots, (\sigma'_n, c'_n))$ and $v = (\sigma_0, (\sigma_1, c_1), \ldots, (\sigma_m, c_m))$ respectively then $(\sigma'_0, (\sigma'_1, c'_1), \ldots, (\sigma'_{n-1}, c'_{n-1}))$ is a prefix of v. Additionally, if the last message of H' is a full message of H then $(\sigma_n, c_n) = (\sigma'_n, c_n)$ and otherwise c_n is a tuple containing c'_n.*

Now we can prune some threat models which are, likewise, way too permissive for the claim we consider to hold.

Lemma 14. *Let T a positive clause over E. Let C be a secrecy claim for role r on the i-th payload of handshake H. We truncate H to i full messages, obtaining H'. If H' is transparent to an attacker targeting r under T then $T \not\vdash C$.*

Proof. Let C' be a copy of C for H' instead of H. The first observation is that $R(C') \not\Vdash \varphi\,(C', [\![T]\!]_t) \Rightarrow R(C) \not\Vdash \varphi\,(C, [\![T]\!]_t)$ (see Lemma 13).

Then we observe that since H' is transparent to an attacker targeting r under T, by Lemma 12 the attacker can compute the key encrypting the test payload which falsifies $\varphi(C', T)$.

□

[4] If you have trouble extracting this information from the pattern, please refer to paragraph 1.2

Lemma 15. *Let T a positive clause over E such that active $\not\rightharpoonup T$. Let C be an agreement claim for role r on the i-th payload of handshake H. We truncate H to i full messages, obtaining H'. If H' is transparent to an attacker targeting r under T then $R(C) \not\Vdash \varphi\,(C, [\![T]\!]_t)$.*

Proof. Let C' be a copy of C for H' instead of H. The first observation is that, again, $R(C') \not\Vdash \varphi\,(C', [\![T]\!]_t) \Rightarrow R(C) \not\Vdash \varphi\,(C, [\![T]\!]_t)$.

Then we observe that since H' is transparent to an attacker targeting r under T, we can obtain by Lemma 12 a trace τ where the attacker can sustain a session with r at least until the Commit claim and without any Running claim. Thus, τ falsifies $\varphi(C', T)$. □

Lemma 16. *Let T a positive clause over E_a. Let C be an anonymity claim for role r on the decorated handshake H. If there exists a way to truncate H into H' such that all of the following hold:*

1. *H' is transparent for an attacker targeting r*
2. *considering the high-level semantics of H' for $\tilde{r}$, the last state σ_n is such that $\sigma_n[rs] \neq \mathsf{empty}$ and $\sigma_n[k] \neq \mathsf{empty}$*
3. *One of the following holds:*
   (a) *the last message token of H' is `msg`, or*
   (b) *(the last message of H' is sent by r or active $\rightharpoonup T$) and the last token of H' is `s`.*

then $T \not\Vdash C$.

Proof. First, observe that given a trace falsifying $[\![T]\!]_d$ over H', we can get a similar trace over H still falsifying $[\![T]\!]_d$ and where the attacker knows at least more. This is because while processing new message tokens, future messages are only appended to: see Lemma 13.

We consider the counterexample trace τ given by Lemma 12 over H'.

By hypotheses 3 and $\sigma_n[k] \neq \mathsf{empty}$ we know that the last message c_n contains a term of the form $y = \mathsf{aead}(\sigma_n[k], n, \sigma_n[h], x)$, where n is public. The hypothesis 1 proves that x is known to the adversary and that $\sigma_n[k]$ and $\sigma_n[h]$ are computable from public values and revealed keys by the adversary. The hypothesis $\sigma_n[rs] \neq \mathsf{empty}$ proves that $\sigma_n[k]$ and $\sigma_n[h]$ depend on $\sigma_n[rs]$.

If c_n is sent by an honest party (this is the case when *active* $\rightharpoonup T$ or r sends this message), then the adversary can compute k_1 and h_1 the values of $\sigma_n[k]$ and $\sigma_n[h]$ under the assumption that diff(Alice, Bob) is Alice and then compute $\mathsf{verif}(k_1, n, h_1, y)$: this will return $\top$ if, and only if, diff(Alice, Bob) is Alice.

Else, the last token of H' is necessarily `msg` by hypothesis 3. The adversary can compute the value of the whole last message (which contains c_n) assuming that diff(Alice, Bob) is Alice and see whether the message is accepted by r.

□

Chapter 3

Results on common Noise handshakes

In this chapter we consider the set of all 53 two-way handshakes patterns mentioned in the specification [14]. For each secrecy, agreement, or anonymity claim on one of these patterns, we attempted to compute the *best threat model* under which it is holds, as per Lemma 8. Our results are complete for secrecy and agreement claims (tables B.4 and B.5), and only partial for anonymity claims (tables B.1, B.2, B.3) because not all proofs have terminated in reasonable time[1].

In the following, we analyse these results and compare them to those presented in the Noise specification [14] and Noise Explorer [11].

3.1 Secrecy and agreement

Before we delve into the analysis of the results themselves, let us quickly discuss their computational cost. We designed a special heuristic to enable Tamarin to prove the 143230 lemmas needed to determine the best threat model of the 928 secrecy and agreement claims we consider. On average, a lemma took 28 CPU seconds to prove.

3.1.1 General considerations

Let us first remind the reader that we assume that the private keys of agents are honestly generated and that the agents check that remote public keys are valid group elements upon reception.

The wide variety of patterns makes it difficult to make general observations. Still, here are some remarks.

Injective and non-injective agreement Let C be an injective agreement claim, and let C' be the corresponding non-injective agreement claim. We have that $B(C) = \mathit{active}$ or $B(C) = B(C')$. In other words, either injective agreement only holds against a passive adversary, or it holds exactly in the same threat models as non-injective agreement. Injective agreement can be viewed

[1]This was enforced subjectively and manually, but on average, the limit was 100 CPU hours and 60 GB of memory per anonymity lemma.

```
XX:
  -> e
  <- e, ee, s, es
  -> s, se
```

```
XXpsk3:
  -> e
  <- e, ee, s, es
  -> s, se, psk
```

Figure 3.1: We would expect XXpsk3 and XX to have the same properties when the pre-shared key is public.

as non-injective agreement plus non-replayability. Therefore, we can interpret this statement as "a message is either trivially replayable, or not at all".

Dummy pre-shared key As already mentioned, some protocols like Wireguard [6] use a pattern H requiring a pre-shared key but may use a public value (namely, 0) as the pre-shared key when the users cannot or do not want to provide one. It would be expected that in this case, H has exactly the same properties as the pattern H' obtained by removing `psk` tokens from H. Figure 3.1 presents an example of pair (H, H'). Our results show that this is always the case except for some agreement claims on payloads sent before the first DH token.

In the `psk` handshake H the first payload is always encrypted (even if the key is very weak) whereas this is not necessarily the case in the non-`psk` handshake H'. For example, the first message of XX is a public key and a completely unencrypted payload. The second one also has a public key as a prefix, therefore a responder can mistake a second message for a first message. This voids agreement, even with a passive adversary. On the other hand, the first payload of XXpsk3 is encrypted with (more or less) the public ephemeral key of the initiator as secret key. This provides no guarantee against an active attacker, since the secret key is actually public, but prevents the message mismatch we just described. As a result, this unexpected form of tagging provides an agreement guarantee against a passive adversary. Overall, this phenomenon can be observed with KN, KX, NN, NX, XN, XX and their `psk` derivatives.

As a summary, when the pre-shared key is public, `psk` handshakes provide exactly the same guarantees as their non-`psk` counterparts when the pre-shared key is revealed, except for the first message of the patterns mentioned above.

About D_{pki} We consider a handshake where all public static keys are exchanged in pre-messages. There is a slight difference between the threat models $T = D_{rs}$ and $T' = D_{pki}$. In the former, the peer may receive a dishonest static key via pre-messages, whereas the latter forbids this. It happens that for the 53 patterns we analysed, no claim is falsified when replacing D_{pki} by D_{rs} in the threat model. We can verify that in Table B.4: no best threat model (which are given in disjunctive normal form) has a clause containing D_{pki} and not D_{rs}. This somewhat reduces the usefulness of D_{pki}.

Gradual protection improvements Let C be a secrecy or agreement claim for role r, handshake H and payload i. Let C' be the same claim as C but for payload $j > i$. It is always the case among the 53 tested patterns that whenever C holds, C' also holds. This can be easily machine-checked with Tables B.4 and B.5 by checking that $B(C') \rightharpoonup B(C)$. In other words, guarantees on the $i+1$-th message of a pattern are always at least as good as those on the i-th message.

3.1.2 Comparison of security levels with the Noise specification

We now consider the set of all 35 two-way non `psk` handshakes mentioned in the specification. A *location* is a pair (handshake, message index) where the message index is between 1 and the number of messages of the handshake plus two. As already mentioned, the Noise specification [14] associates to each location a source property (from 0 to 2, akin to agreement) and destination property (from 0 to 5, akin to secrecy for the sender). In this section we compare our results to those of the specification.

Security levels as equivalence classes The properties of the specification are tricky to formalize, so we will use the best threat model concept to automatically discover their meaning. Formally we are interested in the following claims: for each location, secrecy of the sender, and non-injective agreement of the recipient. We say that a secrecy claim has level n if the Noise specification attributes destination level n to the corresponding locations, and we say that a non-injective agreement claim has a level n if the Noise specification attributes source level n to the corresponding location.

As outlined by Proposition 3, the best threat model fully describes which threat models in $\bar{E}$ are compatible with a claim. In this respect, if two claims have the same best threat model, they can be regarded as providing the same amount of security. So we will quotient the set of claims by the equivalence relation: claims are equivalent if they have the same best threat model.

We found that claims with different security levels are never equivalent, but one security level covers different equivalence classes. That is, our approach is more fine-grained than the specification. Furthermore, we can intentionally drop precision: if we assume that ephemeral cannot be revealed and that the pre-message PKI is dishonest, then we find that equivalence classes match exactly security levels, except for source level 0. Let us define this formally.

Definition 50. For $A, B \in \bar{E}$ we write $A \equiv B$ if $(\neg R_e \mathbin{\dot\wedge} \neg R_{re} \mathbin{\dot\wedge} D_{pki}) \rightharpoonup (A \leftrightarrow B)$. We define the *simplified best threat model* of a claim C (denoted as $B'(C)$) as the equivalence class of $B(C)$ by $\equiv$. By abuse of notations, for C and C' two claims, we write $C \equiv C'$ if $B(C) \equiv B(C')$.

We now consider the quotient of the set of claims by the relation $\equiv$. Intuitively, the simplified best threat model is the best threat model we would have obtained if we had not considered ephemeral reveals and if we had assumed that the pre-message PKI is dishonest. In other words, it can be thought as a best threat model on the set of exclusions $E \setminus \{R_e, R_e^<, R_{re}, R_{re}^<, D_{pki}\}$. The equivalence classes are claims which have the same simplified best threat model. We found that this quotient matches nearly perfectly the classification in levels by the Noise specification. More specifically, for each source of destination level n except source level 0, there exists a formula ℓ_n such that a claim C has level n if, and only if, $B'(C) \equiv \ell_n$. Table 3.1 gives the values of ℓ_n.

Source level 0 is still split in four classes which we will call 0a, 0b, 0c and 0d, as shown in Table 3.2. Class 0a corresponds to cases where non-injective agreement is never guaranteed. This happens for the first message of the handshake when the second message unifies with the first one (see Paragraph "Dummy pre-shared keys" page 64) or when the initiator receives the responder's static key in pre-messages. Class 0b means that agreement only holds against a passive adversary. This happens when the first payload is not encrypted, or encrypted with a public key. Class 0d is interesting when it applies to a message coming from an anonymous agent. In this case, the recipient of the payload has the guarantee that this payload was sent by his peer: either the peer is honest and then the payload was not tampered with, or the payload was manipulated but in this case the peer's ephemeral key belongs to the adversary. This is a weak, but arguably non-zero guarantee. For example, it applies to the transport messages of NN (approximately an unauthenticated Diffie Hellman key exchange). The same applies for class 0c,

Level	ℓ_n
Destination 0	$\ell_{D0} = \top$
Destination 1	$\ell_{D1} = \mathit{active} \mathbin{\dot\wedge} D_{re}$
Destination 2	$\ell_{D2} = R_{rs} \mathbin{\dot\vee} (D_{rs} \mathbin{\dot\wedge} \mathit{active})$
Destination 3	$\ell_{D3} = \mathit{active} \mathbin{\dot\wedge} D_{re} \mathbin{\dot\wedge} (D_{rs} \mathbin{\dot\vee} R_{rs})$
Destination 4	$\ell_{D4} = \mathit{active} \mathbin{\dot\wedge} D_{re} \mathbin{\dot\wedge} (D_{rs} \mathbin{\dot\vee} R^{<}_{rs} \mathbin{\dot\vee} (R^{<}_{s} \mathbin{\dot\wedge} R_{rs}))$
Destination 5	$\ell_{D5} = \mathit{active} \mathbin{\dot\wedge} D_{re} \mathbin{\dot\wedge} (D_{rs} \mathbin{\dot\vee} R^{<}_{rs})$
Source 1	$\ell_{S1} = \mathit{active} \mathbin{\dot\wedge} (R^{<}_{s} \mathbin{\dot\vee} (D_{re} \mathbin{\dot\wedge} (R^{<}_{rs} \mathbin{\dot\vee} D_{rs})))$
Source 2	$\ell_{S2} = \mathit{active} \mathbin{\dot\wedge} D_{re} \mathbin{\dot\wedge} (R^{<}_{rs} \mathbin{\dot\vee} D_{rs}))$

Table 3.1: Interpretation of the source and destination levels of the Noise specification in terms of secrecy and non-injective agreement.

