# Formal Verification of the WireGuard Protocol

www.wireguard.com

Jason A. Donenfeld
jason@zx2c4.com

Kevin Milner
Oxford University
kevin.milner@cs.ox.ac.uk

Draft Revision

**Abstract**

WireGuard, the secure network tunnel, uses an interesting Diffie-Hellman authenticated key exchange protocol based on NoiseIK, custom tailored to suit its unique operational requirements. This paper enumerates the security properties of this key exchange and then explores the formal verification of such properties. The end result is a formally verified secure network tunnel protocol.

## 1 Background

WireGuard is a secure network tunnel for transporting layer 3 packets, for some definition of secure. This paper endeavors to determine exactly what that is, and formally verify it as such. The WireGuard protocol is extensively detailed in [2], which itself is based on the NoiseIK [3] handshake. The WireGuard protocol consists of several mechanisms: an authenticated key exchange, a cookie MAC system for mitigating denial of service attacks, an elegant timer state machine for concealing state to system administrators, and several desirable design decisions from an implementation security perspective. Notably, WireGuard aims to be silent in the face of unauthenticated packets ("stealthiness"), avoid dynamic memory allocation, only adjust state based on authenticated data in order to maintain fixed size memory allocations, avoid parsers, not require any long term storage, allow initiator and responder to swap roles, and have the state machine handle all transitions automatically. These design requirements lead to an interesting authenticated key exchange, around which the other aspects of the protocol revolve.

Permanent ID of this document: d376f649d7f4b68f616e05e5f64d660e9b23d7af. Static link: wireguard.com/papers/wireguard-formal-verification.pdf. Date: July 18, 2017. This is draft revision ed8b131. Copyright © 2017 Jason A. Donenfeld & Kevin Milner. All Rights Reserved.

## 2 Key Exchange Review

While the WireGuard key exchange is extensively discussed in [2], we briefly review the overall structure and import several variables and conventions. Leaving aside the cookie MAC system, WireGuard consists of a 1-RTT handshake—from initiator to responder, and responder back to initiator—after which the initiator can then begin an exchange of transport data, depicted in figure 1. This initial exchange of transport data from the initiator provides key confirmation, and after, either initiator or responder may send transport data. Transport data is encrypted using an AEAD, with a pair of symmetric keys, one for sending and one for receiving, derived from the preceding two handshake messages, which are referred to as the "session keys", existing within a "session".

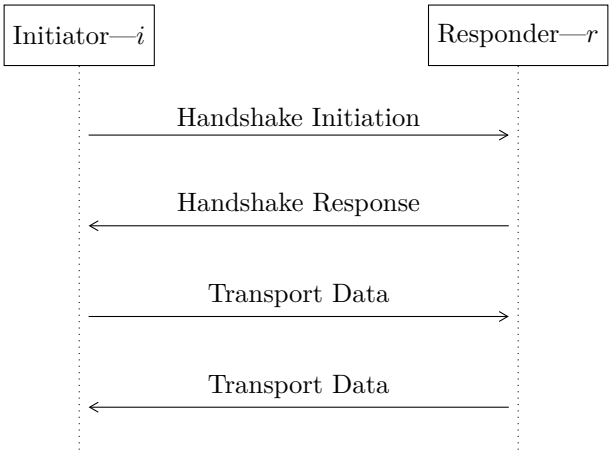


Figure 1: 1-RTT handshake between initiator and responder

The initiator, $i$, and the responder, $r$, both keep several values secret. They each have a static long-term key, $S_i$ and $S_r$, as well as a random session ephemeral key, $E_i$ and $E_r$. They may *optionally* also share a pre-shared symmetric key, $Q$; when $Q$ is in use, WireGuard is using what we call "pre-shared key mode", and otherwise WireGuard is using "normal mode", and $Q$ is considered to be $0^{32}$, the all zero 256-bit string. The initiator and responder exchange messages and keep track of a shared secret, $C$.

## 3 Properties

We enumerate properties first in the case $S_i$, $S_r$, $E_i$, and $E_r$ (and potentially $Q$) are kept secret. Then we explore which properties still remain when one or more of these secret values are compromised by an attacker. An attacker is posited to be in an active man-in-the-middle position. As pre-shared key mode is meant to merely *augment* the security properties, in the face of a quantum attacker

or possibly other circumstances, the security properties described herein first discuss what is provided by normal mode, and then subsequently approach the additional properties afforded by pre-shared key mode.

