A Mechanised Cryptographic Proof of the WireGuard Virtual Private Network Protocol

Benjamin Lipp, Bruno Blanchet, Karthik Bhargavan (INRIA Paris, Prosecco)

June 18, 2019

4th IEEE European Symposium on Security and Privacy

The WireGuard Virtual Private Network (VPN)

Protocol and implementation in progress since 2015

The WireGuard Virtual Private Network (VPN)

Protocol and implementation in progress since 2015

- uses modern cryptography
- no cryptographic agility (unlike e. g., TLS)

WIREGUARD®
FAST, MODERN, SECURE VPN TUNNEL

The WireGuard Virtual Private Network (VPN)

Protocol and implementation in progress since 2015

- uses modern cryptography
- no cryptographic agility (unlike e. g., TLS)
- works directly over UDP
- only a few thousand lines of code

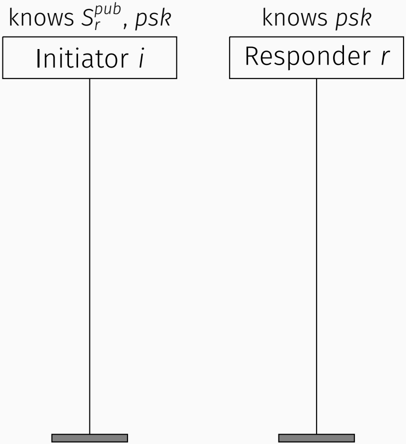
The WireGuard Virtual Private Network (VPN)

Protocol and implementation in progress since 2015

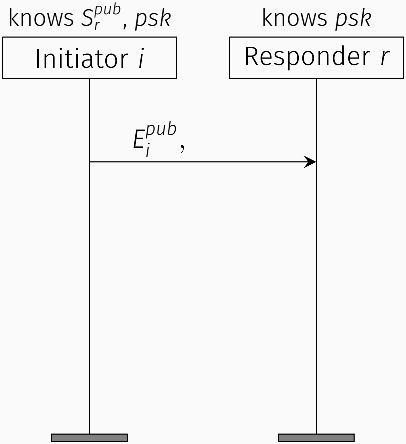
- uses modern cryptography
- no cryptographic agility (unlike e. g., TLS)
- works directly over UDP
- only a few thousand lines of code
- ongoing integration into the Linux kernel
- aims to replace OpenVPN and IPsec
- VPN providers are starting to adopt it

WIREGUARD®
FAST, MODERN, SECURE VPN TUNNEL

WireGuard’s Main Protocol: Noise IKpsk2 (simplified)



WireGuard's Main Protocol: Noise IKpsk2 (simplified)

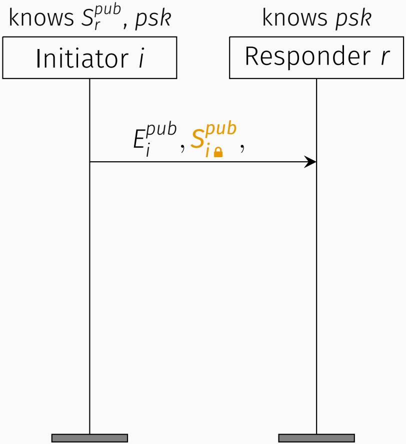


$$C_1 \leftarrow \mathrm{hkdf}_1(C_0 = \mathrm{const}_C, E_i^{pub})$$

$$C_2 \| k_1 \leftarrow \mathrm{hkdf}_2(C_1, \mathrm{dh}(E_i^{priv}, S_r^{pub}))$$

$$H_2 \leftarrow \mathrm{hash}(\mathrm{hash}(\mathrm{const}_H \| S_r^{pub}) \| E_i^{pub})$$

WireGuard's Main Protocol: Noise IKpsk2 (simplified)



$$C_1 \leftarrow \mathrm{hkdf}_1(C_0 = \mathrm{const}_C, E_i^{pub})$$

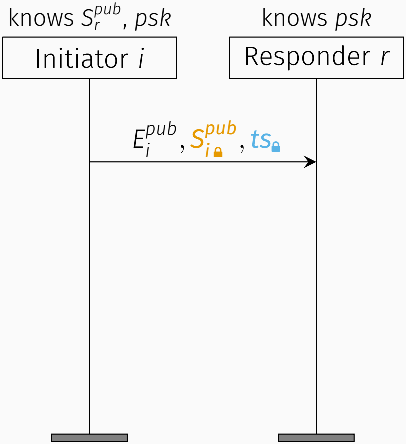
$$C_2 \| k_1 \leftarrow \mathrm{hkdf}_2(C_1, \mathrm{dh}(E_i^{priv}, S_r^{pub}))$$

$$H_2 \leftarrow \mathrm{hash}(\mathrm{hash}(\mathrm{const}_H \| S_r^{pub}) \| E_i^{pub})$$

$$S_i^{pub} \leftarrow \mathrm{aenc}(k_1, 0, S_i^{pub}, H_2)$$

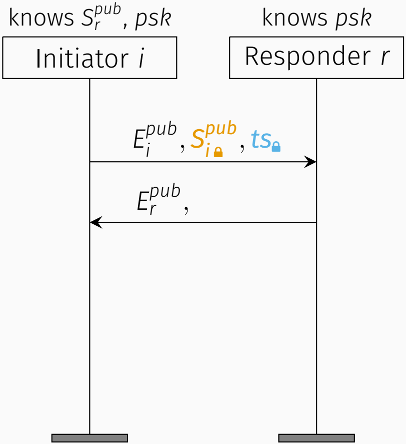
$$C_3 \| k_2 \leftarrow \mathrm{hkdf}_2(C_2, \mathrm{dh}(S_i^{priv}, S_r^{pub}))$$

WireGuard's Main Protocol: Noise IKpsk2 (simplified)



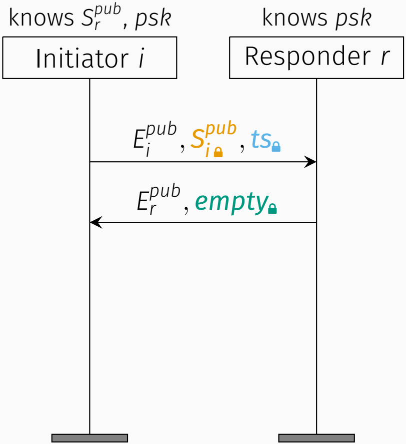
$$C_1 \leftarrow \mathrm{hkdf}_1(C_0 = \mathrm{const}_C, E_i^{pub})$$
$$C_2 \| k_1 \leftarrow \mathrm{hkdf}_2(C_1, \mathrm{dh}(E_i^{priv}, S_r^{pub}))$$
$$H_2 \leftarrow \mathrm{hash}(\mathrm{hash}(\mathrm{const}_H \| S_r^{pub}) \| E_i^{pub})$$
$$S_i^{pub} \leftarrow \mathrm{aenc}(k_1, 0, S_i^{pub}, H_2)$$
$$C_3 \| k_2 \leftarrow \mathrm{hkdf}_2(C_2, \mathrm{dh}(S_i^{priv}, S_r^{pub}))$$
$$ts \leftarrow \mathrm{aenc}(k_2, 0, \mathrm{timestamp}(), H_3)$$