Level	ℓ_n
Source 0a	$\top$
Source 0b	active
Source 0c	$\mathit{active} \mathbin{\dot\wedge} (D_{re} \mathbin{\dot\vee} R^{<}_{s})$
Source 0d	$\mathit{active} \mathbin{\dot\wedge} D_{re}$

Table 3.2: Source level 0 still corresponds to fours equivalence classes.

.

but this time some protection relies on the recipient's static key. This applies for example to the first message of NK.

Table 3.1 can be used to give unambiguous, formally proven definitions for the levels in the specification. They hold, again, under the assumption that ephemerals cannot be revealed.

Destination 0 Secrecy of the payload never holds.

Destination 1 Secrecy of the payload holds unless the adversary is active and the recipient's ephemeral was generated by the adversary.

Destination 2 Secrecy of the payload holds unless (*i*) the recipient's static key is revealed, or (*ii*) the adversary is active and the recipient's static key was generated by the adversary.

Destination 3 Secrecy of the payload holds unless the adversary is active, the recipient's ephemeral key was generated by the adversary and (*i*) either the recipient's static key is revealed, or (*ii*) the recipient's static key was generated by the adversary.

Destination 4 Secrecy of the payload holds unless the adversary is active, the recipient's ephemeral key was generated by the adversary and (*i*) the recipient's static key is revealed before the message is sent, or (*ii*) the recipient's static key is revealed at any time and the sender's static key is revealed before the message is sent, or (*iii*) the recipient's static key was generated by the adversary.

Destination 5 Secrecy of the payload holds unless the adversary is active, the recipient's ephemeral key was generated by the adversary and (*i*) either the recipient's static key is revealed before the message is sent, or (*ii*) the recipient's static key was generated by the adversary.

Figure 3.2: We consider an upgrade from message 2 of IX to message 3 of X1X.

Source 1 The peers agree non-injectively on this payload, the session identifier and the public keys they both know unless the adversary is active and (*i*) the recipient's static key is revealed before the message is sent, or (*ii*) all the sender's asymmetric keys were generated by the adversary, or (*iii*) the sender's ephemeral key was generated by the adversary and his static key is revealed before the message is received.

Source 2 The peers agree non-injectively on this payload, the session identifier and the public keys they both know unless the adversary is active and (*i*) all the sender's asymmetric keys were generated by the adversary, or (*ii*) the sender's ephemeral key was generated by the adversary and his static key is revealed before the message is received.

This analysis has allowed us two discover to typographical errors[2] in Table 18.2 of the specification [14].

Conclusion As a summary, payload security levels presented by the Noise specification are consistent with our results, though less finely grained. Our results subsume the classification in levels if we forbid ephemeral reveals and assume that pre-messages come from a non-particularly trustworthy medium. This suggests the authors of the specification assumed implicitly that ephemerals could not be revealed. This assumption is only explicit for identity hiding properties and destination level 5. It would probably be beneficial to make it explicit for the whole table of source and destination properties in the specification.

3.1.3 Monotonicity of security levels

Let us first give a slightly contrived example situation.

Example 35. David wants to design a protocol conveying one payload from Alice to Bob. Only the secrecy of one payload from the point of view of Alice really matters; both Alice and Bob have a (non pre-shared) static key and no pre-shared symmetric key is available. David wants to choose which pattern to use to obtain secrecy of the sensitive payload.

David has enough confidence in the network to exclude any active attack. He is also very confident in Bob's PRNG quality for ephemeral keys, but Alice's device is headless and frequently runs out of entropy, so he would like to consider the scenario where Alice's ephemeral is revealed (read: can be predicted). Obviously he cannot avoid the scenario where one party is fully revealed; he excludes it. This is encoded in the following threat model: $T = (R_e \dot{\wedge} R_s) \dot{\vee} R_{re} \dot{\vee} active$.

It happens that when Alice is the responder, the second message of IX (see Figure 3.2) fulfils David's needs. Formally, we have: $T \vDash$ Secrecy IX Responder 2. This message is labelled as level 3. David would expect to be able to safely change his mind and send the sensitive payload as message 3 of X1X, which is labelled level 5. More generally David can reasonably expect to be able to use any level 4 or 5 location to send his sensitive payload, *i.e.* for all secrecy claim C of level 4 or 5 $T \vDash C$. We call this intuitive expectation monotonicity of security levels.

[2]Reported in `https://moderncrypto.org/mail-archive/noise/2019/002002.html`.

In our example, it happens that David cannot use the 3rd message of X1X (level 5) with his threat model. As per Tables B.4 and B.5, the best threat model of X1X initiator 3 is:

$$\begin{aligned} B(\text{Secrecy X1X initiator 3}) = &(D_{re} \mathbin{\dot\wedge} D_{rs} \mathbin{\dot\wedge} \mathit{active}) \\ &\dot\vee (D_{rs} \mathbin{\dot\wedge} \mathit{active} \mathbin{\dot\wedge} R_{re}^{<}) \\ &\dot\vee (D_{re} \mathbin{\dot\wedge} \mathit{active} \mathbin{\dot\wedge} R_{rs}^{<}) \\ &\dot\vee R_e \dot\vee (R_{re} \mathbin{\dot\wedge} R_{rs}) \end{aligned}$$

We can see that there is an attack where the attacker just has to reveal Alice's ephemeral. Therefore, under T this payload is not secret with this protocol.

To sum up, the "upgrade" from a level 3 message to a level 5 message has broken secrecy in David's threat model. This is an example of non monotonicity of security levels. △

General case Does monotonicity hold between some levels? That is, from which levels is it always safe to upgrade?

We only consider the case of secrecy, because agreement claims are not really comparable with each other: an agreement claim with an anonymous peer is not really comparable to another with a normal peer, for example.

Definition 51 (Monotonicity of destination security levels)**.** Let $m < n$ be two security levels. Let l_p be the set of claims of the form "secrecy of the sender for some location" where this location is identified as level p in the Noise specification. We say that level n is better than m if

$$\forall C^- \in l_m, \forall T \in \bar{E}, T \vDash C^- \Rightarrow \forall C^+ \in l_n, T \vDash C^+$$

Monotonicity means that if $m < n$ then n is better than m.

Lemma 17. *n is better than m if, and only if,*

$$\left(\mathop{\breve{\bigvee}}_{C^+ \in l_n} B(C^+) \right) \rightharpoonup \mathop{\breve{\bigwedge}}_{C^- \in l_m} B(C^-)$$

Proof. By equivalences:

$$\begin{aligned} &\forall C^- \in l_m, \forall T \in \bar{E}, T \vDash C^- \Rightarrow \forall C^+ \in l_n, T \vDash C^+ \\ &\forall C^- \in l_m, \forall T \in \bar{E}, B(C^-) \rightharpoonup T \Rightarrow \forall C^+ \in l_n, B(C^+) \rightharpoonup T && \text{by Proposition 3} \\ &\forall C^- \in l_m, \forall T \in \bar{E}, B(C^-) \rightharpoonup T \Rightarrow \left(\mathop{\breve{\bigvee}}_{C^+ \in l_n} B(C^+) \right) \rightharpoonup T && \text{by Proposition 2} \\ &\forall C^- \in l_m, \left(\mathop{\breve{\bigvee}}_{C^+ \in l_n} B(C^+) \right) \rightharpoonup B(C^-) \\ &\left(\mathop{\breve{\bigvee}}_{C^+ \in l_n} B(C^+) \right) \rightharpoonup \mathop{\breve{\bigwedge}}_{C^- \in l_m} B(C^-) && \text{by Proposition 2} \end{aligned}$$

□

With the criterion of Lemma 17, and the best threat models of all the patterns (Tables B.4 and B.5), we can easily machine check which levels are better than others. Here are the results: When $n > m > 0$, n is better than m if, and only if, $(n, m) = (4, 1)$. We identified two reasons explaining this result.

The first reason is that some patterns can arguably only be used in different use cases. For example, NX has one agent with a non-preshared static key and one agent with no static key at all, while KK has two agents having a pre-shared static public key. The core of the fictitious scenario of Example 35 was that we could imagine that protocol designers would want to "upgrade" from say a level 3 message to a level 5 message. But it is unlikely that one would want to upgrade from NX to KK. Intuitively, one always gets a better security and lower latency by pre-sharing keys. If the designer could not afford pre-sharing a key with NX, why could this be suddenly possible for him to use KK? For this reason we could restrict our study to upgrades from low level patterns to higher level patterns where the source and destination patterns are "alike". Formally, we will only consider upgrades which keep the following invariants:

- the set of keys known after the claim to the sender r of the message;
- whether r has no static key, a pre-shared static key, or a non-pre-shared static key;
- whether $\tilde{r}$ has no static key, a pre-shared static key, or a non-pre-shared static key.

Example 35 features an upgrade from the 2^{nd} message of IX to the 3^{rd} message of X1X. In the sense above, this upgrade is valid. Even with this restriction, we do not get full monotonicity. Level 3 is still not better than 1; level 4 is not better than 3; and level 5 is not better than 1, 3, and 4.

Example 35 suggests there is a second reason for the lack of monotonicity: we consider ephemeral reveals. If we simplify the best threat model as in Section 3.1.2, then we get monotonicity between all levels except that 2 is not better than 1. If additionally we apply the restriction of the previous paragraph, then we get full monotonicity.

Conclusion Monotonicity of security levels is a property the casual reader will probably take for granted, notably because a location is only said to be "level 5" and not "level 5, 3, 2 and 1". Yet, this property does not hold, notably if we consider ephemeral reveals. This suggests once more that this is a tacit assumption of the whole Section 7.7 of the specification [14] which should be made explicit.

3.1.4 Comparison of our results with Noise explorer

Noise Explorer [11] is an automated tool based on ProVerif [5] which can classify Noise patterns by their source and destination properties. It not only supports verifying the 5 destination levels, and the 2 source levels of the Noise specification [14], but it has also extended them in two ways.

First, two new source levels are added, 3 and 4. The original source levels 1 and 2 are modelled as a weak version of non-injective agreement: if Alice receives a message from Bob, then Bob must have sent this same message to *someone*. Source levels 1 and 2 differ by their threat model (see Table 3.1). The new levels Noise explorer added are the equivalent of level 1 and 2 but with standard non injective agreement instead of the weaker version mentioned above.

The second addition is that PSK handshakes are supported, and the definition of the corresponding levels is adapted in consequence. When level n requires a property to hold unless some key is revealed at any time (respectively, before the end of the session) for a non-PSK handshake, it is required to hold unless this key *and the PSK* are revealed at any time (respectively before the end of the session) for a PSK-handshake.

```
XK:
  <- s
  ...
  -> e, es
  <- e, ee
  -> s, se
```

```
XKpsk3:
  <- s
  ...
  -> e, es
  <- e, ee
  -> s, se, psk
```

Figure 3.3: Noise explorer associates destination level 2 to the first message of XK but level 0 to the first message of XKpsk3, although they have the same best threat model.

```
NX:
  -> e
  <- e, ee, s, es
```

```
I1X:
  -> e, s
  <- e, ee, s, es
  -> se
```

Figure 3.4: Noise explorer associates source level 2 to the first message of NX but level 4 to the second message of I1X, but they have the same best threat model.

Like in Section 3.1.2 we quotient the set of locations by the equality of best threat model. If two claims have the same best threat model, then they have the same security level according to Noise Explorer except in two cases.

Non PSK-protected messages in PSK handshakes The description of level 2 for secrecy of the first message of XKpsk3 is "[whether] an active attacker is able to retrieve the payload plaintext only by compromising Bob's static key and PSK either before or after the protocol session"[3]. This does not hold because the PSK is unused in XKpsk3 until the 3$^{\text{rd}}$ message. For this reason, this message is labelled level 0. The corresponding message in XK has exactly the same properties (the same best threat model) but is labelled level 2 because this time the description of level 2 is "[whether] an active attacker is able to retrieve the payload plaintext only by compromising Bob's static key either before or after the protocol session"—no mention of the PSK. This is illustrated in Figure 3.3. More generally, Noise Explorer attributes level 0 to many messages preceding the first PSK token of a pattern, even when they provide guarantees which are enough for a non-zero level in non-PSK handshakes.