## 3.1 Key Agreement & Correctness

The initiator and responder are able to complete the handshake and arrive at the same session keys. Without an attacker, this constitutes a property of *correctness*—the protocol at least works. With an active attacker, the fact that key agreement necessarily occurs means that the session has authenticity.

### 3.1.1 No Compromised Keys

When no secret values are compromised, WireGuard achieves key agreement and correctness.

### 3.1.2 Compromised Ephemeral Keys

When either $E_i$, $E_r$, or $E_i$ and $E_r$ are compromised, WireGuard achieves key agreement and correctness.

### 3.1.3 Compromised Static Keys

When $S_i$ is compromised, the initiator achieves key agreement with the responder, because WireGuard is resistant to key-compromise impersonation (KCI), but the responder does not achieve key agreement with the initiator. When $S_r$ is compromised, the responder achieves key agreement with the initiator, because WireGuard is resistant to KCI, but the initiator does not achieve key agreement with the responder. (When $S_i$ *and* $S_r$ are both compromised, there is *no* key agreement.)

### 3.1.4 Compromised Static Key & Ephemeral Key

The KCI resistance of the prior section also holds when either $S_i$ and $E_r$ or $S_r$ and $E_i$ are compromised.

When all four keys, $S_i$, $S_r$, $E_i$, and $E_r$, are compromised, WireGuard remains resistant to an unknown key-share (UKS) attack. That is, if the initiator and responder derive $C$, then they must necessarily agree on all four keys, even if they are compromised.

### 3.1.5 Pre-shared Key Mode

When using pre-shared key mode, if $Q$ is not compromised, WireGuard achieves key agreement and correctness, even when $S_i$, $S_r$, $E_i$, and $E_r$ are all compromised. When $Q$ is compromised, the properties of the preceding four sections hold.

## 3.2 Key Secrecy

When there is key agreement, there is *also* key secrecy, when the resultant session keys are only known the initiator and responder, and not to the attacker.

### 3.2.1 No Compromised Keys

When no secret values are compromised, WireGuard achieves key secrecy.

### 3.2.2 Compromised Ephemeral Keys

When either $E_i$, $E_r$, or $E_i$ and $E_r$ are compromised, WireGuard achieves key secrecy.

### 3.2.3 Compromised Static Keys

When $S_i$, $S_r$, or $S_i$ and $S_r$ are compromised, WireGuard achieves key secrecy.

### 3.2.4 Compromised Static Key & Ephemeral Key

When either $S_i$ and $E_r$ or $S_r$ and $E_i$ are compromised, WireGuard achieves key secrecy.

### 3.2.5 Forward Secrecy

Forward secrecy changes the attacker's position to that of a passive man-in-the-middle, recording all traffic, who at some later time compromises either $S_i$, $S_r$, or $S_i$ and $S_r$. As stated above, if the attacker does not also compromise either $E_i$ or $E_r$, WireGuard achieves forward secrecy.

### 3.2.6 Pre-shared Key Mode

When using pre-shared key mode, if $Q$ is not compromised, WireGuard achieves key secrecy, even when $S_i$, $S_r$, $E_i$, and $E_r$ are all compromised. When $Q$ is compromised, the properties of the preceding five sections hold.

## 3.3 Session Uniqueness

Different sessions will always have different, unique, session keys, due to the use of the ephemeral values, $E_i$ and $E_r$.

### 3.3.1 Total Freshness

When both $E_i$ and $E_r$ are fresh, WireGuard achieves session uniqueness.

### 3.3.2 Partial Freshness

When either $E_i$ or $E_r$ is fresh, WireGuard achieves session uniqueness.

### 3.4 Identity Hiding

$S_r$ is never transmitted in any form; it is secret from an attacker. $S_i$ is transmitted, but encrypted; it is secret from an attacker.

#### 3.4.1 No Compromised Keys

When no secret values are compromised, WireGuard achieves identity hiding.

#### 3.4.2 Compromised Ephemeral Keys

When $E_r$ is compromised, WireGuard achieves identity hiding. When $E_i$ is compromised, WireGuard does not achieve identity hiding.