WireGuard’s Main Protocol: Noise IKpsk2 (simplified)



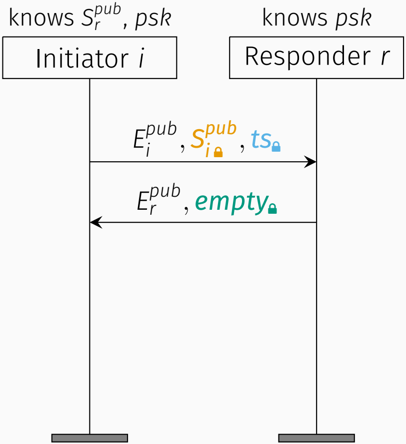
$$
\begin{aligned}
C_1 &\leftarrow \mathrm{hkdf}_1(C_0 = \mathrm{const}_C, E_i^{pub})\\
C_2 \| k_1 &\leftarrow \mathrm{hkdf}_2(C_1, \mathrm{dh}(E_i^{priv}, S_r^{pub}))\\
H_2 &\leftarrow \mathrm{hash}(\mathrm{hash}(\mathrm{const}_H \| S_r^{pub}) \| E_i^{pub})\\
S_i^{pub} &\leftarrow \mathrm{aenc}(k_1, 0, S_i^{pub}, H_2)\\
C_3 \| k_2 &\leftarrow \mathrm{hkdf}_2(C_2, \mathrm{dh}(S_i^{priv}, S_r^{pub}))\\
ts &\leftarrow \mathrm{aenc}(k_2, 0, \mathrm{timestamp}(), H_3)\\
C_4 &\leftarrow \mathrm{hkdf}_1(C_3, E_r^{pub})\\
C_5 &\leftarrow \mathrm{hkdf}_1(C_4, \mathrm{dh}(E_r^{priv}, E_i^{pub}))\\
C_6 &\leftarrow \mathrm{hkdf}_1(C_5, \mathrm{dh}(E_r^{priv}, S_i^{pub}))\\
C_7 \| \pi \| k_3 &\leftarrow \mathrm{hkdf}_3(C_6, psk)
\end{aligned}
$$

WireGuard's Main Protocol: Noise IKpsk2 (simplified)



$$
\begin{aligned}
C_1 &\leftarrow \mathrm{hkdf}_1(C_0 = \mathrm{const}_C, E_i^{pub}) \\
C_2 \| k_1 &\leftarrow \mathrm{hkdf}_2(C_1, \mathrm{dh}(E_i^{priv}, S_r^{pub})) \\
H_2 &\leftarrow \mathrm{hash}(\mathrm{hash}(\mathrm{const}_H \| S_r^{pub}) \| E_i^{pub}) \\
S^{pub}_{i\,\text{🔒}} &\leftarrow \mathrm{aenc}(k_1, 0, S_i^{pub}, H_2) \\
C_3 \| k_2 &\leftarrow \mathrm{hkdf}_2(C_2, \mathrm{dh}(S_i^{priv}, S_r^{pub})) \\
ts_{\text{🔒}} &\leftarrow \mathrm{aenc}(k_2, 0, \mathrm{timestamp}(), H_3) \\
C_4 &\leftarrow \mathrm{hkdf}_1(C_3, E_r^{pub}) \\
C_5 &\leftarrow \mathrm{hkdf}_1(C_4, \mathrm{dh}(E_r^{priv}, E_i^{pub})) \\
C_6 &\leftarrow \mathrm{hkdf}_1(C_5, \mathrm{dh}(E_r^{priv}, S_i^{pub})) \\
C_7 \| \pi \| k_3 &\leftarrow \mathrm{hkdf}_3(C_6, psk) \\
empty_{\text{🔒}} &\leftarrow \mathrm{aenc}(k_3, 0, empty, H_6)
\end{aligned}
$$

WireGuard's Main Protocol: Noise IKpsk2 (simplified)



$$C_1 \leftarrow \mathrm{hkdf}_1(C_0 = \mathrm{const}_C, E_i^{pub})$$

$$C_2 \| k_1 \leftarrow \mathrm{hkdf}_2(C_1, \mathrm{dh}(E_i^{priv}, S_r^{pub}))$$

$$H_2 \leftarrow \mathrm{hash}(\mathrm{hash}(\mathrm{const}_H \| S_r^{pub}) \| E_i^{pub})$$

$$S_i^{pub} \leftarrow \mathrm{aenc}(k_1, 0, S_i^{pub}, H_2)$$

$$C_3 \| k_2 \leftarrow \mathrm{hkdf}_2(C_2, \mathrm{dh}(S_i^{priv}, S_r^{pub}))$$

$$ts \leftarrow \mathrm{aenc}(k_2, 0, \mathrm{timestamp}(), H_3)$$

$$C_4 \leftarrow \mathrm{hkdf}_1(C_3, E_r^{pub})$$

$$C_5 \leftarrow \mathrm{hkdf}_1(C_4, \mathrm{dh}(E_r^{priv}, E_i^{pub}))$$

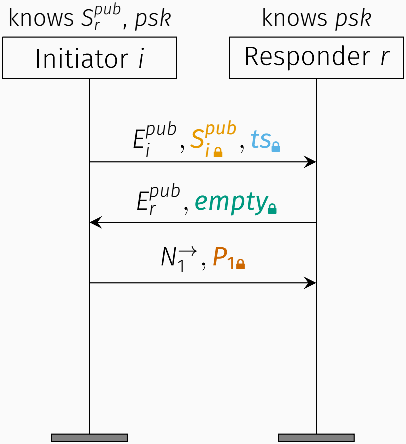
$$C_6 \leftarrow \mathrm{hkdf}_1(C_5, \mathrm{dh}(E_r^{priv}, S_i^{pub}))$$

$$C_7 \| \pi \| k_3 \leftarrow \mathrm{hkdf}_3(C_6, psk)$$

$$empty \leftarrow \mathrm{aenc}(k_3, 0, empty, H_6)$$

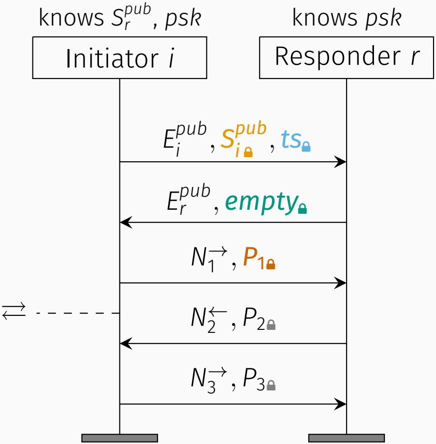
$$T^{\rightarrow} \| T^{\leftarrow} \leftarrow \mathrm{hkdf}_2(C_7, empty)$$