Non-injective agreement for an anonymous agent Noise Explorer defines the identity of an agent as its static key. If a pattern does not use this static key (*i.e.*, one of the peers only has an ephemeral key), then non-injective agreement cannot hold: an agent cannot distinguish between Bob or Charlie as peer. On the other hand, the weak version of agreement Noise Explorer checks for levels 1 and 2 holds. For anonymous agents, we use as identity their ephemeral key. As a result, our flavour of non-injective agreement can hold even for anonymous agents. The associated guarantee is that if an agent agrees with an anonymous peer for two consecutive messages, then necessarily this peer is the same anonymous agent for the two messages. Such "mismatches" are illustrated in Figure 3.4.

We can do the same construction as for Table 3.1 with the levels of Noise Explorer. More specifically, for any destination level n, we determine the set $L_n \subseteq \bar{E}/\equiv$ of the simplified best threat models of the claims labelled as level n. The results are given in Table 3.3. Unlike for the levels of the specification, the correspondence between Noise Explorer levels and equivalence

[3]Source: `https://noiseexplorer.com/patterns/XKpsk3/A.html`.

Level	L_n
Destination 0	$\{\top, \ell_{D2}\}$
Destination 1	$\{\ell_{D1}\}$
Destination 2	$\{R_{psk},\ \ell_{D2},\ \ell_{D2} \mathbin{\dot\wedge} R_{psk}\}$
Destination 3	$\{\ell_{D3}, \ell_{D1} \mathbin{\dot\wedge} R_{psk}, \ell_{D1} \mathbin{\dot\wedge} R^{<}_{psk}, \ell_{D3} \mathbin{\dot\wedge} R_{psk}, \ell_{D3} \mathbin{\dot\wedge} R^{<}_{psk}, \ell_{D4} \mathbin{\dot\wedge} R_{psk}, \ell_{D5} \mathbin{\dot\wedge} R_{psk}, \ell_{D5} \mathbin{\dot\wedge} R^{<}_{psk}\}$
Destination 4	$\{\ell_{D4}, \ell_{D4} \mathbin{\dot\wedge} R^{<}_{psk}\}$
Destination 5	$\{\ell_{D5}, \ell_{D5} \mathbin{\dot\wedge} R^{<}_{psk}\}$
	$\ell_{D1}, \ell_{D2}, \ldots$ are defined in Table 3.1.

Table 3.3: Simplified best threat models covered by Noise Explorer destination levels

classes is not perfect. Most levels cover several equivalence classes. An interesting observation is that some secrecy claims have simplified best threat model $A = \ell_{D5}$, $B = D_{re} \mathbin{\dot\wedge} R^{<}_{psk}$ or $C = \ell_{D5} \mathbin{\dot\wedge} R^{<}_{psk}$, where ℓ_{D5} is defined in Table 3.1 as $\ell_{D5} = D_{re} \mathbin{\dot\wedge} (D_{rs} \mathbin{\dot\wedge} R^{<}_{rs})$. A encodes a form of perfect forward secrecy relying only on asymmetric keys. B encodes a form of perfect forward secrecy relying only on the pre-share symmetric key. C encodes a form or perfect forward secrecy relies on both asymmetric and symmetric keys: it can only be falsified if both a static key and the pre-shared key are revealed before the claim. These nuances are made visible by our tool. Only A and C are labelled as level 5 ("strong forward secrecy") by Noise Explorer whereas B is level 3 ("forward secrecy", against a passive attacker) along with best threat models like ℓ_{D3} which clearly do not encode perfect forward secrecy.

Note that the fact that some claims with simplified best threat model $C = \ell_{D5} \mathbin{\dot\wedge} R^{<}_{psk}$ are labeled level 3 while others are labelled level 5 is likely to be an error. Noise Explorer emits a query of the form "secrecy unless $\perp$" instead of, for example, "secrecy unless my peer's static key and the PSK are revealed before the claim" to test destination level 4 of messages sent by an anonymous agent in PSK handshakes. This has been reported to the authors of Noise Explorer who answered that a proper formalism for these queries was still a work in progress.[4]

This illustrates that choosing the right formal definition for these "security levels" is not easy. The security levels of the Noise specification act as thresholds. By choosing too strict thresholds, one can miss that some weaker, but nevertheless valuable, properties also hold; and by choosing too low thresholds, one misses that some patterns provide better guarantees. Some of the mismatches between our results and those of Noise Explorer illustrate this point: extending the levels of the Noise specification to PSK handshakes is not trivial. Our method has the advantage that we do not need to define thresholds since we test all possible threat models in $\bar{E}$. By computing the best threat model of the claims we need to classify, the relevant thresholds are discovered naturally. This how we rediscovered the levels of the Noise specification in Table 3.1, and how we can refine the classification of Noise Explorer in table Table 3.3.

Conclusion Apart from the small mismatches highlighted above, our results are mostly compatible with Noise Explorer's and offer new insight that was not completely visible with Noise Explorer's classification in levels.

[4] `https://github.com/SymbolicSoft/noiseexplorer/issues/9`

3.2 Identity hiding properties

Identity hiding properties are modelled by a notion of anonymity which can be expressed as "a transcript where Alice communicates with Charlie is observationally equivalent to a transcript where Bob communicates with Charlie". Such proofs are computationally very expensive, which forced us to make a number of assumptions. These assumptions have already been covered in Section 2.2.6 but let us recapitulate them here.

Like for trace properties, we assume that private keys are always honestly generated and that when agents receive a public key, they check that these are valid group elements. Additionally, for performance reasons, we assume that there is at most one initiator and one responder, that the pre-message PKI is always honest, that ephemeral keys cannot be revealed, and that Charlie always receives honest static keys. We also restrict the set of exclusions we consider to $E_a = \{R_s, R_{rs}, R_{psk}, active\}$.

Because our model is symbolic, the adversary has access to a list (or directory) of all existing public keys. As a result, an active adversary can always try to start a session with diff(Alice, Bob) for all public keys in this directory and check which key is accepted. We will call this a directory attack. The existence of this attack falsifies anonymity for a large number of patterns, so to have useful results we exclude them by assuming that diff(Alice, Bob) aborts the handshake if their peer's static is dishonestly generated.

Likewise, if the session identifier mentioned in Section 11.2 of the specification [14] is leaked by the application, then anonymity is falsified for all non-`psk` handshakes at least. To obtain useful results, we assume that this session identifier is kept private.

We considered the 53 handshakes mentioned in the Noise specification [14]. Results are available in Tables B.1, B.2 and B.3. Despite all the extra assumptions we make, not all proofs completed and these results are only partial.

3.2.1 Types of attacks

Tamarin is sound but not complete with observational equivalence proofs. This means that some attacks Tamarin finds may be spurious. As a result, we had to review by hand all attacks Tamarin found. Fortunately, there were very few of them. The reason is that most attacks match the under-approximation of Lemma 16. Since propositions which are proved to be false with this criterion are not submitted to Tamarin, we do not have to review them.

As a result, we have established an informal classification of attacks against anonymity. The first two are statically excluded from our model, the third can be detected without Tamarin, the next ones were found by Tamarin.

Active directory attack As already mentioned, if the adversary can establish a session as the legitimate peer of diff(Alice, Bob), then he can test possible keys until diff(Alice, Bob) accepts to complete a handshake.

Public session identifier If a pattern does not involve the token `psk` or the pre-shared symmetric key is revealed, and the session identifier is leaked by either party, then a passive adversary can falsify anonymity. The reason is that this session identifier is a public function of the static key of diff(Alice, Bob) and the pre-shared symmetric key, so the attacker can test all static keys until he finds the right one.

Key usability This class of attacks happens when threat model is so permissive that Lemma 16 applies. See its proof for a precise description.

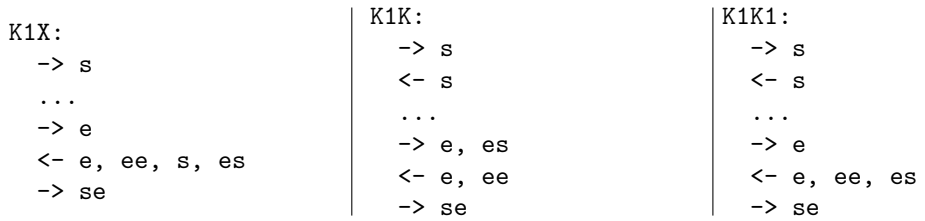


Figure 3.5: The anonymity of the initiator of K1X, K1K and K1K is falsified by knowledge extraction from the peer.

Knowledge extraction from the peer This name denotes a type of falsification of the anonymity lemmas. Arguably, this is not an attack against the anonymity of diff(Alice, Bob) but it is nevertheless an identity leakage. It can only be performed on some handshake patterns where Charlie has received the identity of their peer diff(Alice, Bob) in pre-messages.[5] The active attacker starts a session with Charlie and will discover with whom Charlie expected to speak—that is, the identity of diff(Alice, Bob). Interestingly, diff(Alice, Bob) does *not* need to be alive for his identity to be revealed.

Patterns where Charlie receives the key of their peer in pre-messages do not also transmit it by the network, so if Charlie was expecting a message from several possible peers, he has no reasonable way of determining which one is attempting to start a session with him besides brute force. For example, the first message of K1K1 (see Figure 3.5) is a public ephemeral key and a clear-text payload: the responder has no way to determine the static public key of the initiator from this information. Yet, he needs this value to send a valid second message. This confirms that these patterns were designed with the assumption that a given responder Charlie will only start sessions with one single peer ever. For this reason, after the attacker performs this attack, if Charlie establishes a new session with a peer, the attacker can rightfully suspect that this peer is diff(Alice, Bob) whose identity he knows.

Here are the specific details of the attack. It is performed by an active adversary against the identity of the potential initiator of K1X, K1K and K1K1 (see Figure 3.5) and does not require that the attacker use a static key. We explain the attack on K1K1. The attacker abides by the protocol for the first two messages of the handshake. Let e be the attacker's ephemeral key, and e' and s' be the ephemeral and static keys of the responder Charlie. Let $S = \mathrm{diff}(S_A, S_B)$ be the static key of diff(Alice, Bob), *i.e.* what the attacker wants to discover. Let the p_i be payloads. The attacker sends the message $\langle g^e, p_1 \rangle$. Charlie answers a message which could be approximated by $\langle g^{e'}, \mathsf{aead}(k, 0, h, p_2) \rangle$ where $k = \langle g^{ee'}, g^{es} \rangle$ and $h = \langle g^s, g^S, g^e, g^{e'} \rangle$. The attacker knows k because he knows e, so he can compute $v = \mathsf{verif}(k, 0, \langle g^s, g^{S_A}, g^e, g^{e'} \rangle, \mathsf{aead}(k, 0, h, p_2))$. We have: $v = \top$ if, and only if, diff(Alice, Bob) is Alice.

The attack on KK1 is very similar, and the one on K1X is even simpler: with the same notations as above, the attacker sends the first message $\langle g^e, m \rangle$ where $m = \mathsf{aead}(g^{es'}, 0, \langle g^e, g^{s'}, g^{S_a} \rangle, p_1)$. This message will be accepted by Charlie if, and only if, $S = S_A$.

The four types of attacks above are the only one we found if reveals of diff(Alice, Bob) (exclusion R_s) are disallowed. If we allow them, Tamarin finds some more instances of knowledge extraction from the peer. What these attacks perform is then: given a private key, an attacker can figure out if an agent Charlie expected a message from the owner of this key. Namely, the attack of K1K1 extends easily to the anonymity of the initiator of KX and KK1 (see Figure 3.6). Similarly, the attack on KX1 extends to KK (see Figure 3.6). This also applies to the derivatives

[5] Note that in our modelisation, pre-messages are perfectly confidential and authenticated.

```
KK1:
  -> s
  <- s
  ...
  -> e
  <- e, ee, se, es
IK1:
  <- s
  ...
  -> e, s
  <- e, ee, se, es
```

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

Figure 3.6: Some patterns which become vulnerable to knowledge extraction from the peer if diff(Alice, Bob) is revealed.

```
KN:
  -> s
  ...
  -> e
  <- e, ee, se
```

```
NK1:
  <- s
  ...
  -> e
  <- e, ee, es
```

```
NX:
  -> e
  <- e, ee, s, es
```

Figure 3.7: The initiator or KN, and the responder of NK1 and NX are all anonymous unless the adversary is active (best threat model *active*), but are associated to levels 7, 9 and 3 respectively.

KKpsk2 and, if additionally the pre-shared key is revealed, KKpsk0, as well as IK, and its `psk` derivatives.

IK1 (see Figure 3.6) is also vulnerable to a variant of this attack. If diff(Alice, Bob) is the responder and is revealed, then an attacker can impersonate the responder and check whether Charlie, the initiator, accepts the second message. This situation is slightly different from the previous ones because Charlie is the initiator: the attacker uncovers the identity of agent with whom Charlie was actively trying to establish a session.