#### 3.4.3 Compromised Static Keys

When $S_i$ is compromised, $S_i$ is no longer secret, by definition. When $S_r$ is compromised, WireGuard does not achieve identity hiding; this is a reasonable trade-off permitted by the protocol.

## 4 Formal Model & Proofs

### 4.1 Tamarin

Tamarin [5] is a formal verification tool supporting unbounded verification of security protocols modulo an equational theory. Protocols are specified as a collection of labeled multiset-rewriting rules, which execute subject to an adversary and equational theory. We give a brief overview of how Tamarin works here; for a detailed discussion see [4]. For an introduction to using Tamarin, we suggest the manual at [6].

#### Facts and Rules

A model in Tamarin defines a transition system, with rules specifying how the world's state can change over time. State is represented by a multiset of *facts*, each of which has an associated name and arity. For example, we write $\mathsf{Fa}(x, y)$ for a fact named "Fa" with two associated terms $x$ and $y$. Facts are the atoms from which we build our protocol model, used to represent agent state, limited resources, and other execution constraints.

Rules specify how world state can change over time, representing both adversary capabilities and the behavior of participating agents (similar to oracles in Bellare-Rogaway style models). Executing a rule at some time point may require some facts to exist prior to that time point, and may create some other facts at that time point; these facts are called the *premises* and *conclusions* of the rule respectively.

Facts can either be marked as *persistent* or *linear*; persistent facts may be used repeatedly to satisfy the premises of rules, while linear facts are removed

from the world state at the timepoint they are used to execute a rule. This allows us to efficiently model read-only memory or causal constraints using persistent facts, and also allows us to represent linear constraints like non-monotonic agent state, limited resources, and control flow.

Rules are labeled with *actions*, such that the execution of a rule at some time point associates some action that time point. Actions do not influence rule execution directly, but instead incrementally construct a *trace* which represents a sort of record of rule execution. When we prove that a protocol model has some property, we reference these actions and are proving a property of the set of all traces which can be generated by the protocol model.

Finally, there are a few special facts in Tamarin, to model the generation of fresh terms (e.g. from a PRNG) and to interact with the network. The $\mathsf{Fr}(x)$ fact can be used in the premises of a rule to model generating a fresh term $x$. $\mathsf{In}(\dots)$ and $\mathsf{Out}(\dots)$ can be used in the premises and conclusions respectively to model receiving from or sending to the adversary-controlled network.

#### Adversary and Equational Theories

Tamarin models cryptography symbolically, and assumes a Dolev-Yao adversary [1]. That is, the adversary intercepts every message and can block or modify them as much as desired, restricted only by a message deduction theory.

Tamarin models interaction with the network through special facts as mentioned above. Terms sent to the network using an $\mathsf{Out}(\dots)$ fact in the conclusion of a rule can be used by the adversary to construct new messages, which can then be received by using the $\mathsf{In}(\dots)$ fact in the premises of a rule. The adversary can also construct or deconstruct terms according to public functions. For example, if encrypt(_, _) is a public function, then an adversary with knowledge of two terms $x$ and $y$ the can construct $\text{encrypt}(x, y)$.

Equality constraints in Tamarin are equality modulo an equational theory, rather than being strictly based on syntactic matching. For example, our equational theory may include

$$\text{decrypt}(\text{encrypt}(m, k), k) = m,$$

even though the terms $m$ and $\text{decrypt}(\text{encrypt}(m, k), k)$ are not syntactically identical. Tamarin includes built in definitions for a number of different cryptographic primitives, and additional formulae and equations can be defined easily. Notably, Tamarin includes support for a Diffie-Hellman equational theory, allowing symbolic reasoning about key exchange protocols and adversary capability in that setting.

### 4.2 WireGuard Model

The model of WireGuard makes use of the Diffie-Hellman equational theory defined in [5] with five additional formula: h/1, for hashing, aead/3, decrypt/2, verify/3 for authenticated encryption with additional data (AEAD) operations,

and true/0, for an affirmative result of AEAD decryption verification. Here, $/k$ denotes $k$-arity. These formulae have the equations

$$\text{decrypt}(\text{aead}(k, p, a), k) \to p$$
$$\text{verify}(\text{aead}(k, p, a), a, k) \to \text{true},$$

representing the ability to recover the encrypted plaintext $p$ of $\text{aead}(k, p, a)$ using the key $k$, and also verify its authenticity using both the key $k$ and the additional authenticated data (AAD) $a$.

