A Unified Symbolic Analysis of WireGuard

Pascal Lafourcade[1, 2] Dhekra Mahmoud[1,2] Sylvain Ruhault[3]

[1] Université Clermont Auvergne,

[2] Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes,

[3] Agence Nationale de la Sécurité des Systèmes d'Information

February 27, 2024

NDSS SYMPOSIUM/2024

Presented by Internet Society

#NDSSSymposium2024

Formal Verification of security protocols

Formal Verification of security protocols

Manual proofs

- Error prone
- Tedious
- Active Adversaries
- Guarantees on security ?

Formal Verification of security protocols

Software tools

- ▶ Automated & semi-automated
- ▶ Formal proofs
- ▶ Handle protocols' complexity
- ▶ Dedicated approaches
- ▶ **Symbolic** & Computational

PROVERIF TAMARIN

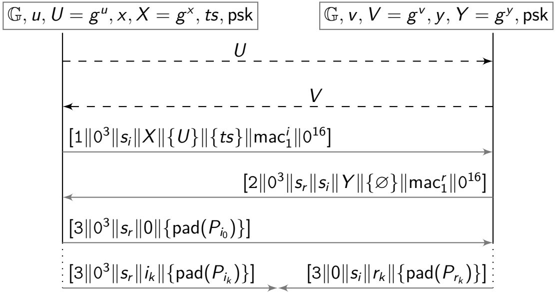
SAPIC$^+$

Manual proofs

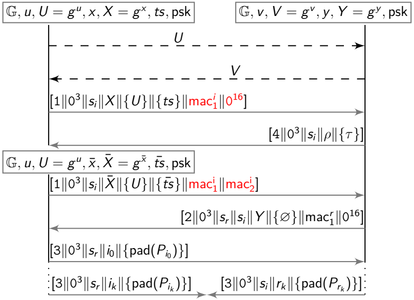
- ▶ Error prone
- ▶ Tedious
- ▶ Active Adversaries
- ▶ Guarantees on security ?

Target models for *WireGuard*

Without cookie



With cookie



- $u, U = g^u, v, V = g^v$ ⇝ static keys, $x, X = g^x, y, Y = g^y$ ⇝ ephemeral keys, psk ⇝ pre-shared key
- ts timestamp, s_i, s_r ⇝ session identifiers, i_* ⇝ counters, P_* ⇝ plaintexts
- $\{\cdot\}$ ⇝ encryption
- ρ ⇝ nonce, τ ⇝ cookie

Current analyses

Symbolic

- 2018: J. A. Donenfeld and K. Milner, “Formal verification of the WireGuard protocol” *WireGuard*
- 2019: N. Kobeissi, G. Nicolas, and K. Bhargavan, “Noise explorer: Fully automated modeling and verification for arbitrary Noise protocols” *IKpsk2*
- 2020: G. Girol, L. Hirschi, R. Sasse, D. Jackson, C. Cremers, and D. A. Basin, “A spectral analysis of Noise: A comprehensive, automated, formal analysis of Diffie-Hellman protocols” *IKpsk2*

Computationnal

- 2018: B. Dowling and K. G. Paterson, “A cryptographic analysis of the WireGuard protocol” *WireGuard*
- 2019: B. Lipp, B. Blanchet, and K. Bhargavan, “A mechanised cryptographic proof of the “WireGuard virtual private network protocol” *WireGuard*

Threats

- Static private key reveal / set
- Ephemeral private key reveal / set
- PSK reveal / set
- Static key distribution corruption

Security Properties

- Message agreement
- Key secrecy (incl. PFS)
- Anonymity

Current analyses

What is the scope of *WireGuard* analyses ?

- Lazy answer: full protocol !

Are `IKpsk2` analyses applicable to *WireGuard* ?

- Lazy answer: yes !

Are threat model equivalent ? Are all verification done ?

- Lazy answer: come on, we have a proof, it's enough !