WireGuard's Main Protocol: Noise IKpsk2 (simplified)



$$C_1 \leftarrow \mathrm{hkdf}_1(C_0 = \mathrm{const}_C, E_i^{pub})$$
$$C_2 \| k_1 \leftarrow \mathrm{hkdf}_2(C_1, \mathrm{dh}(E_i^{priv}, S_r^{pub}))$$
$$H_2 \leftarrow \mathrm{hash}(\mathrm{hash}(\mathrm{const}_H \| S_r^{pub}) \| E_i^{pub})$$
$$S_i^{pub} \leftarrow \mathrm{aenc}(k_1, 0, S_i^{pub}, H_2)$$
$$C_3 \| k_2 \leftarrow \mathrm{hkdf}_2(C_2, \mathrm{dh}(S_i^{priv}, S_r^{pub}))$$
$$ts \leftarrow \mathrm{aenc}(k_2, 0, \mathrm{timestamp}(), H_3)$$
$$C_4 \leftarrow \mathrm{hkdf}_1(C_3, E_r^{pub})$$
$$C_5 \leftarrow \mathrm{hkdf}_1(C_4, \mathrm{dh}(E_r^{priv}, E_i^{pub}))$$
$$C_6 \leftarrow \mathrm{hkdf}_1(C_5, \mathrm{dh}(E_r^{priv}, S_i^{pub}))$$
$$C_7 \| \pi \| k_3 \leftarrow \mathrm{hkdf}_3(C_6, psk)$$
$$empty \leftarrow \mathrm{aenc}(k_3, 0, empty, H_6)$$
$$T^{\rightarrow} \| T^{\leftarrow} \leftarrow \mathrm{hkdf}_2(C_7, empty)$$
$$P_1 \leftarrow \mathrm{aenc}(T^{\rightarrow}, N_1^{\rightarrow} = 0, P_1, empty)$$

WireGuard's Main Protocol: Noise IKpsk2 (simplified)



$$C_1 \leftarrow \mathrm{hkdf}_1(C_0 = \mathrm{const}_C, E_i^{pub})$$

$$C_2 \| k_1 \leftarrow \mathrm{hkdf}_2(C_1, \mathrm{dh}(E_i^{priv}, S_r^{pub}))$$

$$H_2 \leftarrow \mathrm{hash}(\mathrm{hash}(\mathrm{const}_H \| S_r^{pub}) \| E_i^{pub})$$

$$S_i^{pub} \leftarrow \mathrm{aenc}(k_1, 0, S_i^{pub}, H_2)$$

$$C_3 \| k_2 \leftarrow \mathrm{hkdf}_2(C_2, \mathrm{dh}(S_i^{priv}, S_r^{pub}))$$

$$ts \leftarrow \mathrm{aenc}(k_2, 0, \mathrm{timestamp}(), H_3)$$

$$C_4 \leftarrow \mathrm{hkdf}_1(C_3, E_r^{pub})$$

$$C_5 \leftarrow \mathrm{hkdf}_1(C_4, \mathrm{dh}(E_r^{priv}, E_i^{pub}))$$

$$C_6 \leftarrow \mathrm{hkdf}_1(C_5, \mathrm{dh}(E_r^{priv}, S_i^{pub}))$$

$$C_7 \| \pi \| k_3 \leftarrow \mathrm{hkdf}_3(C_6, psk)$$

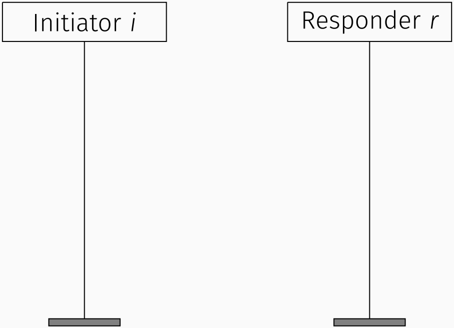
$$empty \leftarrow \mathrm{aenc}(k_3, 0, empty, H_6)$$

$$T^{\rightarrow} \| T^{\leftarrow} \leftarrow \mathrm{hkdf}_2(C_7, empty)$$

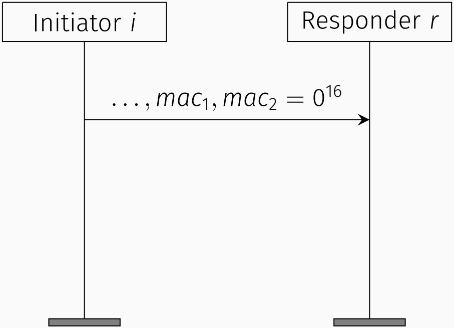
$$P_1 \leftarrow \mathrm{aenc}(T^{\rightarrow}, N_1^{\rightarrow} = 0, P_1, empty)$$

WireGuard's DoS Protection During Handshake (simplified)

1) timestamp in the first message



WireGuard’s DoS Protection During Handshake (simplified)

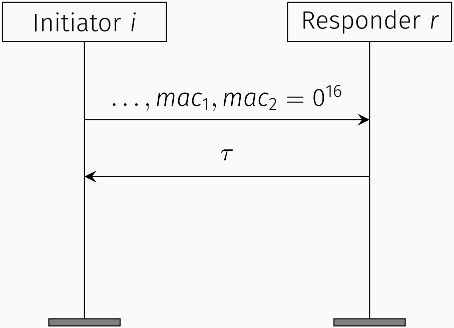


1) timestamp in the first message

2) mac_1 in all handshake messages

$$mac_1 \leftarrow \text{mac}\Big(\underbrace{\text{hash}(\text{label}_{\text{mac1}} \| S_r^{pub})}_{\text{MAC key}}, \underbrace{msg_\alpha}_{\text{msg bytes}}\Big)$$