Note that these equations do not provide the adversary a way to scramble the plaintext in the AEAD such that it could be decrypted incorrectly when the AAD is not known, as could happen in a real-world flawed implementation. For the purposes of modeling WireGuard, this is sufficient, since the AAD is always known. Interestingly, decrypt is never needed by the agents as they can reconstruct the aead term directly; it exists so that the adversary can learn the plaintext of messages after revealing a key.

There are three actions, each of which concern all the relevant state variables, $\mathsf{RKeys}(S_i, S_r, E_i, E_r, Q, C)$, $\mathsf{IKeys}(S_i, S_r, E_i, E_r, Q, C)$, and $\mathsf{RConfirm}(S_i, S_r, E_i, E_r, Q, C)$, associated with the three states in the protocol, which are, respectively: the responder after receiving the first message from the initiator and replying, the initiator after receiving the second message from the responder and deriving transport keys, and the responder after receiving the first transport message from the initiator, thereby confirming the transport keys.

We consider an adversary able to compromise three types of secrets at any time, corresponding to the following actions in the trace: $\mathsf{Reveal}(S_i)$ and $\mathsf{Reveal}(S_r)$ correspond to a compromise of either peer's static key, $\mathsf{Reveal}(E_i)$ and $\mathsf{Reveal}(E_r)$ correspond to a compromise of either peer's ephemeral key, and $\mathsf{Reveal}(Q)$ corresponds to a compromise of the pre-shared key. $\mathsf{Reveal}(Q)$ may also correspond to the pre-shared key mode not being used at all, in which case the WireGuard protocol uses an all zero key, which is equivalent to the adversary simply knowing it in advance. Additionally, for secrecy properties we reference a built-in action $\mathsf{K}(C)$. This action is yielded by a rule equivalent to $\mathsf{In}(C)$ in the premises, and is referenced to add a constraint that the adversary must be able to construct the term $C$ to violate secrecy (and thus that $C$ is secret if no such trace exists).

Wireguard assumes that agents share their public keys and optionally a pre-shared symmetic key before the key exchange occurs, so rather than receiving public keys over the adversary-controlled network, agents add the public key of their peers directly by taking in $\mathsf{Agent}(S^{pub})$ and $\mathsf{Agent}(Q)$ facts directly in the premises. These facts are persistent, as multiple agents might have the same peer, and are created by rules which just generate fresh values. This is equivalent to peers receiving public and pre-shared keys out-of-band before the protocol begins.

The model breaks agent state into two parts, an invariant portion and a mutable portion. The mutable portion is stored in a linear fact $\mathsf{State}(id, x)$, containing a fresh identifier (to uniquely bind it to the relevant invariant

fact) and a generic term representing the current state machine state of the agent (e.g. that a handshake is in progress with a particular ephemeral key and hash chain). The invariants are contained instead in a persistent fact $\mathsf{StateInvariants}(id, Q, S_i^{priv}, S_r^{pub}, \mathrm{DH}(S_i^{priv}, S_r^{pub}))$, containing the same identifier, the relevant pre-shared symmetric key, initiator private key, and responder public key, as well as the precomputed static-static Diffie-Hellman (included to reduce the amount of computation done by Tamarin). Splitting up the state in this way allows Tamarin to immediately link the terms in the invariant to the point where the agents were set up—whereas solving a single linear fact containing all state terms would only give the state transitions that could have occurred immediately prior.

## 4.3 Verified Properties of the Model

The Tamarin model includes the ability of the adversary to compromise keys at any time, so we typically prove the contrapositive of the properties enumerated in Section 3. For example, where a property might state, "when $S_i$ and $E_r$ are compromised, the initiator achieves agreement with the responder," we instead prove that if the initiator does not achieve agreement with the responder then there must have been compromise of some other set of keys. Tamarin's "spthy" description language is here converted into standard first order logic, with one exception: the modifier $@\, t_1$ binds the preceding expression to a particular time $t_1$. Times specifiers like $t_1$ may be temporally compared to other times, such as in the statement $t_1 < t_2$. With this background established, the following properties are verified by the model:

### 4.3.1 Agreement

The key agreement properties stated in Section 3.1 are covered by the following three lemmas. First, from the initiator's perspective, we prove that disagreement implies that either $S_r$ was compromised beforehand, or both $S_i$ and $E_i$ were compromised beforehand.