Current analyses

What is the scope of *WireGuard* analyses ?

- Lazy answer: full protocol !
- **Correct answer: should be studied !**

Are `IKpsk2` analyses applicable to *WireGuard* ?

- Lazy answer: yes !
- **Correct answer: should be studied !**

Are threat model equivalent ? Are all verification done ?

- Lazy answer: come on, we have a proof, it's enough !
- **Correct answer: should be studied !** Adversary can
  - get u, v, x, y, psk before / after protocol execution
  - set u, v, x, y, psk
  - compromise U and V distribution
  - and combine ($2^{5+5+5+2} = 2^{17} = 131072$ combinations per property) !

Symbolic analyis of *WireGuard* (TAMARIN)

2018: J. A. Donenfeld and K. Milner, "Formal verification of the WireGuard protocol"

$\boxed{\mathbb{G}, u, U = g^u, \mathbf{V}, x, X = g^x, \mathsf{psk}}$ $\boxed{\mathbb{G}, v, V = g^v, \mathbf{U}, y, Y = g^y, \mathsf{psk}}$

$\xrightarrow{\varnothing}$

$\xleftarrow{\varnothing}$

$\xrightarrow{[1 \| 0^3 \| s_i \| X \| \{U\} \| \{ts\} \| \mathbf{MAC1} \| \mathbf{MAC2}]}$

$\xleftarrow{[2 \| 0^3 \| s_r \| s_i \| Y \| \{\varnothing\} \| \mathbf{MAC1} \| \mathbf{MAC2}]}$

$\xrightarrow{[3 \| 0^3 \| s_r \| \mathbf{o} \| \{\mathsf{pad}(P_i)\}]}$

$\xrightarrow{\varnothing} \xleftarrow{\varnothing}$

Threats

- Static private key reveal ✓ / set ✗
- Ephemeral private key reveal ✓ / set ✗
- PSK reveal ✓ / set ✗
- Static key distribution corruption ✗

Security Properties

- Message agreement ✓
- Key secrecy ✓ (PFS ✗)
- Anonymity ✓

Verified Combinations

- ✗

Our target threat model for *WireGuard*

Threats

- Static private key reveal ✓ / set ✓
- Ephemeral private key reveal ✓ / set ✓
- PSK reveal ✓ / set ✓
- Static key distribution corruption ✓
- New! Pre-computation reveal ✓ / set ✓

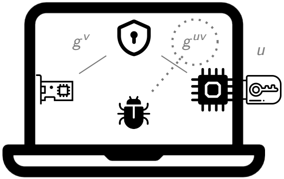
Pre-computation ?

- Static-static key :
  - Initiator $V^u = g^{uv}$
  - Responder $U^v = g^{uv}$

  before session begins, hence WireGuard maintains it.

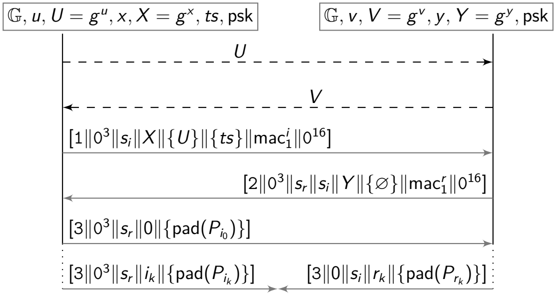
Compromise of g^{uv} is **weaker** than compromise of u or v:

- $u \wedge g^v \implies g^{uv}$
- however $g^v \wedge g^{uv} \not\Longrightarrow u$

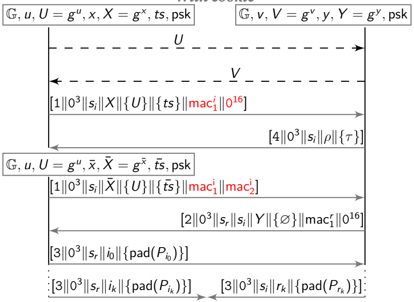


Our symbolic models of *WireGuard* (Tamarin, ProVerif, Sapic$^+$)