WireGuard's DoS Protection During Handshake (simplified)

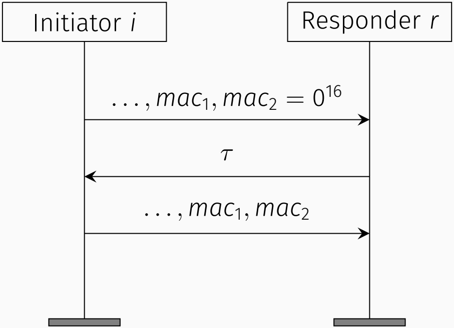


1) timestamp in the first message

2) mac_1 in all handshake messages

$$mac_1 \leftarrow \mathrm{mac}\Big(\underbrace{\mathrm{hash}(\mathrm{label}_{\mathrm{mac1}} \| S_r^{pub})}_{\text{MAC key}}, \underbrace{msg_\alpha}_{\text{msg bytes}}\Big)$$

WireGuard's DoS Protection During Handshake (simplified)



1) timestamp in the first message

2) mac_1 in all handshake messages

$$mac_1 \leftarrow \mathrm{mac}\Big(\underbrace{\mathrm{hash}(\mathrm{label}_{\mathrm{mac1}} \| S_r^{pub})}_{\text{MAC key}}, \underbrace{msg_\alpha}_{\text{msg bytes}}\Big)$$

3) Non-zero mac_2 in response to cookie messages:

$$mac_2 \leftarrow \mathrm{mac}(\tau, \underbrace{msg_\beta}_{\text{msg bytes incl. } mac_1})$$

Security Goals of WireGuard's Secure Channel Protocol

Classic key exchange and secure channel properties

Security Goals of WireGuard's Secure Channel Protocol

Classic key exchange and secure channel properties

Secrecy
- Secrecy
- Forward secrecy

Security Goals of WireGuard's Secure Channel Protocol

Classic key exchange and secure channel properties

Secrecy	• Secrecy • Forward secrecy
Agreement	• Mutual authentication

Security Goals of WireGuard's Secure Channel Protocol

Classic key exchange and secure channel properties

Secrecy	• Secrecy • Forward secrecy
Agreement	• Mutual authentication • Session uniqueness • Channel binding

Security Goals of WireGuard's Secure Channel Protocol

Classic key exchange and secure channel properties

Secrecy	• Secrecy • Forward secrecy
Agreement	• Mutual authentication • Session uniqueness • Channel binding • Resistance against key compromise impersonation (KCI) • Resistance against identity mis-binding

Security Goals of WireGuard's Secure Channel Protocol

Classic key exchange and secure channel properties

Secrecy

- Secrecy
- Forward secrecy

Agreement

- Mutual authentication
- Session uniqueness
- Channel binding
- Resistance against key compromise impersonation (KCI)
- Resistance against identity mis-binding

Additional properties in WireGuard

- Resistance against denial of service
- Identity hiding

Our Contributions

Mechanised cryptographic proof of WireGuard using CryptoVerif, analysing:

- the entire protocol, including transport data messages

Our Contributions

Mechanised cryptographic proof of WireGuard using CryptoVerif, analysing:

- the entire protocol, including transport data messages
- classic properties of key exchange and secure channels

Our Contributions

Mechanised cryptographic proof of WireGuard using CryptoVerif, analysing:

- the entire protocol, including transport data messages
- classic properties of key exchange and secure channels
- identity hiding
- resistance against denial of service

Our Contributions

Mechanised cryptographic proof of WireGuard using CryptoVerif, analysing:

- the entire protocol, including transport data messages
- classic properties of key exchange and secure channels
- identity hiding
- resistance against denial of service

Reusable contributions:

- Precise model of the Curve25519 elliptic curve for Diffie-Hellman

Our Contributions

Mechanised cryptographic proof of WireGuard using CryptoVerif, analysing:

- the entire protocol, including transport data messages
- classic properties of key exchange and secure channels
- identity hiding
- resistance against denial of service

Reusable contributions:

- Precise model of the Curve25519 elliptic curve for Diffie-Hellman
- Indifferentiability lemmas for chains of random oracle calls (e. g., chains of HMAC or HKDF function calls)

Our Contributions

Mechanised cryptographic proof of WireGuard using CryptoVerif, analysing:

- the entire protocol, including transport data messages
- classic properties of key exchange and secure channels
- identity hiding
- resistance against denial of service

Reusable contributions:

- Precise model of the Curve25519 elliptic curve for Diffie-Hellman
- Indifferentiability lemmas for chains of random oracle calls (e. g., chains of HMAC or HKDF function calls)

Related work: DowlingPaterson'18, DonenfeldMilner'18 on WireGuard
KobeissiNicolasBhargavan'19, Suter-Dörig'18, Girol'19 on IKpsk2

The CryptoVerif Automatic Protocol Prover

Proof assistant for game-based cryptographic proofs

The CryptoVerif Automatic Protocol Prover

Proof assistant for game-based cryptographic proofs

initial game

The CryptoVerif Automatic Protocol Prover

Proof assistant for game-based cryptographic proofs

initial game $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ intermediate games

The CryptoVerif Automatic Protocol Prover

Proof assistant for game-based cryptographic proofs

initial game $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ intermediate games $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ final game

The CryptoVerif Automatic Protocol Prover

Proof assistant for game-based cryptographic proofs

initial game $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ intermediate games $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ final game

security properties easily provable here

The CryptoVerif Automatic Protocol Prover

Proof assistant for game-based cryptographic proofs

$$\text{initial game} \underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}} \text{intermediate games} \underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}} \text{final game}$$

security properties easily provable here

- security games in a probabilistic process calculus
- generates next game from previous game, given transformation

The CryptoVerif Automatic Protocol Prover

Proof assistant for game-based cryptographic proofs

initial game $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ intermediate games $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ final game

security properties easily provable here

- security games in a probabilistic process calculus
- generates next game from previous game, given transformation
- built-in proof strategy
- supports secrecy and correspondence properties

The CryptoVerif Automatic Protocol Prover

Proof assistant for game-based cryptographic proofs

initial game $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ intermediate games $\underbrace{\rightarrow \cdots \rightarrow}_{\text{transformations}}$ final game

security properties easily provable here

- security games in a probabilistic process calculus
- generates next game from previous game, given transformation
- built-in proof strategy
- supports secrecy and correspondence properties
- successful termination ⇒ asymptotic security
- exact security given by probability bound

Cryptographic Assumptions

- the BLAKE2s hash function (in hash, hkdf, mac)