3.2.2 Comparison with the Noise specification

We now consider the 15 two-way handshake patterns of Section 7.8 of the Noise specification [14]. Some pairs (pattern, role) are associated to an identity hiding property between 0 and 9. We can associate to each of these pairs an anonymity claim, and then use Table B.1 of their best threat model.

As we did with agreement and secrecy claims, we consider the equivalence relation: two claims are equivalent when they have the same best threat model. This time, our results are compatible but not completely comparable to those of the specification.

In some cases, the specification is more precise than our analysis. For example the identity of the responder of NX and NK1 (see Figure 3.7) are protected at level 1 (the second lowest level) and 9 (the highest level) respectively. Level 1 can be paraphrased as "an active attacker can kindly request his static key from the responder" and level 9 as "an active attacker can establish a session with the responder and then, later, can use a directory of public keys to find which one was the responder's". In both cases, the best threat model is *active*: anonymity holds if, and only if, the adversary is passive. Our analysis is unable to distinguish that in one case a directory of public keys is necessary and not in the other.

In some other cases, though, the best threat model carries more information than the description of the corresponding level. For example, the identity of initiator of KN (see Figure 3.7) is protected at level 7:

> An active attacker who pretends to be the initiator without the initiator's static private key, then later learns a candidate for the initiator private key, can then check whether the candidate is correct.

The attack described involves an active attacker and a reveal. Yet, the best threat model for the anonymity of the initiator of KN is *active*, which means that anonymity is falsified with an active adversary, even if no honest agent is revealed. Indeed, in this situation, Lemma 16 applies. With the usual notations, the initiator is diff(Alice, Bob) and his static key is diff(g^{S_A}, g^{S_B}). His ephemeral key is g^e; the attacker's is $g^{e'}$. Let p_i be payloads. The initiator sends a message $\langle g^e, p_1\rangle$ to an honest party. The attacker intercepts it, and answers with $\langle g^{e'}, \mathsf{aead}(k, 0, h, p_2)\rangle$ where $k = \langle g^{ee'}, g^{S_A e'}\rangle$ and $h = \langle g^e, g^{e'}, g^{S_A}\rangle$. This message will be accepted if, and only if, diff(Alice, Bob) is in fact Alice.

Finally, let us repeat here that our analysis shows that if the session identifier mentioned in Section 11.2 of the Noise specification [14] is public, then even a passive attacker can falsify anonymity of agents involved in the corresponding session. As a result, the specification might benefit from requiring applications to treat the session identifier as a secret value when anonymity is a desired property.

Conclusion

The main contribution of this work is an automated tool able to verify, for any two-way Noise handshake pattern, a number of security properties. The supported properties are secrecy of payloads from the point of view of the sender and the receiver, and injective and non-injective agreement on messages. For any of these properties, the tool can determine in which of billions of threat models it holds, and summarize this information in a concise way which highlights minimal necessary assumptions for every possible attacks. To a lesser extent, the tool can also provide similar results for the anonymity of agents.

As a proof of concept, we apply this tool to 53 handshake patterns mentioned in the Noise specification, which have already been studied in the existing literature. In the cases of secrecy and agreement, our results subsume and refine a large part of existing security analyses.

Existing work focused on verifying a set of properties called "security levels" in the Noise specification [14]. By exploring billions of threat models, we showed how these levels can be expressed as the standard properties mentioned above, and that, at least for non-PSK handshakes, this classification in levels emerges naturally and is mostly accurate. In other words, we justify formally the choice of the security levels of the Noise specification.

Still, limitations apply. Notably, our modelling of anonymity has a number of restrictions which are only justified by the necessary performance improvements they bring. For example, we assume that there is at most one initiator and one responder, and that the peer of the test agent only accepts honestly generated static keys. Lifting these limitations is possible for some patterns only as of now; doing so for any pattern probably requires improvements in the Tamarin prover such as induction support in diff mode.

While the support for agreement and secrecy is more complete than that of anonymity, there is still room for improvement. As part of future work, it would notably be interesting to introduce an exclusion denoting that the adversary has not interfered with the test session. This would enable us to model weak forward secrecy. Currently, this can only be approximated either by making the adversary passive or by having test agents refuse dishonestly generated keys.

Finally, let us point out that Noise is still a work in progress. Various extensions of the language of patterns are currently being discussed. Our tool would have to be adapted to reflect these changes. It would nevertheless be unlikely to become obsolete, as current patterns are expected to remain valid.

Appendices

Appendix A

Table of notations

In general, operators decorated with a $\breve{}$ are for trace formulas and $\dot{}$ for exclusion-based formulas. $\vdash, \Vdash, \ldots$ denote syntactic statements whose truth value is denoted as $\vDash, \Vdash, \ldots$

Table A.1: Notations

Notation	Description	Reference
$\langle x, y \rangle$	pairing	
$[]$	empty sequence	
$\mathcal{T}$	set of terms	Section 1.3.1, page 13
$\mathcal{E}$	equational theory	Section 1.3.1, page 13
σ_i	usually, state after message i	Definition 11
$\hat{\sigma}$	knowledge associated to state σ	Definition 12
$l \xrightarrow{a} r$	labelled multiset rewriting rule	Definition 18
Out, In, K	Output, input, adversary knowledge	Section 1.4.2
Φ	set of trace formulas	Definition 23
Θ	Set of timepoint variables	Definition 23
$f@t$	action fact f happens at time t	Definition 23
$\approx$	equality in trace formulas	Definition 23
$\prec$	timepoint ordering	Definition 23
$\breve{\vee}$, $\breve{\wedge}$, $\rightsquigarrow$, $\leftrightsquigarrow$	disjunction, conjunction, implication, equivalence of trace formulas	Definition 23
$\tau \breve{\vdash} \phi$	statement that a trace τ satisfies a trace formula ϕ	Definition 23
$\tau \breve{\vDash} \phi$	truth value of $\tau \breve{\vdash} \phi$	Definition 23
$R \dot{\Vdash} \phi$	statement that a set of rules R satisfies a trace formula ϕ	Definition 25
$R \dot{\Vdash} \breve{\vDash} \phi$	truth value of $R \breve{\vdash} \phi$	Definition 25
$\mathcal{L}(R)$	left-hand side of a bi-system	Definition 28
$\mathcal{R}(R)$	right-hand side of a bi-system	Definition 28
$\mathrm{dgs}(R)$	set of dependency graphs of a system	Definition 29
$\mathrm{mirrors}(R)$	set of mirrors of a dependency graph	Definition 30
$\sim_{Env}$	dependency graph equivalence	Definition 31
$\tau(dg)$	trace corresponding to a dependency graph	Definition 32
$R\vert_{\phi}$	bi-system R restricted by trace formula ϕ	Definition 32
$\varphi(C, T)$	secrecy or agreement lemma	Definitions 34 and 35
$R(C)$	theory for $\varphi(C, T)$	Definitions 34 and 35

Table A.1: Notations (continued)

Notation	Description	Reference
E	set of exclusions	Definition 36
$active$	exclusion "the adversary is active"	Definition 36
D_{pki}	exclusion "the pre-message PKI contains a dishonestly generated key"	Definition 36
D_{rs}, D_{re}	exclusion "the peer has a dishonestly generated static or ephemeral key"	Definition 36
R_k	exclusion "key k is revealed"	Definition 36
$R_k^<$	exclusion "key k is revealed before the claim"	Definition 36
$E^\star$	transitive closure of disjunction and conjunction on exclusions	Definition 37
$\dot\vee$, $\dot\wedge$	disjunction and conjunction of exclusion	Definition 37
$[\![T]\!]_t$	trace semantics of T	Definition 38
E°	boolean formulas over exclusions	Definition 40
$\twoheadrightarrow$, $\twoheadleftarrow\!\!\twoheadrightarrow$	implication and equivalence in E°	Definition 40
$\Vdash$	entailment predicate on E°	Definition 40
$\rightharpoonup$, $\rightleftharpoons$	subsumption of threat models, corresponding equivalence relation	Definition 41
$\bar E$	set of the distinct threat models	Definition 42
E_a	set of exclusions for anonymity models	Definition 43
$[\![T]\!]_d$	diff semantics of a threat model T	Definition 43
$R_0(C)$	bi-system modelling anonymity claim C	Definition 44
$T \vdash C$	statement "claim C holds under threat model T"	Definitions 39 and 45
$T \vDash C$	truth value of $T \vdash C$	Definitions 39 and 45
$B_F(C)$	best threat model for C in $\bar F$	Lemma 8
$B(C)$	$B_E(C)$ for secrecy or agreement claims, or $B_{E_a}(C)$ for anonymity claims	Lemma 8

Appendix B

Best threat models of common Noise handshake patterns

In the following pages, the reader will find our actual results. We consider the 53 two-way handshake patterns mentioned in the Noise specification [14].

Results about anonymity are presented in Tables B.1, B.2 and B.3. Some anonymity proofs have not completed within a reasonable amount of time or memory. Since Tamarin is correct but not complete for observational equivalence, it makes sense to consider that anonymity lemmas which timed out were falsified. When this happens, we do not compute the best threat model $B_{E_a}(C)$ of the claim C, but an upper bound T such that $B_{E_a}(C) \rightharpoonup T$.

Table B.1 gives the best threat models of all anonymity claims whose proofs completed within the timeout. Table B.2 gives an upper bound of the best threat model of all anonymity claims where only proofs about clauses containing R_s timed out. This means that if one disallows reveals of diff(Alice, Bob), when one replaces R_s by $\bot$ in this upper bound, one gets the best threat model $B_{E_a \setminus \{R_s\}}(C)$ of the claim C over the exclusions $\{active, R_{rs}, R_{psk}\}$. Finally, Table B.3 gives an upper bound of the best threat model of anonymity claims which suffered timeouts on clauses without R_s.

Results for secrecy and agreement claims are presented in Tables B.4 and B.5. To each secrecy or agreement claim C on one of these patterns, table B.5 associates a *class index* c. Table B.4 associates class indices to the best threat model that all claims with class index c share. Class indices have been chosen to be a topological order compatible with $\rightharpoonup$. The structure of the set of best threat models of secrecy and agreement claims on non-PSK handshakes is presented in Figure B.1.

If a claim on one of the 53 handshakes we consider is missing of the following tables, it means that its best threat model is $\bot$, *i.e.* that this claim never holds.

The Tamarin proof files and source code of the tool to generate them are available at the address [8].

Table B.1: Best threat model for anonymity claims whose computation completed without timeouts

Pattern	Role	Class	Best threat model
I1K	Responder	3	$R_s \dot{\vee} active$
I1K1	Responder	9	$active$
I1X	Responder	9	$active$

Table B.1: Best threat model for anonymity claims without timeouts (continued)

Pattern	Role	Class	Best threat model
I1X1	Responder	9	$active$
IK	Responder	7	$R_s \dot{\vee} (active \dot{\wedge} R_{rs})$
IK1	Responder	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
IKpsk2	Responder	7	$R_s \dot{\vee} (active \dot{\wedge} R_{rs})$
K1K	Initiator	2	$R_{rs} \dot{\vee} active$
K1K	Responder	3	$R_s \dot{\vee} active$
K1K1	Initiator	9	$active$
K1K1	Responder	9	$active$
K1N	Initiator	9	$active$
K1X	Initiator	9	$active$
K1X	Responder	9	$active$
K1X1	Initiator	9	$active$
K1X1	Responder	9	$active$
KK	Initiator	11	$R_{rs} \dot{\vee} (active \dot{\wedge} R_s)$
KK	Responder	7	$R_s \dot{\vee} (active \dot{\wedge} R_{rs})$
KK1	Initiator	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
KK1	Responder	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
KKpsk0	Initiator	13	$(R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (active \dot{\wedge} R_s \dot{\wedge} R_{psk})$
KKpsk0	Responder	8	$(R_s \dot{\wedge} R_{psk}) \dot{\vee} (active \dot{\wedge} R_{rs} \dot{\wedge} R_{psk})$
KKpsk2	Initiator	11	$R_{rs} \dot{\vee} (active \dot{\wedge} R_s)$
KKpsk2	Responder	7	$R_s \dot{\vee} (active \dot{\wedge} R_{rs})$
KN	Initiator	9	$active$
KNpsk0	Initiator	1	R_{psk}
KX	Initiator	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
KX1	Initiator	9	$active$
NK	Responder	3	$R_s \dot{\vee} active$
NK1	Responder	9	$active$
NKpsk0	Responder	6	$(R_s \dot{\wedge} R_{psk}) \dot{\vee} (active \dot{\wedge} R_{psk})$
NKpsk2	Responder	3	$R_s \dot{\vee} active$
NX	Responder	9	$active$
NX1	Responder	9	$active$
NXpsk2	Responder	9	$active$
X1K	Responder	3	$R_s \dot{\vee} active$
X1K1	Responder	9	$active$
X1N	Initiator	9	$active$
X1X1	Responder	9	$active$
XK	Responder	3	$R_s \dot{\vee} active$
XK1	Responder	9	$active$
XKpsk3	Responder	3	$R_s \dot{\vee} active$
XN	Initiator	9	$active$
XNpsk3	Initiator	9	$active$
XX	Responder	9	$active$
XX1	Responder	9	$active$

Table B.2: Upper bound of the best threat model of anonymity claims where only R_s proofs timed out.
Replace R_s by $\bot$ in this value to obtain $B_{\{active,R_{rs},R_{psk}\}}(C)$, where C is the claim we consider.