Without cookie



With cookie



Threats

- Static private key reveal ✓ / set ✓
- Ephemeral private key reveal ✓ / set ✓
- PSK reveal ✓ / set ✓
- Static key distribution corruption ✓
- New! Pre-computation reveal ✓ / set ✓

Security Properties

- Message agreement ✓
- Key secrecy ✓ (PFS ✓)
- Anonymity ✓

Verified Combinations

- New! 2^{21} per property ✓

Our results : necessary and sufficient conditions

- D_u, D_v: adversary corrupts public keys distribution
- $R_u, R_v, R_x, R_y, R_s, R_c$: adversary gets private keys (u, v, x, y), psk (s) or pre-comp. value (c)
- $R_u^*, R_v^*, R_s^*, R_c^*$: adversary gets private keys (u, v), psk (s) or pre-comp. value (c) after protocol execution (for PFS)

Results

- agreement of `RecHello` and `TransData` (R to I) messages hold ***unless***
  $(D_v \wedge R_s) \vee (R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x)$
- agreement of `TransData` (I to R) messages hold ***unless***
  $(D_u \wedge R_s) \vee (R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y)$
- Key Secrecy from Initiator's view, including PFS hold ***unless***
  $(D_v \wedge R_s) \vee (R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x) \vee (R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$
- Key Secrecy from Responder's view, including PFS hold ***unless***
  $(D_u \wedge R_s) \vee (R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y) \vee (R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$

Our results : interpretation

Results

- agreement of `RecHello` and `TransData` (R to I) messages hold ***unless*** $(D_v \wedge R_s) \vee (R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x)$
- agreement of `TransData` (I to R) messages hold ***unless*** $(D_u \wedge R_s) \vee (R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y)$
- Key Secrecy from Initiator's view, including PFS hold ***unless*** $(D_v \wedge R_s) \vee (R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x) \vee (R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$
- Key Secrecy from Responder's view, including PFS hold ***unless*** $(D_u \wedge R_s) \vee (R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y) \vee (R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$

Key distribution corruption

Agreement and key secrecy hold ***unless*** adversary:

- compromises U distribution **AND** gets psk
- **OR** compromises V distribution **AND** gets psk

$\Longrightarrow$ **Shall not be eluded !**

Our results : interpretation

Results

- agreement of `RecHello` and `TransData` (R to I) messages hold ***unless***
  $(D_v \wedge R_s) \vee (R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x)$
- agreement of `TransData` (I to R) messages hold ***unless***
  $(D_u \wedge R_s) \vee (R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y)$
- Key Secrecy from Initiator's view, including PFS hold ***unless***
  $(D_v \wedge R_s) \vee (R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x) \vee (R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$
- Key Secrecy from Responder's view, including PFS hold ***unless***
  $(D_u \wedge R_s) \vee (R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y) \vee (R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$

Pre-shared key

psk compromise is *necessary* to break all properties.
$\Longrightarrow$ **Shall be mandatory (and not optional) !**

Our results : interpretation

Results

- agreement of `RecHello` and `TransData` (R to I) messages hold ***unless***
  $(D_v \wedge R_s) \vee (R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x)$
- agreement of `TransData` (I to R) messages hold ***unless***
  $(D_u \wedge R_s) \vee (R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y)$
- Key Secrecy from Initiator's view, including PFS hold ***unless***
  $(D_v \wedge R_s) \vee (R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x) \vee (R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$
- Key Secrecy from Responder's view, including PFS hold ***unless***
  $(D_u \wedge R_s) \vee (R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y) \vee (R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$

Pre-computation

In some cases, R_c has same impact as R_u or R_v, although *weaker*.
$\Longrightarrow$ **Shall be removed !**

