Conclusion

- Work still in progress, next steps:
  - Analyse "false" queries
  - Link with TAMARIN prover
  - Mount concrete attacks
- Verified properties:
  - Agreement
  - Key and message secrecy
  - Anonymity
- Contributions:
  - Complete model of WireGuard
  - Precise threat model, including initial key distribution and precomputations
  - Process with SAPIC$^+$, PROVERIF, TAMARIN

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

A new symbolic analysis of WireGuard VPN

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,

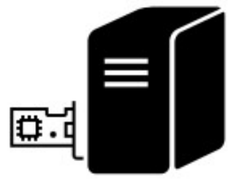
[3] Agence Nationale de la Sécurité des Systèmes d'Information

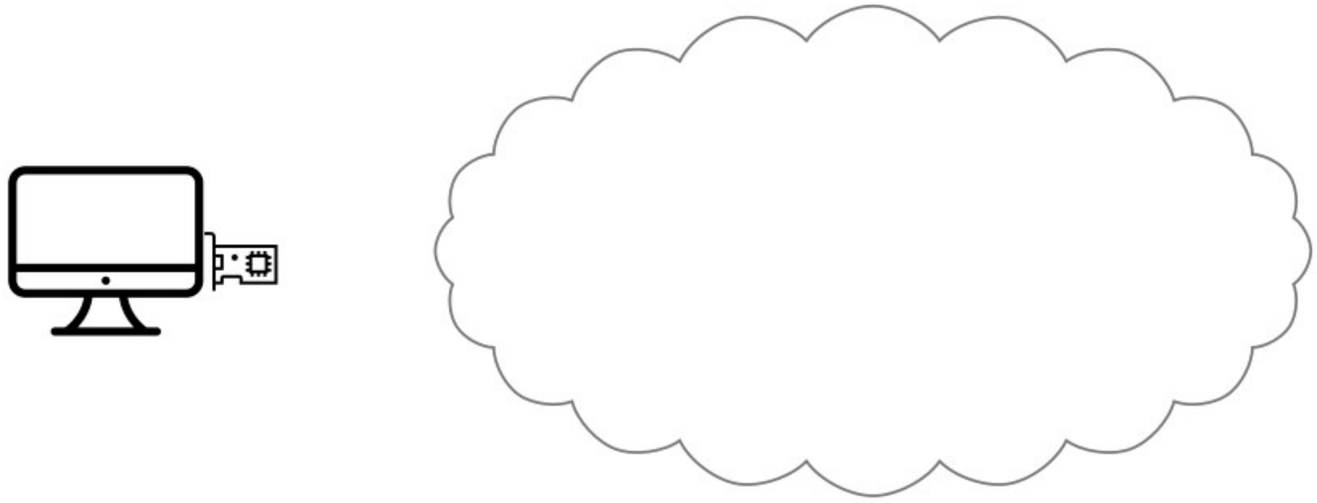
May 12, 2023

Agenda

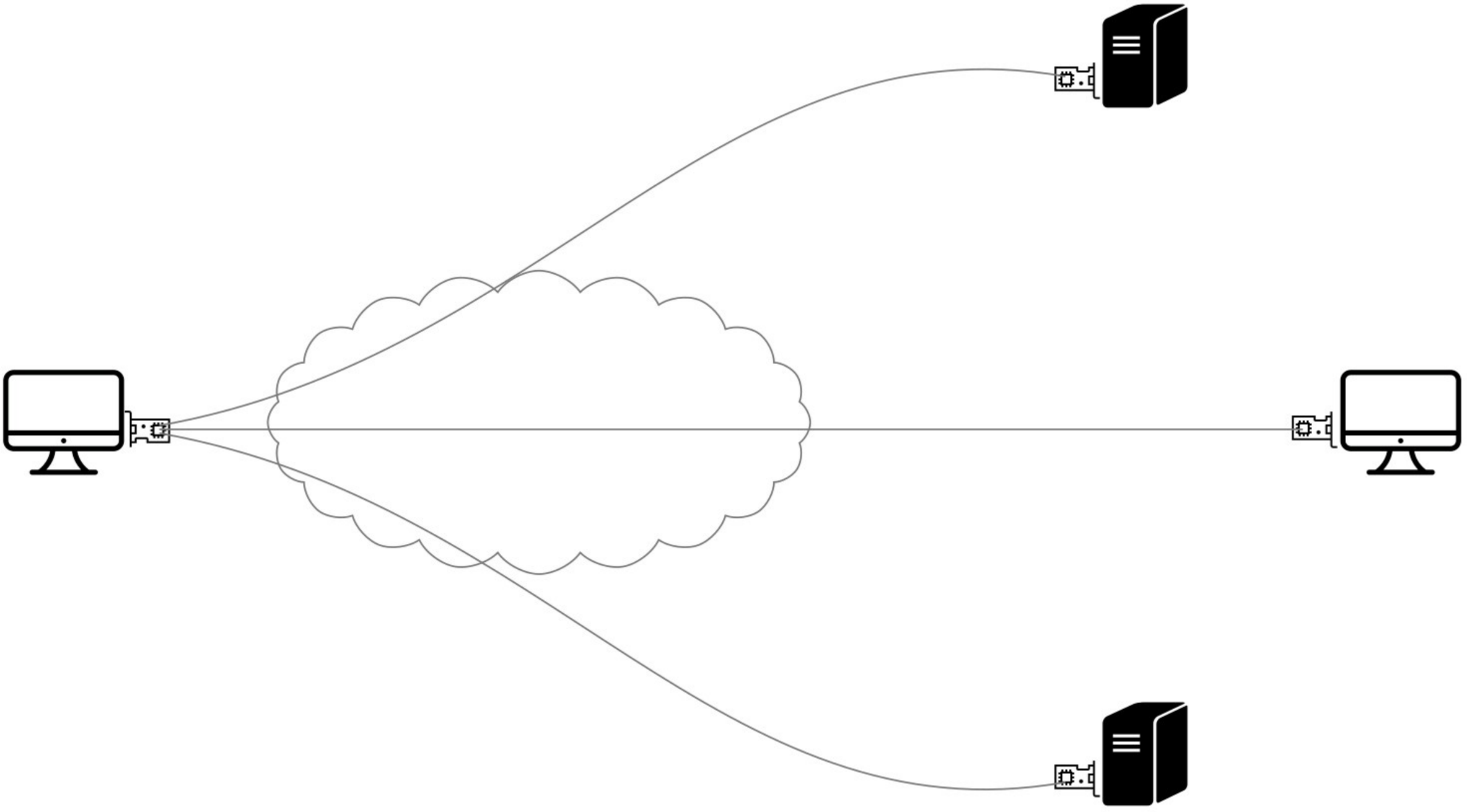
- Context - VPN
- Simple protocol
- Key agreement - `IKpsk2` in action
- Complete protocol
- Current analyses
- New model