**Lemma 1** (Unmatched initiator session requires $S_r$ or $(S_i, E_i)$ compromise, as well as $Q$ compromise)**.**

$$\begin{aligned}\forall S_i, S_r, E_i, E_r, Q, C, t_1 \,.\, \mathsf{IKeys}(S_i, S_r, E_i, E_r, Q, C) @ t_1 \wedge \\ \neg\,(\exists t_2 < t_1 \,.\, \mathsf{RKeys}(S_i, S_r, E_i, E_r, Q, C) @ t_2) \\ \implies ((\exists t_2 < t_1 \,.\, \mathsf{Reveal}(S_r) @ t_2) \vee \\ (\exists t_2 < t_1, t_3 < t_1 \,.\, \mathsf{Reveal}(S_i) @ t_2 \wedge \mathsf{Reveal}(E_i) @ t_3)) \wedge \\ (\exists t_2 < t_1 \,.\, \mathsf{Reveal}(Q) @ t_2)\end{aligned}$$

Similarly, from the responder's perspective, disagreement after the first transport message implies either $S_i$ or both $S_r$ and $E_r$ were compromised before that message.

**Lemma 2** (Unmatched responder session requires $S_i$ or $(S_r, E_r)$ compromise, as well as $Q$ compromise)**.**

$$\begin{aligned}\forall S_i, S_r, E_i, E_r, Q, C, t_1 \,.\, \mathsf{RConfirm}(S_i, S_r, E_i, E_r, Q, C) \mathbin{@} t_1 \wedge \\ \neg\,(\exists t_2 < t_1 \,.\, \mathsf{IKeys}(S_i, S_r, E_i, E_r, Q, C) \mathbin{@} t_2) \\ \implies ((\exists t_2 < t_1 \,.\, \mathsf{Reveal}(S_i) \mathbin{@} t_2) \vee \\ (\exists t_2 < t_1, t_3 < t_1 \,.\, \mathsf{Reveal}(S_r) \mathbin{@} t_2 \wedge \mathsf{Reveal}(E_r) \mathbin{@} t_3)) \wedge \\ (\exists t_2 < t_1 \,.\, \mathsf{Reveal}(Q) \mathbin{@} t_2)\end{aligned}$$

Finally, we show that if there is a session key, it necessarily came from the same set of identity keys, thereby preventing an unknown key-share attack:

**Lemma 3** (Session derivation implies key agreement)**.**

$$\begin{aligned}\forall S_i, S_i', S_r, S_i', E_i, E_i', E_r, E_r', Q, Q', C, t_1, t_2 \,. \\ \mathsf{IKeys}(S_i, S_r, E_i, E_r, Q, C) \mathbin{@} t_1 \wedge \mathsf{RKeys}(S_i', S_r', E_i', E_r', Q', C) \mathbin{@} t_2 \\ \implies S_i = S_i' \wedge S_r = S_r' \wedge E_i = E_i' \wedge E_r = E_r' \wedge Q = Q'\end{aligned}$$

#### 4.3.2 Secrecy

For the secrecy properties in Section 3.2, we prove a secrecy property conditioned on key agreement. Specifically, if I and R agree on their keys (even if R has not yet confirmed them), then either the adversary never learns the key $C$ they derived, or the adversary compromised both the static and ephemeral of one of the agents.

**Lemma 4** (Key secrecy unless $(S_i, E_i)$ or $(S_r, E_r)$, as well as $Q$, are compromised)**.**

$$\begin{aligned}\forall S_i, S_r, E_i, E_r, Q, C, t_1, t_2 \,. \\ \mathsf{IKeys}(S_i, S_r, E_i, E_r, Q, C) \mathbin{@} t_1 \wedge \mathsf{RKeys}(S_i, S_r, E_i, E_r, Q, C) \mathbin{@} t_2 \\ \implies \neg\,(\exists t_3 \,.\, \mathsf{K}(C) \mathbin{@} t_3) \vee \\ ((\exists t_3 \,.\, \mathsf{Reveal}(Q) \mathbin{@} t_3) \wedge \\ ((\exists t_3, t_4 \,.\, \mathsf{Reveal}(S_r) \mathbin{@} t_3 \wedge \mathsf{Reveal}(E_r) \mathbin{@} t_4) \vee \\ (\exists t_3, t_4 \,.\, \mathsf{Reveal}(S_i) \mathbin{@} t_3 \wedge \mathsf{Reveal}(E_i) \mathbin{@} t_4)))\end{aligned}$$