Pattern	Role	Class	Upper bound
I1K	Initiator	11	$R_{rs} \dot{\vee} (active \dot{\wedge} R_s)$
IK	Initiator	11	$R_{rs} \dot{\vee} (active \dot{\wedge} R_s)$
IKpsk1	Initiator	12	$R_{rs} \dot{\vee} (active \dot{\wedge} R_s \dot{\wedge} R_{psk})$
IKpsk2	Initiator	11	$R_{rs} \dot{\vee} (active \dot{\wedge} R_s)$
IKpsk2	Initiator	11	$R_{rs} \dot{\vee} (active \dot{\wedge} R_s)$
IX	Responder	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
IX1	Responder	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
IXpsk2	Responder	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
IXpsk2	Responder	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
KX	Responder	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
KX1	Responder	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
KXpsk2	Responder	15	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s \dot{\wedge} R_{psk})$
X1K	Initiator	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
X1K1	Initiator	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
X1X	Responder	4	$(R_s \dot{\wedge} R_{rs}) \dot{\vee} active$
XK	Initiator	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
XK1	Initiator	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
XKpsk3	Initiator	15	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s \dot{\wedge} R_{psk})$
XX	Initiator	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
XX1	Initiator	14	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s)$
XXpsk3	Responder	5	$(R_s \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} active$

Table B.3: Upper bound of the best threat model of anonymity claims not in Tables B.1 or B.2.

Pattern	Role	Class	Upper bound
IKpsk1	Responder	7	$R_s \dot{\vee} (active \dot{\wedge} R_{rs})$
X1X	Initiator	4	$(R_s \dot{\wedge} R_{rs}) \dot{\vee} active$
X1X1	Initiator	9	$active$
XXpsk3	Initiator	10	$(active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_{psk})$

Table B.4: Best threat model equivalence classes

Class	Best threat model
0	$\top$
1	$R_{re} \dot{\vee} R_e \dot{\vee} R_{psk} \dot{\vee} D_{pki} \dot{\vee} D_{rs}$
2	$R_{rs} \dot{\vee} R_{re} \dot{\vee} R_e \dot{\vee} D_{pki} \dot{\vee} active \dot{\vee} D_{rs}$
3	$R_{re} \dot{\vee} R_s \dot{\vee} R_e \dot{\vee} D_{pki} \dot{\vee} active \dot{\vee} D_{rs}$
4	$R_{re} \dot{\vee} (R_s \dot{\wedge} R_{rs}) \dot{\vee} R_e \dot{\vee} D_{pki} \dot{\vee} active \dot{\vee} D_{rs}$
5	$R_{re} \dot{\vee} R_e \dot{\vee} (R_s \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} D_{pki} \dot{\vee} active \dot{\vee} D_{rs}$
6	$R_{re} \dot{\vee} R_e \dot{\vee} (R_s \dot{\wedge} R_{psk}) \dot{\vee} D_{pki} \dot{\vee} (active \dot{\wedge} R_{psk}) \dot{\vee} D_{rs}$
7	$R_{re} \dot{\vee} R_s \dot{\vee} R_e \dot{\vee} D_{pki} \dot{\vee} (active \dot{\wedge} R_{rs}) \dot{\vee} D_{rs}$
8	$R_{re} \dot{\vee} R_e \dot{\vee} (R_s \dot{\wedge} R_{psk}) \dot{\vee} D_{pki} \dot{\vee} (active \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} D_{rs}$

Table B.4: Best threat model equivalence classes (continued)

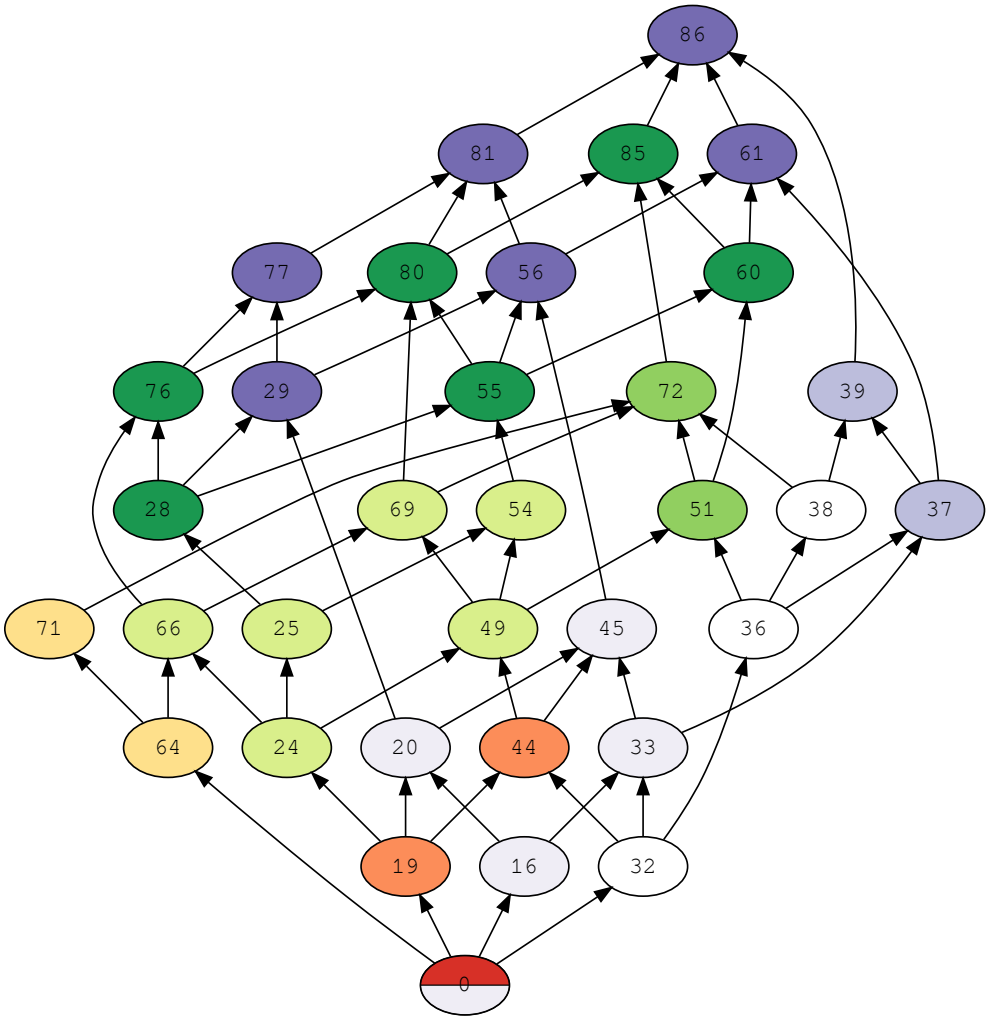
Class	Best threat model
9	$R_{re} \dot{\vee} R_e \dot{\vee} D_{pki} \dot{\vee} active \dot{\vee} D_{rs}$
10	$R_{re} \dot{\vee} R_e \dot{\vee} D_{pki} \dot{\vee} (active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_{psk}) \dot{\vee} D_{rs}$
11	$R_{rs} \dot{\vee} R_{re} \dot{\vee} R_e \dot{\vee} D_{pki} \dot{\vee} (active \dot{\wedge} R_s) \dot{\vee} D_{rs}$
12	$R_{rs} \dot{\vee} R_{re} \dot{\vee} R_e \dot{\vee} D_{pki} \dot{\vee} (active \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} D_{rs}$
13	$R_{re} \dot{\vee} R_e \dot{\vee} (R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} D_{pki} \dot{\vee} (active \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} D_{rs}$
14	$R_{re} \dot{\vee} R_e \dot{\vee} D_{pki} \dot{\vee} (active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s) \dot{\vee} D_{rs}$
15	$R_{re} \dot{\vee} R_e \dot{\vee} D_{pki} \dot{\vee} (active \dot{\wedge} R_{rs}) \dot{\vee} (active \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} D_{rs}$
16	$active$
17	R_{psk}
18	$active \dot{\wedge} R^<_{psk}$
19	$R_{re} \dot{\vee} R_e \dot{\vee} D_{re}$
20	$(active \dot{\wedge} R^<_{re}) \dot{\vee} (active \dot{\wedge} R^<_e) \dot{\vee} D_{re}$
21	$(R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R_{psk})$
22	$(R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^<_{psk})$
23	$(active \dot{\wedge} R^<_{re} \dot{\wedge} R^<_{psk}) \dot{\vee} (active \dot{\wedge} R^<_e \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^<_{psk})$
24	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} R_e \dot{\vee} (D_{re} \dot{\wedge} R_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} R_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs})$
25	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} R_e \dot{\vee} (D_{re} \dot{\wedge} R_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} R^<_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs})$
26	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} R_{psk})$
27	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R_{rs} \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R_{re} \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} R^<_{psk})$
28	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} R_e \dot{\vee} (D_{re} \dot{\wedge} R^<_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} R^<_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs})$
29	$(active \dot{\wedge} R^<_{re} \dot{\wedge} R^<_{rs}) \dot{\vee} (active \dot{\wedge} R^<_e) \dot{\vee} (D_{re} \dot{\wedge} R^<_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} R^<_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs})$
30	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^<_{rs} \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R^<_{re} \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} R^<_{psk})$
31	$(active \dot{\wedge} R^<_{re} \dot{\wedge} R^<_{rs} \dot{\wedge} R^<_{psk}) \dot{\vee} (active \dot{\wedge} R^<_e \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^<_{rs} \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R^<_{re} \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} R^<_{psk})$
32	$R_{re} \dot{\vee} R_s \dot{\vee} D_{re}$
33	$(active \dot{\wedge} R^<_{re}) \dot{\vee} (active \dot{\wedge} R^<_s) \dot{\vee} D_{re}$
34	$(R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^<_{psk})$
35	$(active \dot{\wedge} R^<_{re} \dot{\wedge} R^<_{psk}) \dot{\vee} (active \dot{\wedge} R^<_s \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^<_{psk})$
36	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} R_s \dot{\vee} (D_{re} \dot{\wedge} R^<_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} R^<_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs})$
37	$(active \dot{\wedge} R^<_{re} \dot{\wedge} R^<_{rs}) \dot{\vee} (active \dot{\wedge} R^<_s) \dot{\vee} (D_{re} \dot{\wedge} R^<_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} R^<_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs})$
38	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} R_s \dot{\vee} (D_{re} \dot{\wedge} R^<_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^<_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki})$
39	$(active \dot{\wedge} R^<_{re} \dot{\wedge} R^<_{rs}) \dot{\vee} (active \dot{\wedge} R^<_s) \dot{\vee} (D_{re} \dot{\wedge} R^<_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^<_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki})$
40	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^<_{rs} \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R^<_{re} \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} R^<_{psk})$
41	$(active \dot{\wedge} R^<_{re} \dot{\wedge} R^<_{rs} \dot{\wedge} R^<_{psk}) \dot{\vee} (active \dot{\wedge} R^<_s \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^<_{rs} \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R^<_{re} \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} R^<_{psk})$
42	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^<_{rs} \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^<_{re} \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^<_{psk})$
43	$(active \dot{\wedge} R^<_{re} \dot{\wedge} R^<_{rs} \dot{\wedge} R^<_{psk}) \dot{\vee} (active \dot{\wedge} R^<_s \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^<_{rs} \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^<_{re} \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^<_{psk})$
44	$R_{re} \dot{\vee} (R_e \dot{\wedge} R_s) \dot{\vee} D_{re}$
45	$(active \dot{\wedge} R^<_{re}) \dot{\vee} (active \dot{\wedge} R^<_e \dot{\wedge} R^<_s) \dot{\vee} D_{re}$
46	$(R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R_{psk})$
47	$(R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^<_{psk})$
48	$(active \dot{\wedge} R^<_{re} \dot{\wedge} R^<_{psk}) \dot{\vee} (active \dot{\wedge} R^<_e \dot{\wedge} R^<_s \dot{\wedge} R^<_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^<_{psk})$
49	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} (R_e \dot{\wedge} R_{re}) \dot{\vee} (R_e \dot{\wedge} R_s) \dot{\vee} (D_{re} \dot{\wedge} R_{rs}) \dot{\vee} (D_{re} \dot{\wedge} R_e) \dot{\vee} (D_{rs} \dot{\wedge} R_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs})$

Table B.4: Best threat model equivalence classes (continued)