Context - VPN

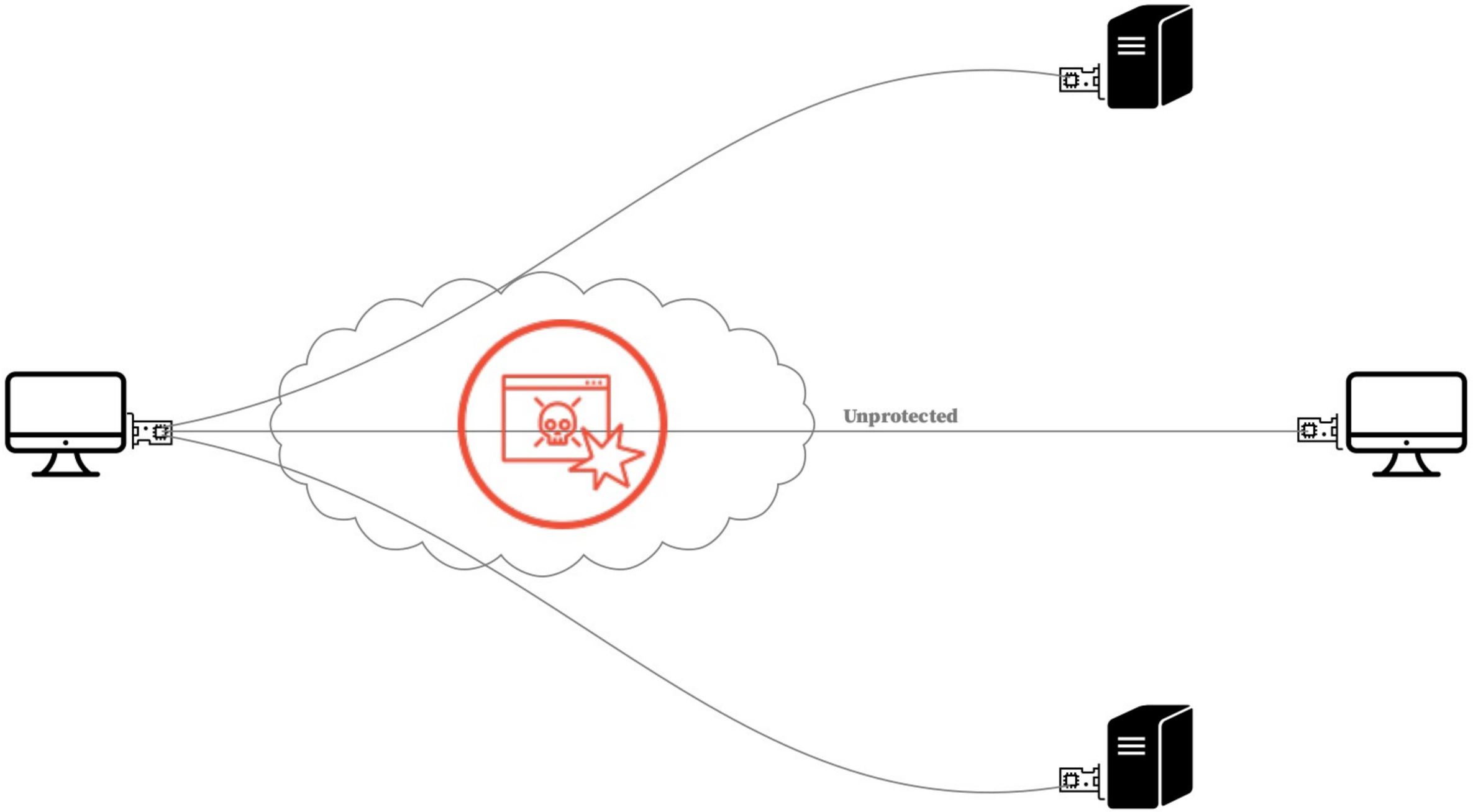