Cryptographic Assumptions

- the BLAKE2s hash function (in hash, hkdf, mac)
    - random oracle

Cryptographic Assumptions

- the BLAKE2s hash function (in hash, hkdf, mac)
    - random oracle
- the ChaCha20Poly1305 AEAD

Cryptographic Assumptions

- the BLAKE2s hash function (in hash, hkdf, mac)
    - random oracle
- the ChaCha20Poly1305 AEAD
    - IND-CPA- and INT-CTXT-secure

Cryptographic Assumptions

- the BLAKE2s hash function (in hash, hkdf, mac)
    - random oracle
- the ChaCha20Poly1305 AEAD
    - IND-CPA- and INT-CTXT-secure
    - for identity hiding: preserves secrecy of associated data

Cryptographic Assumptions

- the BLAKE2s hash function (in hash, hkdf, mac)
  - random oracle
- the ChaCha20Poly1305 AEAD
  - IND-CPA- and INT-CTXT-secure
  - for identity hiding: preserves secrecy of associated data
- Curve25519 Diffie-Hellman

Cryptographic Assumptions

- the BLAKE2s hash function (in hash, hkdf, mac)
  - random oracle
- the ChaCha20Poly1305 AEAD
  - IND-CPA- and INT-CTXT-secure
  - for identity hiding: preserves secrecy of associated data
- Curve25519 Diffie-Hellman
  - Gap Diffie-Hellman in the appropriate subgroup

Model of Environment and Corruption

- any number of available parties (i. e., static key pairs)

Model of Environment and Corruption

- any number of available parties (i. e., static key pairs)
    - two explicit honest parties i and r

Model of Environment and Corruption

- any number of available parties (i. e., static key pairs)
    - two explicit honest parties i and r
    - adversary plays all other honest and dishonest parties

Model of Environment and Corruption

- any number of available parties (i. e., static key pairs)
  - two explicit honest parties *i* and *r*
  - adversary plays all other honest and dishonest parties
- polynomial number of sessions
- polynomial number of transport data messages in a session

Model of Environment and Corruption

- any number of available parties (i. e., static key pairs)
  - two explicit honest parties i and r
  - adversary plays all other honest and dishonest parties
- polynomial number of sessions
- polynomial number of transport data messages in a session
- prove security properties for *clean* sessions between i and r

Model of Environment and Corruption

- any number of available parties (i. e., static key pairs)
  - two explicit honest parties *i* and *r*
  - adversary plays all other honest and dishonest parties
- polynomial number of sessions
- polynomial number of transport data messages in a session
- prove security properties for *clean* sessions between *i* and *r*
  - a session is clean if it's not trivially broken
    i. e. if *not all* secrets of one party are compromised

The Random Oracle in the Key Derivation

The Random Oracle in the Key Derivation

Reminder: a random oracle returns

The Random Oracle in the Key Derivation

Reminder: a random oracle returns

- a fresh random value on new input

The Random Oracle in the Key Derivation

Reminder: a random oracle returns

- a fresh random value on new input
- the same value than before on previously seen input

The Random Oracle in the Key Derivation

Reminder: a random oracle returns

- a fresh random value on new input
- the same value than before on previously seen input

Mechanised analysis has to treat *all cases of collision*

The Random Oracle in the Key Derivation

Reminder: a random oracle returns

- a fresh random value on new input
- the same value than before on previously seen input

Mechanised analysis has to treat *all cases of collision*
$\Rightarrow$ CryptoVerif creates nested branches for hkdf inputs

The Random Oracle in the Key Derivation

Reminder: a random oracle returns

- a fresh random value on new input
- the same value than before on previously seen input

Mechanised analysis has to treat *all cases of collision*

⇒ CryptoVerif creates nested branches for hkdf inputs

(remember: 8 *dependent* random oracle calls with inputs $v_0, \ldots, v_7$)

The Random Oracle in the Key Derivation

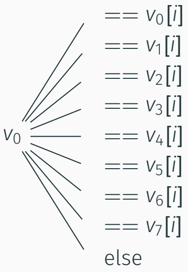
Reminder: a random oracle returns

- a fresh random value on new input
- the same value than before on previously seen input

Mechanised analysis has to treat *all cases of collision*

$\Rightarrow$ CryptoVerif creates nested branches for hkdf inputs

(remember: 8 *dependent* random oracle calls with inputs $v_0, \ldots, v_7$)



The Random Oracle in the Key Derivation

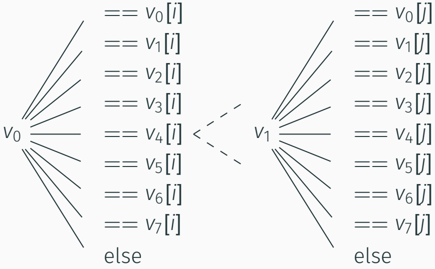
Reminder: a random oracle returns

- a fresh random value on new input
- the same value than before on previously seen input

Mechanised analysis has to treat *all cases of collision*

$\Rightarrow$ CryptoVerif creates nested branches for hkdf inputs

(remember: 8 *dependent* random oracle calls with inputs $v_0, \ldots, v_7$)



The Random Oracle in the Key Derivation

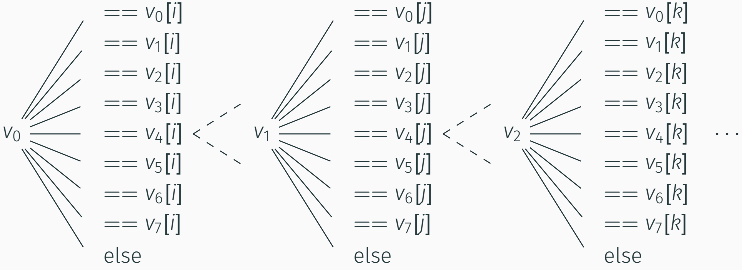
Reminder: a random oracle returns

- a fresh random value on new input
- the same value than before on previously seen input

Mechanised analysis has to treat *all cases of collision*

$\Rightarrow$ CryptoVerif creates nested branches for hkdf inputs

(remember: 8 *dependent* random oracle calls with inputs $v_0, \ldots, v_7$)



Simplification of the Key Derivation's HKDF Chain

8 chained calls to
one random oracle.