Anonymity

Claim: *Identity Hiding Forward Secrecy*

- *a compromise of the responder's private key and a traffic log of previous handshakes would enable an attacker to figure out who has sent handshakes*
- *it is possible to trial hash to guess whether or not a packet is intended for a particular responder*

(Identity hiding also proven in 2018 model with TAMARIN*)*

$\mathbb{G}, u, U = g^u, V_1, V_2, x, X = g^x, ts, \mathsf{psk}$ | $\mathbb{G}, \mathbf{v_*}, \mathbf{V_*} = \mathbf{g^{v_*}}, U, y, Y = g^y, \mathsf{psk}$

$[1 \| 0^3 \| s_i \| X \| \{U\} \| \{ts\} \| \mathsf{mac}_1^i \| 0^{16}]$

$\mathbf{mac(H(}V_1\mathbf{), [2\| \cdots \|\{\varnothing\}) \stackrel{?}{=} mac_1^i}$

$\mathbf{mac(H(}V_2\mathbf{), [2\| \cdots \|\{\varnothing\}) \stackrel{?}{=} mac_1^i}$

- $\{U\}$ is encrypted with g^{xv}, hence if v leaks then U is known.
- `InitHello` message is $[1 \| 0^3 \| s_i \| X \| \{U\} \| \{ts\} \| \mathbf{mac_1^i} \| 0^{16}]$
- $\mathsf{mac}_1^i = \mathrm{mac}(\mathrm{H}(V), [1 \| \cdots \| \{ts\}])$, where V is public $\Longrightarrow$ V can leak !

Anonymity

Claim: *Identity Hiding Forward Secrecy*

- *a compromise of the responder's private key and a traffic log of previous handshakes would enable an attacker to figure out who has sent handshakes*
- *it is possible to trial hash to guess whether or not a packet is intended for a particular responder*

(Identity hiding also proven in 2018 model with TAMARIN*)*

$\mathbb{G}, \mathbf{u}_*, \mathbf{U}_* = \mathbf{g}^{\mathbf{u}_*}, V, x, X = g^x, ts, \mathsf{psk}$

$\mathbb{G}, v, V = g^v, \mathbf{U}_1, \mathbf{U}_2, y, Y = g^y, \mathsf{psk}$

$[1 \| 0^3 \| s_i \| X \| \{U\} \| \{ts\} \| \mathsf{mac}_1^i \| 0^{16}]$ →

← $[2 \| 0^3 \| s_r \| s_i \| Y \| \{\varnothing\} \| \mathbf{mac}_1^r \| 0^{16}]$

$\mathsf{mac}(\mathsf{H}(U_1), [2\| \cdots \|\{\varnothing\}) \stackrel{?}{=} \mathsf{mac}_1^r$

$\mathsf{mac}(\mathsf{H}(U_2), [2\| \cdots \|\{\varnothing\}) \stackrel{?}{=} \mathsf{mac}_1^r$

However issue is the same for `RecHello` message ! **(explained in "A mechanised cryptographic proof of the WireGuard VPN protocol")**

- `RecHello` message is $[2 \| 0^3 \| s_r \| s_i \| Y \| \{\varnothing\} \| \mathbf{mac}_1^r \| 0^{16}]$
- $\mathsf{mac}_1^r = \mathsf{mac}(\mathsf{H}(U), [2\| \cdots \|\{\varnothing\}])$, where U is public $\Longrightarrow$ *U can leak !*

Anonymity

Claim: *Identity Hiding Forward Secrecy*

- *a compromise of the responder's private key and a traffic log of previous handshakes would enable an attacker to figure out who has sent handshakes*
- *it is possible to trial hash to guess whether or not a packet is intended for a particular responder*

(Identity hiding also proven in 2018 model with TAMARIN*)*

⇝ Reality: WireGuard does **not** provide anonymity at all (key compromise is not necessary) ...