#### 4.3.3 Uniqueness

Whereas uniqueness in Section 3.3 is stated in terms of whether terms are fresh, in the Tamarin model any agent reaching an "agreement" action must have generated at least their own ephemeral fresh; recall that we do not model a broken randomness source that produces adversary-controlled values, just one which can be revealed by the adversary. Thus, we prove that an agent who negotiated a particular key can only do so once, even using different ephemeral and pre-shared keys, with no restriction on adversary compromise rules.

**Lemma 5** (Initiator session uniqueness)**.**

$$\forall S_i, S_r, E_i, E_r, Q, C, t_1 \,.\, \mathsf{IKeys}(S_i, S_r, E_i, E_r, Q, C) \mathbin{@} t_1 \implies \neg\,(\exists E_i', E_r', Q', t_2 \,.\, \mathsf{IKeys}(S_i, S_r, E_i', E_r', Q', C) \mathbin{@} t_2)$$

**Lemma 6** (Responder session uniqueness)**.**

$$\forall S_i, S_r, E_i, E_r, Q, C, t_1 \,.\, \mathsf{RConfirm}(S_i, S_r, E_i, E_r, Q, C) \mathbin{@} t_1 \implies \neg\,(\exists E_i', E_r', Q', t_2 \,.\, \mathsf{RConfirm}(S_i, S_r, E_i', E_r', Q', C) \mathbin{@} t_2)$$

#### 4.3.4 Identity Hiding

Identity hiding is difficult to model directly because it requires describing what an adversary can learn from a particular message. We cannot prove anything about whether the adversary learned the public key of the initiator, because the adversary *does* know the public key of every agent. Instead, we model identity hiding through the use of a fresh "surrogate" term, $\sigma$, which is added to the first message in the same place as the initiators public key; we then prove that the adversary did not learn $\sigma$ from the first message.

It is important to note that this proves a more limited property than the identity hiding discussed in Section 3.4. For example, the public static key of $S_i$, unlike $\sigma$ is not fresh each message. It may therefore still be possible for the adversary to link multiple initiator messages back to the same identity while still fulfilling the property below.

**Lemma 7** (Initiator identity (surrogate) hiding)**.**

$$\forall \sigma, S_i, S_r, E_i, E_r, Q, C, t_1 \,. \mathsf{RKeys}(S_i, S_r, E_i, E_r, Q, C) \mathbin{@} t_1 \wedge \sigma \mathbin{@} t_1 \implies (\exists t_2 \,.\, \mathsf{Reveal}(S_r) \mathbin{@} t_2) \vee (\exists t_2 \,.\, \mathsf{Reveal}(S_i) \mathbin{@} t_2) \vee (\exists t_2 \,.\, \mathsf{Reveal}(E_i) \mathbin{@} t_2)$$

## References


[1] Danny Dolev and Andrew Yao. "On the security of public key protocols". In: *IEEE Transactions on information theory* 29.2 (1983), pp. 198–208 (cit. on p. 6).

[2] Jason A. Donenfeld. "WireGuard: Next Generation Kernel Network Tunnel". In: *Proceedings of the 2017 Network and Distributed System Security Symposium*. NDSS'17. Document ID: 205657bc. San Diego, CA, Feb. 2017. ISBN: 1-891562-46-0. DOI: 10.14722/ndss.2017.23160. URL: https://www.wireguard.com/papers/wireguard.pdf (cit. on pp. 1, 2).

[3] Trevor Perrin. *The Noise Protocol Framework*. 2016. URL: http://noiseprotocol.org/noise.pdf (cit. on p. 1).

[4] Benedikt Schmidt. “Formal analysis of key exchange protocols and physical protocols”. Doctoral Thesis. ETH Zürich, 2012. DOI: 10.3929/ethz-a-009898924 (cit. on p. 5).

[5] Benedikt Schmidt et al. “Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties”. In: *CSF*. June 2012, pp. 78–94. DOI: 10.1109/CSF.2012.25 (cit. on pp. 5, 6).

[6] *Tamarin Prover Manual*. URL: https://tamarin-prover.github.io/manual/ (cit. on p. 5).