$$
\begin{aligned}
C &\leftarrow \mathrm{const} \\
C &\leftarrow \mathrm{hkdf}(C, v_0) \\
C \| k_1 &\leftarrow \mathrm{hkdf}(C, v_1) \\
C \| k_2 &\leftarrow \mathrm{hkdf}(C, v_2) \\
C &\leftarrow \mathrm{hkdf}(C, v_3) \\
C &\leftarrow \mathrm{hkdf}(C, v_4) \\
C &\leftarrow \mathrm{hkdf}(C, v_5) \\
C \| \pi \| k_3 &\leftarrow \mathrm{hkdf}(C, v_6) \\
T^{\rightarrow} \| T^{\leftarrow} &\leftarrow \mathrm{hkdf}(C, \epsilon)
\end{aligned}
$$

Simplification of the Key Derivation's HKDF Chain

8 chained calls to
one random oracle.

$$
\begin{aligned}
C &\leftarrow \mathrm{const} \\
C &\leftarrow \mathrm{hkdf}(C, v_0) \\
C \| k_1 &\leftarrow \mathrm{hkdf}(C, v_1) \\
C \| k_2 &\leftarrow \mathrm{hkdf}(C, v_2) \\
C &\leftarrow \mathrm{hkdf}(C, v_3) \\
C &\leftarrow \mathrm{hkdf}(C, v_4) \\
C &\leftarrow \mathrm{hkdf}(C, v_5) \\
C \| \pi \| k_3 &\leftarrow \mathrm{hkdf}(C, v_6) \\
T^{\rightarrow} \| T^{\leftarrow} &\leftarrow \mathrm{hkdf}(C, \epsilon)
\end{aligned}
$$

3 independent calls to
3 *independent* random oracles.

$$
\begin{aligned}
k_1 &\leftarrow \mathrm{chain}_1(v_0, v_1) \\
k_2 &\leftarrow \mathrm{chain}_2(v_0, v_1, v_2) \\
\pi \| k_3 \| T^{\rightarrow} \| T^{\leftarrow} &\leftarrow \mathrm{chain}_6(v_0, v_1, v_2, v_3, v_4, v_5, v_6)
\end{aligned}
$$

Simplification of the Key Derivation's HKDF Chain

8 chained calls to
one random oracle.

$$
\begin{aligned}
C &\leftarrow \mathrm{const} \\
C &\leftarrow \mathrm{hkdf}(C, v_0) \\
C \| k_1 &\leftarrow \mathrm{hkdf}(C, v_1) \\
C \| k_2 &\leftarrow \mathrm{hkdf}(C, v_2) \\
C &\leftarrow \mathrm{hkdf}(C, v_3) \\
C &\leftarrow \mathrm{hkdf}(C, v_4) \\
C &\leftarrow \mathrm{hkdf}(C, v_5) \\
C \| \pi \| k_3 &\leftarrow \mathrm{hkdf}(C, v_6) \\
T^{\rightarrow} \| T^{\leftarrow} &\leftarrow \mathrm{hkdf}(C, \epsilon)
\end{aligned}
$$

3 independent calls to
3 *independent* random oracles.

$$
\begin{aligned}
k_1 &\leftarrow \mathrm{chain}_1(v_0, v_1) \\
k_2 &\leftarrow \mathrm{chain}_2(v_0, v_1, v_2) \\
\pi \| k_3 \| T^{\rightarrow} \| T^{\leftarrow} &\leftarrow \mathrm{chain}_6(v_0, v_1, v_2, v_3, v_4, v_5, v_6)
\end{aligned}
$$

Idea: extract only whenever the protocol needs a key

Simplification of the Key Derivation's HKDF Chain

Indifferentiable in any context:
(manually proved, some lemmas with CryptoVerif)

8 chained calls to
one random oracle.

$$
\begin{aligned}
C &\leftarrow \text{const} \\
C &\leftarrow \text{hkdf}(C, v_0) \\
C \| k_1 &\leftarrow \text{hkdf}(C, v_1) \\
C \| k_2 &\leftarrow \text{hkdf}(C, v_2) \\
C &\leftarrow \text{hkdf}(C, v_3) \\
C &\leftarrow \text{hkdf}(C, v_4) \\
C &\leftarrow \text{hkdf}(C, v_5) \\
C \| \pi \| k_3 &\leftarrow \text{hkdf}(C, v_6) \\
T^{\rightarrow} \| T^{\leftarrow} &\leftarrow \text{hkdf}(C, \epsilon)
\end{aligned}
$$

3 independent calls to
3 *independent* random oracles.

$$
\begin{aligned}
k_1 &\leftarrow \text{chain}_1(v_0, v_1) \\
k_2 &\leftarrow \text{chain}_2(v_0, v_1, v_2) \\
\pi \| k_3 \| T^{\rightarrow} \| T^{\leftarrow} &\leftarrow \text{chain}_6(v_0, v_1, v_2, v_3, v_4, v_5, v_6)
\end{aligned}
$$

Idea: extract only whenever the protocol needs a key

A Precise Model of Curve25519

A Precise Model of Curve25519

- Curve25519 is *not* a prime order group

A Precise Model of Curve25519

- Curve25519 is *not* a prime order group
  - group of order kq where $k = 8$ and q large prime, base point g_0 of order q used for honest key generation

A Precise Model of Curve25519

- Curve25519 is *not* a prime order group
    - group of order kq where $k = 8$ and q large prime,
      base point g_0 of order q used for honest key generation
    - twist of curve: group of order $k'q'$ where $k' = 4$ and q' large prime

A Precise Model of Curve25519

- Curve25519 is *not* a prime order group
  - group of order kq where $k = 8$ and q large prime, base point g_0 of order q used for honest key generation
  - twist of curve: group of order $k'q'$ where $k' = 4$ and q' large prime
- curve points are represented by only the x coordinate

A Precise Model of Curve25519

- Curve25519 is *not* a prime order group
  - group of order kq where $k = 8$ and q large prime,
    base point g_0 of order q used for honest key generation
  - twist of curve: group of order $k'q'$ where $k' = 4$ and q' large prime
- curve points are represented by only the x coordinate

Updated model in the paper's long version:
`https://cryptoverif.inria.fr/WireGuard`

A Precise Model of Curve25519

- Curve25519 is *not* a prime order group
  - group of order kq where $k = 8$ and q large prime, base point g_0 of order q used for honest key generation
  - twist of curve: group of order $k'q'$ where $k' = 4$ and q' large prime
- curve points are represented by only the x coordinate
- incoming public keys are *not verified*

Updated model in the paper's long version:
`https://cryptoverif.inria.fr/WireGuard`

A Precise Model of Curve25519

- Curve25519 is *not* a prime order group
  - group of order kq where $k = 8$ and q large prime, base point g_0 of order q used for honest key generation
  - twist of curve: group of order $k'q'$ where $k' = 4$ and q' large prime
- curve points are represented by only the x coordinate
- incoming public keys are *not verified*
  - could be on curve or twist, and in subgroup generated by g_0 or not