Class	Best threat model
50	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R_e \dot{\wedge} R_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} R_{psk})$
51	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} (R_e \dot{\wedge} R_s) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{s} \dot{\wedge} R_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{re}) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{s} \dot{\wedge} R_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs})$
52	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{s} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{re} \dot{\wedge} R_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{s} \dot{\wedge} R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} R_{psk})$
53	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{s} \dot{\wedge} R_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{s} \dot{\wedge} R_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} R^{<}_{psk})$
54	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} (R_e \dot{\wedge} R_{re}) \dot{\vee} (R_e \dot{\wedge} R_s) \dot{\vee} (D_{re} \dot{\wedge} R_{rs}) \dot{\vee} (D_{re} \dot{\wedge} R_e) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs})$
55	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} (R_e \dot{\wedge} R_{re}) \dot{\vee} (R_e \dot{\wedge} R_s) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{e}) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs})$
56	$(active \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (active \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{re}) \dot{\vee} (active \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{s}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{e}) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs})$
57	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{e} \dot{\wedge} R_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{re} \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} R_{psk})$
58	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} R^{<}_{psk})$
59	$(active \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (active \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (active \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{s} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} R^{<}_{psk})$
60	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} (R_e \dot{\wedge} R_s) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs})$
61	$(active \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (active \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{s}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs})$
62	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} R^{<}_{psk})$
63	$(active \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (active \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{s} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} R^{<}_{psk})$
64	$R_{rs} \dot{\vee} R_e \dot{\vee} (D_{rs} \dot{\wedge} D_{pki})$
65	$(R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R_{psk})$
66	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} R_e \dot{\vee} (D_{re} \dot{\wedge} R_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki})$
67	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R_{psk})$
68	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{psk})$
69	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} (R_e \dot{\wedge} R_{re}) \dot{\vee} (R_e \dot{\wedge} R_s) \dot{\vee} (D_{re} \dot{\wedge} R_{rs}) \dot{\vee} (D_{re} \dot{\wedge} R_e) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki})$
70	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R_e \dot{\wedge} R_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R_{psk})$
71	$R_{rs} \dot{\vee} (R_e \dot{\wedge} R_s) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki})$
72	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} (R_e \dot{\wedge} R_s) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{s} \dot{\wedge} R_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{s} \dot{\wedge} R_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki})$
73	$(R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R_{psk})$
74	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{s} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re} \dot{\wedge} R_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{s} \dot{\wedge} R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R_{psk})$
75	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{s} \dot{\wedge} R_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{s} \dot{\wedge} R_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{psk})$
76	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} R_e \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki})$
77	$(active \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (active \dot{\wedge} R^{<}_{e}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki})$

Table B.4: Best threat model equivalence classes (continued)

Class	Best threat model
78	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{psk})$
79	$(active \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (active \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{psk})$
80	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} (R_e \dot{\wedge} R_{re}) \dot{\vee} (R_e \dot{\wedge} R_s) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{e}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki})$
81	$(active \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (active \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{re}) \dot{\vee} (active \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{s}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{e}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki})$
82	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{e} \dot{\wedge} R_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re} \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R_{psk})$
83	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_{re} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{psk})$
84	$(active \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (active \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (active \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{s} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{psk})$
85	$(R_{re} \dot{\wedge} R_{rs}) \dot{\vee} (R_e \dot{\wedge} R_s) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki})$
86	$(active \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (active \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{s}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki})$
87	$(R_{re} \dot{\wedge} R_{rs} \dot{\wedge} R_{psk}) \dot{\vee} (R_e \dot{\wedge} R_s \dot{\wedge} R_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{psk})$
88	$(active \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (active \dot{\wedge} R^{<}_{e} \dot{\wedge} R^{<}_{s} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} R^{<}_{rs} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{re} \dot{\wedge} R^{<}_{psk}) \dot{\vee} (D_{re} \dot{\wedge} D_{rs} \dot{\wedge} D_{pki} \dot{\wedge} R^{<}_{psk})$



Nodes are element of $\bar{E}$. The number on a node is the class index of this element in Table B.4. Only nodes which correspond to the best threat model of some secrecy or agreement claim on a non-PSK handshake are represented.
An arrow from a node A to a node B denotes that $B \rightharpoonup A$. In other word, threat models at the top of the graph are more permissive for the adversary; claims with such best threat models provide better guarantees.
The colour of a node denotes its security level as per the Noise specification [14]. From red to green, destination levels 0 to 5; and from light blue to dark blue, source destination level 0 to 2. A node has the colour of level n when there exists a claim with level n and the best threat model of this node. The node 0 at the bottom corresponds to best threat model $\top$ (no guarantees) and is shared by some source level 0 and destination level 0 claims, hence the two colours of this node.

Figure B.1: Structure of a subset of $\bar{E}$

Table B.5: Best threat model for agreement and secrecy claims

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
S	I1K	I	1	64
S	I1K	I	2	76
S	I1K	I	3	80
S	I1K	I	4	80
S	I1K	I	5	80
S	I1K	R	1	32
S	I1K	R	2	44
S	I1K	R	3	55
S	I1K	R	4	55
S	I1K	R	5	55
S	I1K1	I	1	0
S	I1K1	I	2	76
S	I1K1	I	3	80
S	I1K1	I	4	80
S	I1K1	I	5	80
S	I1K1	R	2	44
S	I1K1	R	3	55
S	I1K1	R	4	55
S	I1K1	R	5	55
S	I1N	I	1	0
S	I1N	I	2	19
S	I1N	I	3	44
S	I1N	I	4	44
S	I1N	I	5	44
S	I1N	R	2	19
S	I1N	R	3	28
S	I1N	R	4	28
S	I1N	R	5	28
S	I1X	I	1	0
S	I1X	I	2	28
S	I1X	I	3	55
S	I1X	I	4	55
S	I1X	I	5	55
S	I1X	R	2	44
S	I1X	R	3	55
S	I1X	R	4	55
S	I1X	R	5	55
S	I1X1	I	1	0
S	I1X1	I	2	19
S	I1X1	I	3	54
S	I1X1	I	4	55
S	I1X1	I	5	55
S	I1X1	R	2	19
S	I1X1	R	3	55
S	I1X1	R	4	55
S	I1X1	R	5	55
S	IK	I	1	71
S	IK	I	2	85
S	IK	I	3	85
S	IK	I	4	85
S	IK	R	1	36
S	IK	R	2	51
S	IK	R	3	60
S	IK	R	4	60
S	IK1	I	1	0
S	IK1	I	2	80
S	IK1	I	3	80
S	IK1	I	4	80
S	IK1	R	2	49
S	IK1	R	3	55
S	IK1	R	4	55
S	IKpsk1	I	1	73
S	IKpsk1	I	2	87
S	IKpsk1	I	3	87
S	IKpsk1	I	4	87
S	IKpsk1	R	1	40
S	IKpsk1	R	2	53
S	IKpsk1	R	3	62
S	IKpsk1	R	4	62
S	IKpsk2	I	1	71
S	IKpsk2	I	2	87
S	IKpsk2	I	3	87
S	IKpsk2	I	4	87
S	IKpsk2	R	1	36
S	IKpsk2	R	2	52
S	IKpsk2	R	3	62
S	IKpsk2	R	4	62
S	IN	I	1	0
S	IN	I	2	44
S	IN	I	3	44
S	IN	I	4	44
S	IN	R	2	24
S	IN	R	3	28
S	IN	R	4	28
S	INpsk1	I	1	17

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
S	INpsk1	I	2	47
S	INpsk1	I	3	47
S	INpsk1	I	4	47
S	INpsk1	R	1	17
S	INpsk1	R	2	27
S	INpsk1	R	3	30
S	INpsk1	R	4	30
S	INpsk2	I	2	47
S	INpsk2	I	3	47
S	INpsk2	I	4	47
S	INpsk2	R	2	26
S	INpsk2	R	3	30
S	INpsk2	R	4	30
S	IX	I	1	0
S	IX	I	2	55
S	IX	I	3	55
S	IX	I	4	55
S	IX	R	2	49
S	IX	R	3	55
S	IX	R	4	55
S	IX1	I	1	0
S	IX1	I	2	44
S	IX1	I	3	54
S	IX1	I	4	55
S	IX1	I	5	55
S	IX1	R	2	24
S	IX1	R	3	55
S	IX1	R	4	55
S	IX1	R	5	55
S	IXpsk2	I	2	58
S	IXpsk2	I	3	58
S	IXpsk2	I	4	58
S	IXpsk2	R	2	50
S	IXpsk2	R	3	58
S	IXpsk2	R	4	58
S	K1K	I	1	64
S	K1K	I	2	76
S	K1K	I	3	80
S	K1K	I	4	80
S	K1K	I	5	80
S	K1K	R	1	32
S	K1K	R	2	44
S	K1K	R	3	80
S	K1K	R	4	80

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
S	K1K	R	5	80
S	K1K1	I	1	0
S	K1K1	I	2	76
S	K1K1	I	3	80
S	K1K1	I	4	80
S	K1K1	I	5	80
S	K1K1	R	2	44
S	K1K1	R	3	80
S	K1K1	R	4	80
S	K1K1	R	5	80
S	K1N	I	1	0
S	K1N	I	2	19
S	K1N	I	3	44
S	K1N	I	4	44
S	K1N	I	5	44
S	K1N	R	2	19
S	K1N	R	3	76
S	K1N	R	4	76
S	K1N	R	5	76
S	K1X	I	1	0
S	K1X	I	2	28
S	K1X	I	3	55
S	K1X	I	4	55
S	K1X	I	5	55
S	K1X	R	2	44
S	K1X	R	3	80
S	K1X	R	4	80
S	K1X	R	5	80
S	K1X1	I	1	0
S	K1X1	I	2	19
S	K1X1	I	3	54
S	K1X1	I	4	55
S	K1X1	I	5	55
S	K1X1	R	2	19
S	K1X1	R	3	80
S	K1X1	R	4	80
S	K1X1	R	5	80
S	KK	I	1	71
S	KK	I	2	85
S	KK	I	3	85
S	KK	I	4	85
S	KK	R	1	38
S	KK	R	2	72
S	KK	R	3	85

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
S	KK	R	4	85
S	KK1	I	1	0
S	KK1	I	2	80
S	KK1	I	3	80
S	KK1	I	4	80
S	KK1	R	2	69
S	KK1	R	3	80
S	KK1	R	4	80
S	KKpsk0	I	1	73
S	KKpsk0	I	2	87
S	KKpsk0	I	3	87
S	KKpsk0	I	4	87
S	KKpsk0	R	1	42
S	KKpsk0	R	2	75
S	KKpsk0	R	3	87
S	KKpsk0	R	4	87
S	KKpsk2	I	1	71
S	KKpsk2	I	2	87
S	KKpsk2	I	3	87
S	KKpsk2	I	4	87
S	KKpsk2	R	1	38
S	KKpsk2	R	2	74
S	KKpsk2	R	3	87
S	KKpsk2	R	4	87
S	KN	I	1	0
S	KN	I	2	44
S	KN	I	3	44
S	KN	I	4	44
S	KN	R	2	66
S	KN	R	3	76
S	KN	R	4	76
S	KNpsk0	I	1	17
S	KNpsk0	I	2	47
S	KNpsk0	I	3	47
S	KNpsk0	I	4	47
S	KNpsk0	R	1	17
S	KNpsk0	R	2	68
S	KNpsk0	R	3	78
S	KNpsk0	R	4	78
S	KNpsk2	I	2	47
S	KNpsk2	I	3	47
S	KNpsk2	I	4	47
S	KNpsk2	R	2	67
S	KNpsk2	R	3	78

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
S	KNpsk2	R	4	78
S	KX	I	1	0
S	KX	I	2	55
S	KX	I	3	55
S	KX	I	4	55
S	KX	R	2	69
S	KX	R	3	80
S	KX	R	4	80
S	KX1	I	1	0
S	KX1	I	2	44
S	KX1	I	3	54
S	KX1	I	4	55
S	KX1	I	5	55
S	KX1	R	2	66
S	KX1	R	3	80
S	KX1	R	4	80
S	KX1	R	5	80
S	KXpsk2	I	2	58
S	KXpsk2	I	3	58
S	KXpsk2	I	4	58
S	KXpsk2	R	2	70
S	KXpsk2	R	3	83
S	KXpsk2	R	4	83
S	NK	I	1	64
S	NK	I	2	76
S	NK	I	3	76
S	NK	I	4	76
S	NK	R	1	32
S	NK	R	2	44
S	NK	R	3	44
S	NK	R	4	44
S	NK1	I	1	0
S	NK1	I	2	76
S	NK1	I	3	76
S	NK1	I	4	76
S	NK1	R	2	44
S	NK1	R	3	44
S	NK1	R	4	44
S	NKpsk0	I	1	65
S	NKpsk0	I	2	78
S	NKpsk0	I	3	78
S	NKpsk0	I	4	78
S	NKpsk0	R	1	34
S	NKpsk0	R	2	47