Anonymity

Claim: *Identity Hiding Forward Secrecy*

- *a compromise of the responder's private key and a traffic log of previous handshakes would enable an attacker to figure out who has sent handshakes*
- *it is possible to trial hash to guess whether or not a packet is intended for a particular responder*

(Identity hiding also proven in 2018 model with TAMARIN*)*

$\rightsquigarrow$ Reality: WireGuard does **not** provide anonymity at all (key compromise is not necessary) ...

Proposed fixes

- Remove **mac** (i.e. use `IKpsk2`)
- Change **mac** computation :
  - $\mathbf{mac}_1^r = \mathsf{mac}(\mathsf{H}(U \| g^{uv}), [2 \| \cdots \| \{\varnothing\}])$
  - $\mathbf{mac}_1^r = \mathsf{mac}(\mathsf{H}(U \| \mathsf{psk}), [2 \| \cdots \| \{\varnothing\}])$

$\Longrightarrow$ With these fixes anonymity is **verified** with PROVERIF

Conclusion

- Currently WireGuard ensures:
  - Agreement
  - Key secrecy and PFS

- Recommandations for end users:
  - Use pre-shared key
  - Care about static key distribution
  - Do not rely on WireGuard for anonymity
- Recommandations for stakeholders:
  - Remove pre-computation
  - Fix anonymity

Conclusion

- Currently WireGuard ensures:
  - Agreement
  - Key secrecy and PFS

- Recommandations for end users:
  - Use pre-shared key
  - Care about static key distribution
  - Do not rely on WireGuard for anonymity
- Recommandations for stakeholders:
  - Remove pre-computation
  - Fix anonymity

- Complete model of WireGuard
- **Fix** for anonymity property
- Precise threat model, including initial key distribution and **pre-computations**
- Necessary and sufficient conditions
- Process with SAPIC$^+$, PROVERIF, TAMARIN

Artifact Evaluated
NDSS SYMPOSIUM
Available
Functional
Reproduced

Conclusion

- Currently WireGuard ensures:
  - Agreement
  - Key secrecy and PFS

- Recommandations for end users:
  - Use pre-shared key
  - Care about static key distribution
  - Do not rely on WireGuard for anonymity
- Recommandations for stakeholders:
  - Remove pre-computation
  - Fix anonymity

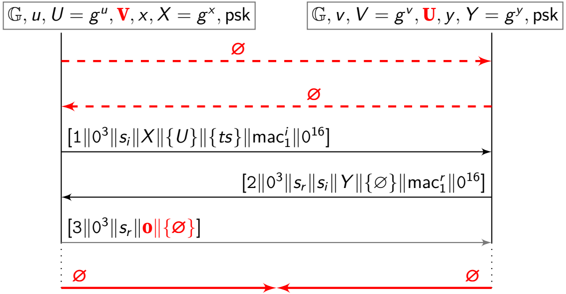
- Complete model of WireGuard
- **Fix** for anonymity property
- Precise threat model, including initial key distribution and **pre-computations**
- Necessary and sufficient conditions
- Process with SAPIC$^+$, PROVERIF, TAMARIN

Artifact Evaluated
NDSS
SYMPOSIUM
Available
Functional
Reproduced

- Thanks for your attention !
- Do you have questions ?

Computationnal analysis of *WireGuard* (manual)

2018: B. Dowling *et al.*, "A cryptographic analysis of the WireGuard protocol"



Threats

- Static private key reveal ✓ / set ✗
- Ephemeral private key reveal ✓ / set ✗
- PSK reveal ✓ / set ✗
- Static key distribution corruption ✗

Security Properties

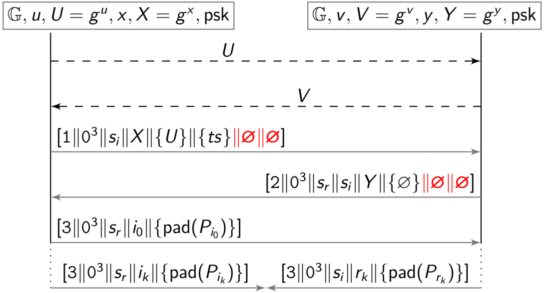
- Message agreement ✓
- Key secrecy ✓ (PFS ✗)
- Anonymity ✗