Updated model in the paper's long version:
`https://cryptoverif.inria.fr/WireGuard`

A Precise Model of Curve25519

- Curve25519 is *not* a prime order group
  - group of order kq where $k = 8$ and q large prime,
    base point g_0 of order q used for honest key generation
  - twist of curve: group of order $k'q'$ where $k' = 4$ and q' large prime
- curve points are represented by only the x coordinate
- incoming public keys are *not verified*
  - could be on curve or twist, and in subgroup generated by g_0 or not
- for each public key in the subgroup, there is a small number of *equivalent* public keys outside the subgroup

Updated model in the paper's long version:
`https://cryptoverif.inria.fr/WireGuard`

A Precise Model of Curve25519

- Curve25519 is *not* a prime order group
    - group of order kq where $k = 8$ and q large prime,
      base point g_0 of order q used for honest key generation
    - twist of curve: group of order $k'q'$ where $k' = 4$ and q' large prime
- curve points are represented by only the x coordinate
- incoming public keys are *not verified*
    - could be on curve or twist, and in subgroup generated by g_0 or not
- for each public key in the subgroup, there is a small number of *equivalent* public keys outside the subgroup
    - produced by adding a low-order point to a public key

Updated model in the paper's long version:
`https://cryptoverif.inria.fr/WireGuard`

A Precise Model of Curve25519

- Curve25519 is *not* a prime order group
  - group of order kq where $k = 8$ and q large prime, base point g_0 of order q used for honest key generation
  - twist of curve: group of order $k'q'$ where $k' = 4$ and q' large prime
- curve points are represented by only the x coordinate
- incoming public keys are *not verified*
  - could be on curve or twist, and in subgroup generated by g_0 or not
- for each public key in the subgroup, there is a small number of *equivalent* public keys outside the subgroup
  - produced by adding a low-order point to a public key
  - they lead to the same shared secret

Updated model in the paper's long version:
`https://cryptoverif.inria.fr/WireGuard`

A Theoretical Attack leading to Identity Mis-Binding

A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

Two honest parties deriving the same traffic keys in some sessions

A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

Two honest parties deriving the same traffic keys in some sessions

- agree on each other's identities

A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

Two honest parties deriving the same traffic keys in some sessions

- agree on each other's identities
- even if one or both of them have been interacting with a dishonest party or a honest party with compromised keys

A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

Two honest parties deriving the same traffic keys in some sessions

- agree on each other's identities
- even if one or both of them have been interacting with a dishonest party or a honest party with compromised keys

Theoretical Attack:

A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

Two honest parties deriving the same traffic keys in some sessions

- agree on each other's identities
- even if one or both of them have been interacting with a dishonest party or a honest party with compromised keys

Theoretical Attack:

- Let S_i, S_r, E_i, E_r, and *psk* be compromised.

A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

Two honest parties deriving the same traffic keys in some sessions

- agree on each other's identities
- even if one or both of them have been interacting with a dishonest party or a honest party with compromised keys

Theoretical Attack:

- Let S_i, S_r, E_i, E_r, and *psk* be compromised.
- Adversary constructs $S_i' \neq S_i, S_r' \neq S_r$ as *different but equivalent* static keys

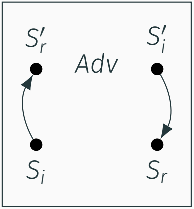
A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

Two honest parties deriving the same traffic keys in some sessions

- agree on each other's identities
- even if one or both of them have been interacting with a dishonest party or a honest party with compromised keys

Theoretical Attack:



- Let S_i, S_r, E_i, E_r, and *psk* be compromised.
- Adversary constructs $S_i' \neq S_i, S_r' \neq S_r$ as *different but equivalent* static keys

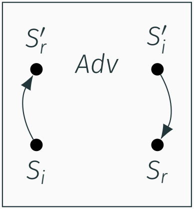
A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

Two honest parties deriving the same traffic keys in some sessions

- agree on each other's identities
- even if one or both of them have been interacting with a dishonest party or a honest party with compromised keys

Theoretical Attack:



- Let S_i, S_r, E_i, E_r, and *psk* be compromised.
- Adversary constructs $S_i' \neq S_i, S_r' \neq S_r$ as *different but equivalent* static keys

$\Rightarrow$ The two sessions derive the same traffic keys but are between different parties.

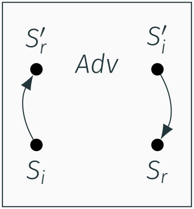
A Theoretical Attack leading to Identity Mis-Binding

Definition: Resistance against Identity Mis-Binding

Two honest parties deriving the same traffic keys in some sessions

- agree on each other's identities
- even if one or both of them have been interacting with a dishonest party or a honest party with compromised keys

Theoretical Attack:



- Let S_i, S_r, E_i, E_r, and *psk* be compromised.
- Adversary constructs $S'_i \neq S_i, S'_r \neq S_r$ as *different but equivalent* static keys

$\Rightarrow$ The two sessions derive the same traffic keys but are between different parties.

Mitigation: include static public keys S_i^{pub} and S_r^{pub} into key derivation

Identity Hiding: Known Weaknesses

Identity Hiding: Known Weaknesses

WireGuard values DoS resistance over privacy

Identity Hiding: Known Weaknesses

WireGuard values DoS resistance over privacy

- knowing a candidate public key S_Y^{pub}, adversary can compare $mac_1 = \mathrm{mac}(\mathrm{hash}(\mathrm{label}_{\mathrm{mac1}} \| S_Y^{pub}), msg_\alpha)$

Identity Hiding: Known Weaknesses

WireGuard values DoS resistance over privacy

- knowing a candidate public key S_Y^{pub}, adversary can compare $mac_1 = \mathrm{mac}(\mathrm{hash}(\mathrm{label}_{\mathrm{mac1}} \| S_Y^{pub}), msg_\alpha)$
- similar test possible on the encrypted cookie

Identity Hiding: Known Weaknesses

WireGuard values DoS resistance over privacy

- knowing a candidate public key S_Y^{pub}, adversary can compare $mac_1 = \mathrm{mac}(\mathrm{hash}(\mathrm{label}_{\mathrm{mac1}} \| S_Y^{pub}), msg_\alpha)$
- similar test possible on the encrypted cookie
- at least VPN provider's keys usually public