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
S	NKpsk0	R	3	47
S	NKpsk0	R	4	47
S	NKpsk2	I	1	64
S	NKpsk2	I	2	78
S	NKpsk2	I	3	78
S	NKpsk2	I	4	78
S	NKpsk2	R	1	32
S	NKpsk2	R	2	46
S	NKpsk2	R	3	47
S	NKpsk2	R	4	47
S	NN	I	1	0
S	NN	I	2	19
S	NN	I	3	19
S	NN	I	4	19
S	NN	R	2	19
S	NN	R	3	19
S	NN	R	4	19
S	NNpsk0	I	1	17
S	NNpsk0	I	2	22
S	NNpsk0	I	3	22
S	NNpsk0	I	4	22
S	NNpsk0	R	1	17
S	NNpsk0	R	2	22
S	NNpsk0	R	3	22
S	NNpsk0	R	4	22
S	NNpsk2	I	2	22
S	NNpsk2	I	3	22
S	NNpsk2	I	4	22
S	NNpsk2	R	2	21
S	NNpsk2	R	3	22
S	NNpsk2	R	4	22
S	NX	I	1	0
S	NX	I	2	28
S	NX	I	3	28
S	NX	I	4	28
S	NX	R	2	44
S	NX	R	3	44
S	NX	R	4	44
S	NX1	I	1	0
S	NX1	I	2	19
S	NX1	I	3	25
S	NX1	I	4	28
S	NX1	I	5	28
S	NX1	R	2	19

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
S	NX1	R	3	44
S	NX1	R	4	44
S	NX1	R	5	44
S	NXpsk2	I	2	30
S	NXpsk2	I	3	30
S	NXpsk2	I	4	30
S	NXpsk2	R	2	46
S	NXpsk2	R	3	47
S	NXpsk2	R	4	47
S	X1K	I	1	64
S	X1K	I	2	76
S	X1K	I	3	76
S	X1K	I	4	80
S	X1K	I	5	80
S	X1K	I	6	80
S	X1K	R	1	32
S	X1K	R	2	44
S	X1K	R	3	44
S	X1K	R	4	54
S	X1K	R	5	55
S	X1K	R	6	55
S	X1K1	I	1	0
S	X1K1	I	2	76
S	X1K1	I	3	76
S	X1K1	I	4	80
S	X1K1	I	5	80
S	X1K1	I	6	80
S	X1K1	R	2	44
S	X1K1	R	3	44
S	X1K1	R	4	54
S	X1K1	R	5	55
S	X1K1	R	6	55
S	X1N	I	1	0
S	X1N	I	2	19
S	X1N	I	3	19
S	X1N	I	4	44
S	X1N	I	5	44
S	X1N	I	6	44
S	X1N	R	2	19
S	X1N	R	3	19
S	X1N	R	4	25
S	X1N	R	5	28
S	X1N	R	6	28
S	X1X	I	1	0

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
S	X1X	I	2	28
S	X1X	I	3	28
S	X1X	I	4	55
S	X1X	I	5	55
S	X1X	I	6	55
S	X1X	R	2	44
S	X1X	R	3	44
S	X1X	R	4	54
S	X1X	R	5	55
S	X1X	R	6	55
S	X1X1	I	1	0
S	X1X1	I	2	19
S	X1X1	I	3	25
S	X1X1	I	4	55
S	X1X1	I	5	55
S	X1X1	I	6	55
S	X1X1	R	2	19
S	X1X1	R	3	44
S	X1X1	R	4	54
S	X1X1	R	5	55
S	X1X1	R	6	55
S	XK	I	1	64
S	XK	I	2	76
S	XK	I	3	80
S	XK	I	4	80
S	XK	I	5	80
S	XK	R	1	32
S	XK	R	2	44
S	XK	R	3	55
S	XK	R	4	55
S	XK	R	5	55
S	XK1	I	1	0
S	XK1	I	2	76
S	XK1	I	3	80
S	XK1	I	4	80
S	XK1	I	5	80
S	XK1	R	2	44
S	XK1	R	3	55
S	XK1	R	4	55
S	XK1	R	5	55
S	XKpsk3	I	1	64
S	XKpsk3	I	2	76
S	XKpsk3	I	3	82
S	XKpsk3	I	4	83

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
S	XKpsk3	I	5	83
S	XKpsk3	R	1	32
S	XKpsk3	R	2	44
S	XKpsk3	R	3	58
S	XKpsk3	R	4	58
S	XKpsk3	R	5	58
S	XN	I	1	0
S	XN	I	2	19
S	XN	I	3	44
S	XN	I	4	44
S	XN	I	5	44
S	XN	R	2	19
S	XN	R	3	28
S	XN	R	4	28
S	XN	R	5	28
S	XNpsk3	I	2	19
S	XNpsk3	I	3	46
S	XNpsk3	I	4	47
S	XNpsk3	I	5	47
S	XNpsk3	R	2	19
S	XNpsk3	R	3	30
S	XNpsk3	R	4	30
S	XNpsk3	R	5	30
S	XX	I	1	0
S	XX	I	2	28
S	XX	I	3	55
S	XX	I	4	55
S	XX	I	5	55
S	XX	R	2	44
S	XX	R	3	55
S	XX	R	4	55
S	XX	R	5	55
S	XX1	I	1	0
S	XX1	I	2	19
S	XX1	I	3	54
S	XX1	I	4	55
S	XX1	I	5	55
S	XX1	R	2	19
S	XX1	R	3	55
S	XX1	R	4	55
S	XX1	R	5	55
S	XXpsk3	I	2	28
S	XXpsk3	I	3	57
S	XXpsk3	I	4	58

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
S	XXpsk3	I	5	58
S	XXpsk3	R	2	44
S	XXpsk3	R	3	58
S	XXpsk3	R	4	58
S	XXpsk3	R	5	58
NA	I1K	I	2	77
NA	I1K	I	4	81
NA	I1K	R	1	33
NA	I1K	R	3	56
NA	I1K	R	5	56
NA	I1K1	I	2	77
NA	I1K1	I	4	81
NA	I1K1	R	1	0
NA	I1K1	R	3	56
NA	I1K1	R	5	56
NA	I1N	I	2	20
NA	I1N	I	4	45
NA	I1N	R	1	16
NA	I1N	R	3	29
NA	I1N	R	5	29
NA	I1X	I	2	29
NA	I1X	I	4	56
NA	I1X	R	1	16
NA	I1X	R	3	56
NA	I1X	R	5	56
NA	I1X1	I	2	20
NA	I1X1	I	4	56
NA	I1X1	R	1	16
NA	I1X1	R	3	56
NA	I1X1	R	5	56
NA	IK	I	2	86
NA	IK	I	4	86
NA	IK	R	1	37
NA	IK	R	3	61
NA	IK1	I	2	81
NA	IK1	I	4	81
NA	IK1	R	1	0
NA	IK1	R	3	56
NA	IKpsk1	I	2	88
NA	IKpsk1	I	4	88
NA	IKpsk1	R	1	41
NA	IKpsk1	R	3	63
NA	IKpsk2	I	2	88
NA	IKpsk2	I	4	88

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
NA	IKpsk2	R	1	37
NA	IKpsk2	R	3	63
NA	IN	I	2	45
NA	IN	I	4	45
NA	IN	R	1	16
NA	IN	R	3	29
NA	INpsk1	I	2	48
NA	INpsk1	I	4	48
NA	INpsk1	R	1	18
NA	INpsk1	R	3	31
NA	INpsk2	I	2	48
NA	INpsk2	I	4	48
NA	INpsk2	R	1	16
NA	INpsk2	R	3	31
NA	IX	I	2	56
NA	IX	I	4	56
NA	IX	R	1	16
NA	IX	R	3	56
NA	IX1	I	2	45
NA	IX1	I	4	56
NA	IX1	R	1	16
NA	IX1	R	3	56
NA	IX1	R	5	56
NA	IXpsk2	I	2	59
NA	IXpsk2	I	4	59
NA	IXpsk2	R	1	16
NA	IXpsk2	R	3	59
NA	K1K	I	2	77
NA	K1K	I	4	81
NA	K1K	R	1	33
NA	K1K	R	3	81
NA	K1K	R	5	81
NA	K1K1	I	2	77
NA	K1K1	I	4	81
NA	K1K1	R	1	0
NA	K1K1	R	3	81
NA	K1K1	R	5	81
NA	K1N	I	2	20
NA	K1N	I	4	45
NA	K1N	R	1	0
NA	K1N	R	3	77
NA	K1N	R	5	77
NA	K1X	I	2	29
NA	K1X	I	4	56

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
NA	K1X	R	1	0
NA	K1X	R	3	81
NA	K1X	R	5	81
NA	K1X1	I	2	20
NA	K1X1	I	4	56
NA	K1X1	R	1	0
NA	K1X1	R	3	81
NA	K1X1	R	5	81
NA	KK	I	2	86
NA	KK	I	4	86
NA	KK	R	1	39
NA	KK	R	3	86
NA	KK1	I	2	81
NA	KK1	I	4	81
NA	KK1	R	1	0
NA	KK1	R	3	81
NA	KKpsk0	I	2	88
NA	KKpsk0	I	4	88
NA	KKpsk0	R	1	43
NA	KKpsk0	R	3	88
NA	KKpsk2	I	2	88
NA	KKpsk2	I	4	88
NA	KKpsk2	R	1	39
NA	KKpsk2	R	3	88
NA	KN	I	2	45
NA	KN	I	4	45
NA	KN	R	1	0
NA	KN	R	3	77
NA	KNpsk0	I	2	48
NA	KNpsk0	I	4	48
NA	KNpsk0	R	1	18
NA	KNpsk0	R	3	79
NA	KNpsk2	I	2	48
NA	KNpsk2	I	4	48
NA	KNpsk2	R	1	16
NA	KNpsk2	R	3	79
NA	KX	I	2	56
NA	KX	I	4	56
NA	KX	R	1	0
NA	KX	R	3	81
NA	KX1	I	2	45
NA	KX1	I	4	56
NA	KX1	R	1	0
NA	KX1	R	3	81

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
NA	KX1	R	5	81
NA	KXpsk2	I	2	59
NA	KXpsk2	I	4	59
NA	KXpsk2	R	1	16
NA	KXpsk2	R	3	84
NA	NK	I	2	77
NA	NK	I	4	77
NA	NK	R	1	33
NA	NK	R	3	45
NA	NK1	I	2	77
NA	NK1	I	4	77
NA	NK1	R	1	0
NA	NK1	R	3	45
NA	NKpsk0	I	2	79
NA	NKpsk0	I	4	79
NA	NKpsk0	R	1	35
NA	NKpsk0	R	3	48
NA	NKpsk2	I	2	79
NA	NKpsk2	I	4	79
NA	NKpsk2	R	1	33
NA	NKpsk2	R	3	48
NA	NN	I	2	20
NA	NN	I	4	20
NA	NN	R	1	0
NA	NN	R	3	20
NA	NNpsk0	I	2	23
NA	NNpsk0	I	4	23
NA	NNpsk0	R	1	18
NA	NNpsk0	R	3	23
NA	NNpsk2	I	2	23
NA	NNpsk2	I	4	23
NA	NNpsk2	R	1	16
NA	NNpsk2	R	3	23
NA	NX	I	2	29
NA	NX	I	4	29
NA	NX	R	1	0
NA	NX	R	3	45
NA	NX1	I	2	20
NA	NX1	I	4	29
NA	NX1	R	1	0
NA	NX1	R	3	45
NA	NX1	R	5	45
NA	NXpsk2	I	2	31
NA	NXpsk2	I	4	31

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
NA	NXpsk2	R	1	16
NA	NXpsk2	R	3	48
NA	X1K	I	2	77
NA	X1K	I	4	81
NA	X1K	I	6	81
NA	X1K	R	1	33
NA	X1K	R	3	45
NA	X1K	R	5	56
NA	X1K1	I	2	77
NA	X1K1	I	4	81
NA	X1K1	I	6	81
NA	X1K1	R	1	0
NA	X1K1	R	3	45
NA	X1K1	R	5	56
NA	X1N	I	2	20
NA	X1N	I	4	45
NA	X1N	I	6	45
NA	X1N	R	1	0
NA	X1N	R	3	20
NA	X1N	R	5	29
NA	X1X	I	2	29
NA	X1X	I	4	56
NA	X1X	I	6	56
NA	X1X	R	1	0
NA	X1X	R	3	45
NA	X1X	R	5	56
NA	X1X1	I	2	20
NA	X1X1	I	4	56
NA	X1X1	I	6	56
NA	X1X1	R	1	0
NA	X1X1	R	3	45
NA	X1X1	R	5	56
NA	XK	I	2	77
NA	XK	I	4	81
NA	XK	R	1	33
NA	XK	R	3	56
NA	XK	R	5	56
NA	XK1	I	2	77
NA	XK1	I	4	81
NA	XK1	R	1	0
NA	XK1	R	3	56
NA	XK1	R	5	56
NA	XKpsk3	I	2	77
NA	XKpsk3	I	4	84