Verified Combinations

- ✗

Computationnal analysis of *WireGuard* (CryptoVerif)

2019: B. Lipp *et al.*, "A mechanised cryptographic proof of the WireGuard VPN protocol"



Threats

- Static private key reveal ✓ / set ✓
- Ephemeral private key reveal ✓ / set ✗
- PSK reveal ✓ / set ✓
- Static key distribution corruption ✓

Security Properties

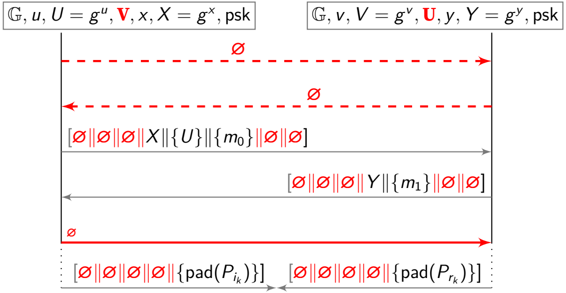
- Message agreement ✓
- Key secrecy ✓ (PFS ✓)
- Anonymity ✗

Verified Combinations

- ✗

Symbolic analysis of IKpsk2 (ProVerif)

2019: N. Kobeissi *et al.*, "Noise explorer: Fully automated modeling and verification for arbitrary Noise protocols"



Threats

- Static private key reveal ✓ / set ✗
- Ephemeral private key reveal ✗ / set ✗
- PSK reveal ✓ / set ✗
- Static key distribution corruption ✗

Security Properties

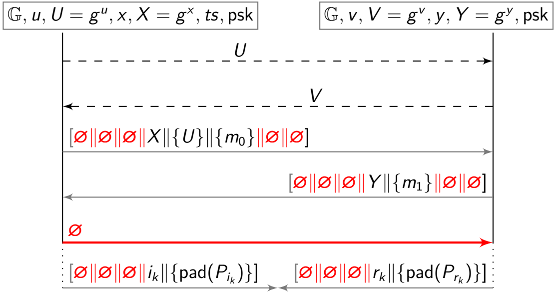
- Message agreement ✓
- Key secrecy ✓ (PFS ✓)
- Anonymity ✗

Verified Combinations

- ✗

Symbolic analysis of IKpsk2 (Tamarin)

2020: G. Girol *et al.*, "A spectral analysis of Noise: A comprehensive, automated, formal analysis of Diffie-Hellman protocols"



Threats

- Static private key reveal ✓ / set ✓
- Ephemeral private key reveal ✓ / set ✓
- PSK reveal ✓ / set ✓
- Static key distribution corruption ✓

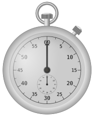
Security Properties

- Message agreement ✓
- Key secrecy ✓ (PFS ✓)
- Anonymity ✓

Verified Combinations

- ✓

Benchmarks



With a dedicated 256 cores server

- Evaluation of agreement and secrecy properties (ProVerif, Tamarin, Sapic$^+$) : 9 hours
- Evaluation of fix for anonymity, based on g^{uv} (ProVerif) : 12 hours
- Evaluation of fix for anonymity, based on psk (ProVerif) : 2 hours

Combinations

With pre-computation

Adversary can

- get $u, v, x, y, \mathrm{psk}, g^{uv}$ before / after protocol execution
- set $u, v, x, y, \mathrm{psk}, g^{uv}$ for Initiator and g^{uv} for Responder
- compromise U and V distribution
- and combine ($2^{6+6+7+2} = 2^{21} = 2097152$ combinations per property) !