Identity Hiding: Known Weaknesses

WireGuard values DoS resistance over privacy

- knowing a candidate public key S_Y^{pub}, adversary can compare $mac_1 = \text{mac}(\text{hash}(\text{label}_{\text{mac1}} \| S_Y^{pub}), msg_\alpha)$
- similar test possible on the encrypted cookie
- at least VPN provider's keys usually public

$\Rightarrow$ the MACs weaken identity hiding properties

Identity Hiding: Known Weaknesses

WireGuard values DoS resistance over privacy

- knowing a candidate public key S_Y^{pub}, adversary can compare $mac_1 = \mathrm{mac}(\mathrm{hash}(\mathrm{label}_{\mathrm{mac1}} \| S_Y^{pub}), msg_\alpha)$
- similar test possible on the encrypted cookie
- at least VPN provider's keys usually public

$\Rightarrow$ the MACs weaken identity hiding properties

- $S_{i🔒}^{pub}$ not forward secret, would require a round-trip more

Identity Hiding: Our Contribution on Noise IKpsk2

Identity Hiding: Our Contribution on Noise IKpsk2

$$S_{i\,🔒}^{pub} \leftarrow \mathrm{aenc}(k_1, 0, S_i^{pub}, H_2)$$

Identity Hiding: Our Contribution on Noise IKpsk2

$$H_2 \leftarrow \text{hash}(\text{hash}(\text{const}_H \| S_r^{pub}) \| E_i^{pub})$$
$$S_{i🔒}^{pub} \leftarrow \text{aenc}(k_1, 0, S_i^{pub}, H_2)$$

Identity Hiding: Our Contribution on Noise IKpsk2

$$H_2 \leftarrow \mathrm{hash}(\mathrm{hash}(\mathrm{const}_H \| S_r^{pub}) \| E_i^{pub})$$
$$S_{i\,\text{\faLock}}^{pub} \leftarrow \mathrm{aenc}(k_1, 0, S_i^{pub}, H_2)$$

- Adversary could compare additional data of $S_{i\,\text{\faLock}}^{pub}$ with $\mathrm{hash}(\mathrm{hash}(\mathrm{const}_H \| S_Y^{pub}) \| E_i^{pub})$

Identity Hiding: Our Contribution on Noise IKpsk2

$$H_2 \leftarrow \text{hash}(\text{hash}(\text{const}_H \| S_r^{pub}) \| E_i^{pub})$$

$$S_{i\text{🔒}}^{pub} \leftarrow \text{aenc}(k_1, 0, S_i^{pub}, H_2)$$

- Adversary could compare additional data of $S_{i\text{🔒}}^{pub}$ with $\text{hash}(\text{hash}(\text{const}_H \| S_Y^{pub}) \| E_i^{pub})$

$\Rightarrow$ AEAD needs to preserve AD secrecy

Identity Hiding: Our Contribution on Noise IKpsk2

$$H_2 \leftarrow \text{hash}(\text{hash}(\text{const}_H \| S_r^{pub}) \| E_i^{pub})$$

$$S_{i\text{🔒}}^{pub} \leftarrow \text{aenc}(k_1, 0, S_i^{pub}, H_2)$$

- Adversary could compare additional data of $S_{i\text{🔒}}^{pub}$ with hash(hash(const$_H \| S_Y^{pub}) \| E_i^{pub}$)

$\Rightarrow$ AEAD needs to preserve AD secrecy

- We prove:
  knowing S_{A1}^{pub}, S_{A2}^{pub}, S_{B1}^{pub}, S_{B2}^{pub}, adversary cannot distinguish

Identity Hiding: Our Contribution on Noise IKpsk2

$$H_2 \leftarrow \text{hash}(\text{hash}(\text{const}_H \| S_r^{pub}) \| E_i^{pub})$$
$$S_{i🔒}^{pub} \leftarrow \text{aenc}(k_1, 0, S_i^{pub}, H_2)$$

- Adversary could compare additional data of $S_{i🔒}^{pub}$ with $\text{hash}(\text{hash}(\text{const}_H \| S_Y^{pub}) \| E_i^{pub})$

$\Rightarrow$ AEAD needs to preserve AD secrecy

- We prove:
  knowing $S_{A1}^{pub}, S_{A2}^{pub}, S_{B1}^{pub}, S_{B2}^{pub}$, adversary cannot distinguish
    - public key S_{A1}^{pub} initiating sessions with S_{B1}^{pub}
    - public key S_{A2}^{pub} initiating sessions with S_{B2}^{pub}

Metrics

Numbers based on the largest variant of our model:

- approx. 1300 lines of model code
- 36 proof instructions
- 168 games produced by CryptoVerif
- 16 minutes runtime on Intel Xeon 3.6 GHz

Metrics

Numbers based on the largest variant of our model:

- approx. 1300 lines of model code
- 36 proof instructions
- 168 games produced by CryptoVerif
- 16 minutes runtime on Intel Xeon 3.6 GHz

Compared to manual proof from DowlingPaterson'18: 11 games

Metrics

Numbers based on the largest variant of our model:

- approx. 1300 lines of model code
- 36 proof instructions
- 168 games produced by CryptoVerif
- 16 minutes runtime on Intel Xeon 3.6 GHz

Compared to manual proof from DowlingPaterson’18: 11 games

- uses a different Diffie-Hellman assumption (PRF-ODH)

Metrics

Numbers based on the largest variant of our model:

- approx. 1300 lines of model code
- 36 proof instructions
- 168 games produced by CryptoVerif
- 16 minutes runtime on Intel Xeon 3.6 GHz

Compared to manual proof from DowlingPaterson'18: 11 games

- uses a different Diffie-Hellman assumption (PRF-ODH)
- CryptoVerif formally encodes many small steps, separately

Results Compared to Goals

Classic key exchange and secure channel properties:

Secrecy	• Secrecy (by proving message indistinguishability) • Forward secrecy
Agreement	• Mutual authentication (as of 2nd message) • Session uniqueness • Channel binding • Resistance against key compromise impersonation (KCI) • Resistance against identity mis-binding (except theoretical attack)

Additional properties in WireGuard:

- Resistance against denial of service
  (no replay of 1st msg, cookie enforces round-trip)
- Identity hiding (weak)

Conclusion and Main Take-Aways

- WireGuard *protocol* is cryptographically safe
  - weak identity hiding
- more context in key derivation prevents theoretical attack
  - (teaser: and makes proofs easier)
- chains of random oracle calls can be reduced to fewer calls
- precise model for Curve25519 and Curve448

- detailed comparison to other analyses of WireGuard in our paper's related work section
- the long version and our models with an updated version of Curve25519 are available at: `https://cryptoverif.inria.fr/WireGuard`