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
NA	XKpsk3	R	1	33
NA	XKpsk3	R	3	59
NA	XKpsk3	R	5	59
NA	XN	I	2	20
NA	XN	I	4	45
NA	XN	R	1	0
NA	XN	R	3	29
NA	XN	R	5	29
NA	XNpsk3	I	2	20
NA	XNpsk3	I	4	48
NA	XNpsk3	R	1	16
NA	XNpsk3	R	3	31
NA	XNpsk3	R	5	31
NA	XX	I	2	29
NA	XX	I	4	56
NA	XX	R	1	0
NA	XX	R	3	56
NA	XX	R	5	56
NA	XX1	I	2	20
NA	XX1	I	4	56
NA	XX1	R	1	0
NA	XX1	R	3	56
NA	XX1	R	5	56
NA	XXpsk3	I	2	29
NA	XXpsk3	I	4	59
NA	XXpsk3	R	1	16
NA	XXpsk3	R	3	59
NA	XXpsk3	R	5	59
IA	I1K	I	2	77
IA	I1K	I	4	81
IA	I1K	R	1	16
IA	I1K	R	3	56
IA	I1K	R	5	56
IA	I1K1	I	2	77
IA	I1K1	I	4	81
IA	I1K1	R	1	0
IA	I1K1	R	3	56
IA	I1K1	R	5	56
IA	I1N	I	2	20
IA	I1N	I	4	45
IA	I1N	R	1	16
IA	I1N	R	3	29
IA	I1N	R	5	29
IA	I1X	I	2	29

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
IA	I1X	I	4	56
IA	I1X	R	1	16
IA	I1X	R	3	56
IA	I1X	R	5	56
IA	I1X1	I	2	20
IA	I1X1	I	4	56
IA	I1X1	R	1	16
IA	I1X1	R	3	56
IA	I1X1	R	5	56
IA	IK	I	2	86
IA	IK	I	4	86
IA	IK	R	1	16
IA	IK	R	3	61
IA	IK1	I	2	81
IA	IK1	I	4	81
IA	IK1	R	1	0
IA	IK1	R	3	56
IA	IKpsk1	I	2	88
IA	IKpsk1	I	4	88
IA	IKpsk1	R	1	16
IA	IKpsk1	R	3	63
IA	IKpsk2	I	2	88
IA	IKpsk2	I	4	88
IA	IKpsk2	R	1	16
IA	IKpsk2	R	3	63
IA	IN	I	2	45
IA	IN	I	4	45
IA	IN	R	1	16
IA	IN	R	3	29
IA	INpsk1	I	2	48
IA	INpsk1	I	4	48
IA	INpsk1	R	1	16
IA	INpsk1	R	3	31
IA	INpsk2	I	2	48
IA	INpsk2	I	4	48
IA	INpsk2	R	1	16
IA	INpsk2	R	3	31
IA	IX	I	2	56
IA	IX	I	4	56
IA	IX	R	1	16
IA	IX	R	3	56
IA	IX1	I	2	45
IA	IX1	I	4	56
IA	IX1	R	1	16

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
IA	IX1	R	3	56
IA	IX1	R	5	56
IA	IXpsk2	I	2	59
IA	IXpsk2	I	4	59
IA	IXpsk2	R	1	16
IA	IXpsk2	R	3	59
IA	K1K	I	2	77
IA	K1K	I	4	81
IA	K1K	R	1	16
IA	K1K	R	3	81
IA	K1K	R	5	81
IA	K1K1	I	2	77
IA	K1K1	I	4	81
IA	K1K1	R	1	0
IA	K1K1	R	3	81
IA	K1K1	R	5	81
IA	K1N	I	2	20
IA	K1N	I	4	45
IA	K1N	R	1	0
IA	K1N	R	3	77
IA	K1N	R	5	77
IA	K1X	I	2	29
IA	K1X	I	4	56
IA	K1X	R	1	0
IA	K1X	R	3	81
IA	K1X	R	5	81
IA	K1X1	I	2	20
IA	K1X1	I	4	56
IA	K1X1	R	1	0
IA	K1X1	R	3	81
IA	K1X1	R	5	81
IA	KK	I	2	86
IA	KK	I	4	86
IA	KK	R	1	16
IA	KK	R	3	86
IA	KK1	I	2	81
IA	KK1	I	4	81
IA	KK1	R	1	0
IA	KK1	R	3	81
IA	KKpsk0	I	2	88
IA	KKpsk0	I	4	88
IA	KKpsk0	R	1	16
IA	KKpsk0	R	3	88
IA	KKpsk2	I	2	88

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
IA	KKpsk2	I	4	88
IA	KKpsk2	R	1	16
IA	KKpsk2	R	3	88
IA	KN	I	2	45
IA	KN	I	4	45
IA	KN	R	1	0
IA	KN	R	3	77
IA	KNpsk0	I	2	48
IA	KNpsk0	I	4	48
IA	KNpsk0	R	1	16
IA	KNpsk0	R	3	79
IA	KNpsk2	I	2	48
IA	KNpsk2	I	4	48
IA	KNpsk2	R	1	16
IA	KNpsk2	R	3	79
IA	KX	I	2	56
IA	KX	I	4	56
IA	KX	R	1	0
IA	KX	R	3	81
IA	KX1	I	2	45
IA	KX1	I	4	56
IA	KX1	R	1	0
IA	KX1	R	3	81
IA	KX1	R	5	81
IA	KXpsk2	I	2	59
IA	KXpsk2	I	4	59
IA	KXpsk2	R	1	16
IA	KXpsk2	R	3	84
IA	NK	I	2	77
IA	NK	I	4	77
IA	NK	R	1	16
IA	NK	R	3	45
IA	NK1	I	2	77
IA	NK1	I	4	77
IA	NK1	R	1	0
IA	NK1	R	3	45
IA	NKpsk0	I	2	79
IA	NKpsk0	I	4	79
IA	NKpsk0	R	1	16
IA	NKpsk0	R	3	48
IA	NKpsk2	I	2	79
IA	NKpsk2	I	4	79
IA	NKpsk2	R	1	16
IA	NKpsk2	R	3	48

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
IA	NN	I	2	20
IA	NN	I	4	20
IA	NN	R	1	0
IA	NN	R	3	20
IA	NNpsk0	I	2	23
IA	NNpsk0	I	4	23
IA	NNpsk0	R	1	16
IA	NNpsk0	R	3	23
IA	NNpsk2	I	2	23
IA	NNpsk2	I	4	23
IA	NNpsk2	R	1	16
IA	NNpsk2	R	3	23
IA	NX	I	2	29
IA	NX	I	4	29
IA	NX	R	1	0
IA	NX	R	3	45
IA	NX1	I	2	20
IA	NX1	I	4	29
IA	NX1	R	1	0
IA	NX1	R	3	45
IA	NX1	R	5	45
IA	NXpsk2	I	2	31
IA	NXpsk2	I	4	31
IA	NXpsk2	R	1	16
IA	NXpsk2	R	3	48
IA	X1K	I	2	77
IA	X1K	I	4	81
IA	X1K	I	6	81
IA	X1K	R	1	16
IA	X1K	R	3	45
IA	X1K	R	5	56
IA	X1K1	I	2	77
IA	X1K1	I	4	81
IA	X1K1	I	6	81
IA	X1K1	R	1	0
IA	X1K1	R	3	45
IA	X1K1	R	5	56
IA	X1N	I	2	20
IA	X1N	I	4	45
IA	X1N	I	6	45
IA	X1N	R	1	0
IA	X1N	R	3	20
IA	X1N	R	5	29
IA	X1X	I	2	29

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
IA	X1X	I	4	56
IA	X1X	I	6	56
IA	X1X	R	1	0
IA	X1X	R	3	45
IA	X1X	R	5	56
IA	X1X1	I	2	20
IA	X1X1	I	4	56
IA	X1X1	I	6	56
IA	X1X1	R	1	0
IA	X1X1	R	3	45
IA	X1X1	R	5	56
IA	XK	I	2	77
IA	XK	I	4	81
IA	XK	R	1	16
IA	XK	R	3	56
IA	XK	R	5	56
IA	XK1	I	2	77
IA	XK1	I	4	81
IA	XK1	R	1	0
IA	XK1	R	3	56
IA	XK1	R	5	56
IA	XKpsk3	I	2	77
IA	XKpsk3	I	4	84
IA	XKpsk3	R	1	16
IA	XKpsk3	R	3	59
IA	XKpsk3	R	5	59
IA	XN	I	2	20
IA	XN	I	4	45
IA	XN	R	1	0
IA	XN	R	3	29
IA	XN	R	5	29
IA	XNpsk3	I	2	20
IA	XNpsk3	I	4	48
IA	XNpsk3	R	1	16
IA	XNpsk3	R	3	31
IA	XNpsk3	R	5	31
IA	XX	I	2	29
IA	XX	I	4	56
IA	XX	R	1	0
IA	XX	R	3	56
IA	XX	R	5	56
IA	XX1	I	2	20
IA	XX1	I	4	56
IA	XX1	R	1	0

Goal	Pattern	Role	i	c
IA:	Injective agreement			
NA:	Non-injective agreement			
S:	Secrecy			
R:	Responder	I:	Initiator	
IA	XX1	R	3	56
IA	XX1	R	5	56
IA	XXpsk3	I	2	29
IA	XXpsk3	I	4	59
IA	XXpsk3	R	1	16
IA	XXpsk3	R	3	59
IA	XXpsk3	R	5	59

Bibliography

[1] The On-Line Encyclopedia of Integer Sequences. Sequence A007153.

[2] Adrian Antipa, Daniel Brown, Alfred Menezes, René Struik, and Scott Vanstone. Validation of elliptic curve public keys. In Yvo G. Desmedt, editor, *Public Key Cryptography — PKC 2003*, pages 211–223, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-36288-3.

[3] 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. ISSN 1094-9224. doi: 10.1145/2658996. URL `http://doi.acm.org/10.1145/2658996`.

[4] 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.

[5] 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, 2016. ISSN 2474-1566. doi: 10.1561/3300000004. URL `http://dx.doi.org/10.1561/3300000004`.

[6] Jason A. Donenfeld. Wireguard: Next generation kernel network tunnel. *Proceedings 2017 Network and Distributed System Security Symposium*, 2017. doi: 10.14722/ndss.2017.23160. URL `http://dx.doi.org/10.14722/ndss.2017.23160`.

[7] Jason A. Donenfeld and Kevin Milner. Formal verification of the wireguard protocol. July 2017. URL `https://www.wireguard.com/papers/wireguard-formal-verification.pdf`.

[8] Guillaume Girol. Formalizing and Verifying the Security Protocols from the Noise Framework. `https://www.ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/information-security-group-dam/research/software/noise-ggirol.zip`. Source code and Tamarin proof files.

[9] WhatsApp Inc. Whatsapp encryption overwiew—technical white paper, December 2017. URL `https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf`.

[10] Nadim Kobeissi and Karthikeyan Bhargavan. Noise explorer: Fully automated modeling and verification for arbitrary noise protocols. In *Real World Cryptography Symposium*, 2018.

[11] Nadim Kobeissi and Karthikeyan Bhargavan. Noise Explorer: Fully Automated Modeling and Verification for Arbitrary Noise Protocols. In *RWC 2019 - Real World Cryptography Symposium*, San Jose, United States, January 2019. URL `https://hal.inria.fr/hal-01948964`.

[12] G. Lowe. A hierarchy of authentication specifications. In *Proceedings 10th Computer Security Foundations Workshop*, pages 31–43, June 1997. doi: 10.1109/CSFW.1997.596782.

[13] The Lightning Network. Bolt 8: Encrypted and authenticated transport, December 2017. URL `https://github.com/lightningnetwork/lightning-rfc/blob/130bc5da2c05f212fba09ae309e53fec8cde2c6d/08-transport.md`.

[14] Trevor Perrin. The noise protocol framework, July 2018. URL `https://github.com/noiseprotocol/noise_spec/tree/ecdf084ece2bf92b16b1201b6ae5c99d23fb4151`. (revision 34).

[15] The Debian Project. Openssl – predictable random number generator. DSA-1571-1, May 2008. URL `https://www.debian.org/security/2008/dsa-1571`.

[16] Benedikt Schmidt. *Formal analysis of key exchange protocols and physical protocols*. PhD thesis, ETH, Zürich, 2012.

[17] A. Suter-Dörig. Formalizing and verifying the security protocols from the noise framework. Master's thesis, ETH, November 2018. URL `https://www.ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/information-security-group-dam/research/software/noise_suter-doerig.pdf`.

[18] The Tamarin Team. *Tamarin-Prover Manual*. URL `https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf`.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

FORMALIZING AND VERIFYING THE SECURITY PROTOCOLS FROM THE NOISE FRAMEWORK.

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
GIROL	GUILLAUME YVES

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 12/03/2019	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.