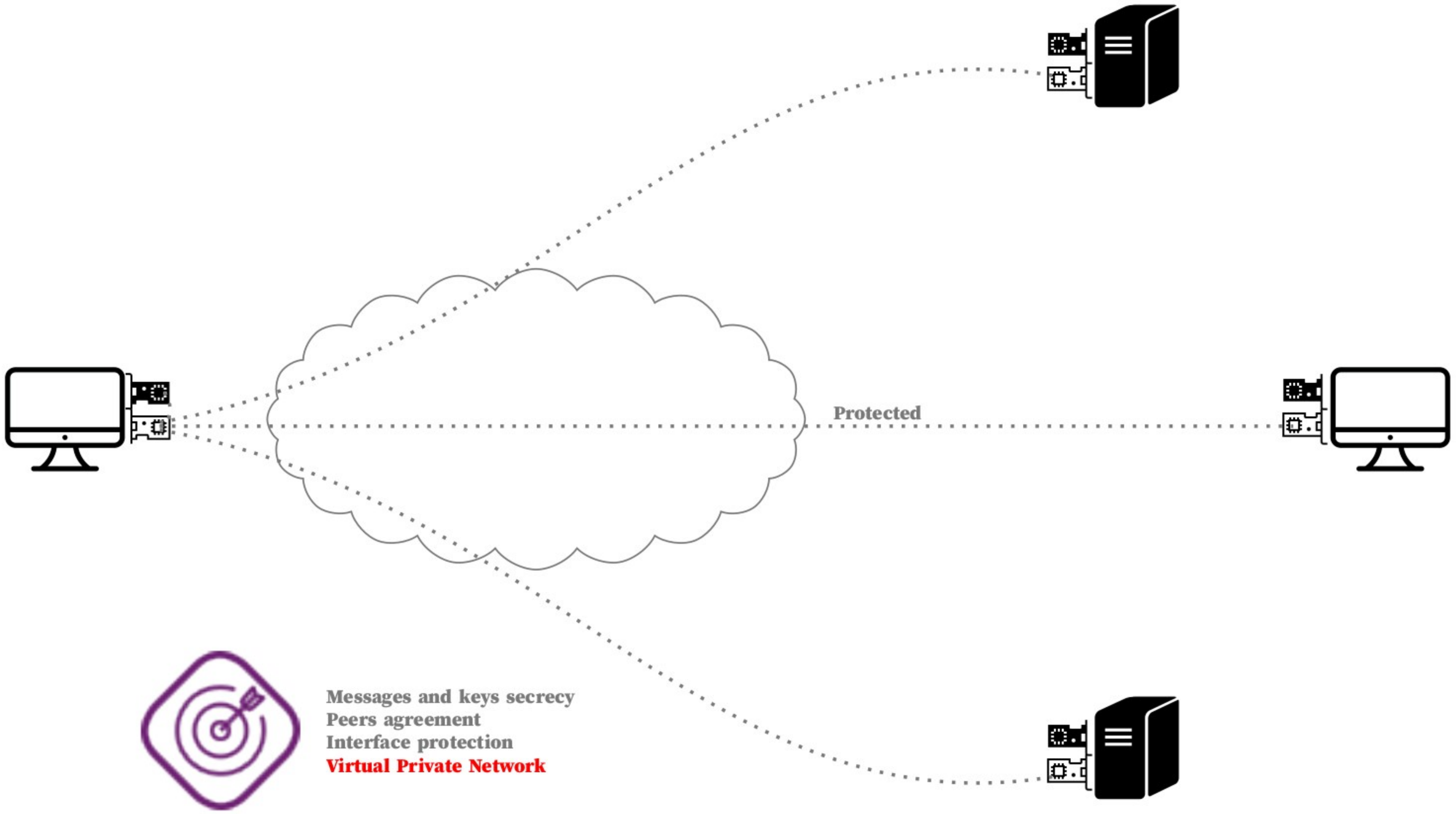
Context - VPN



Context - VPN

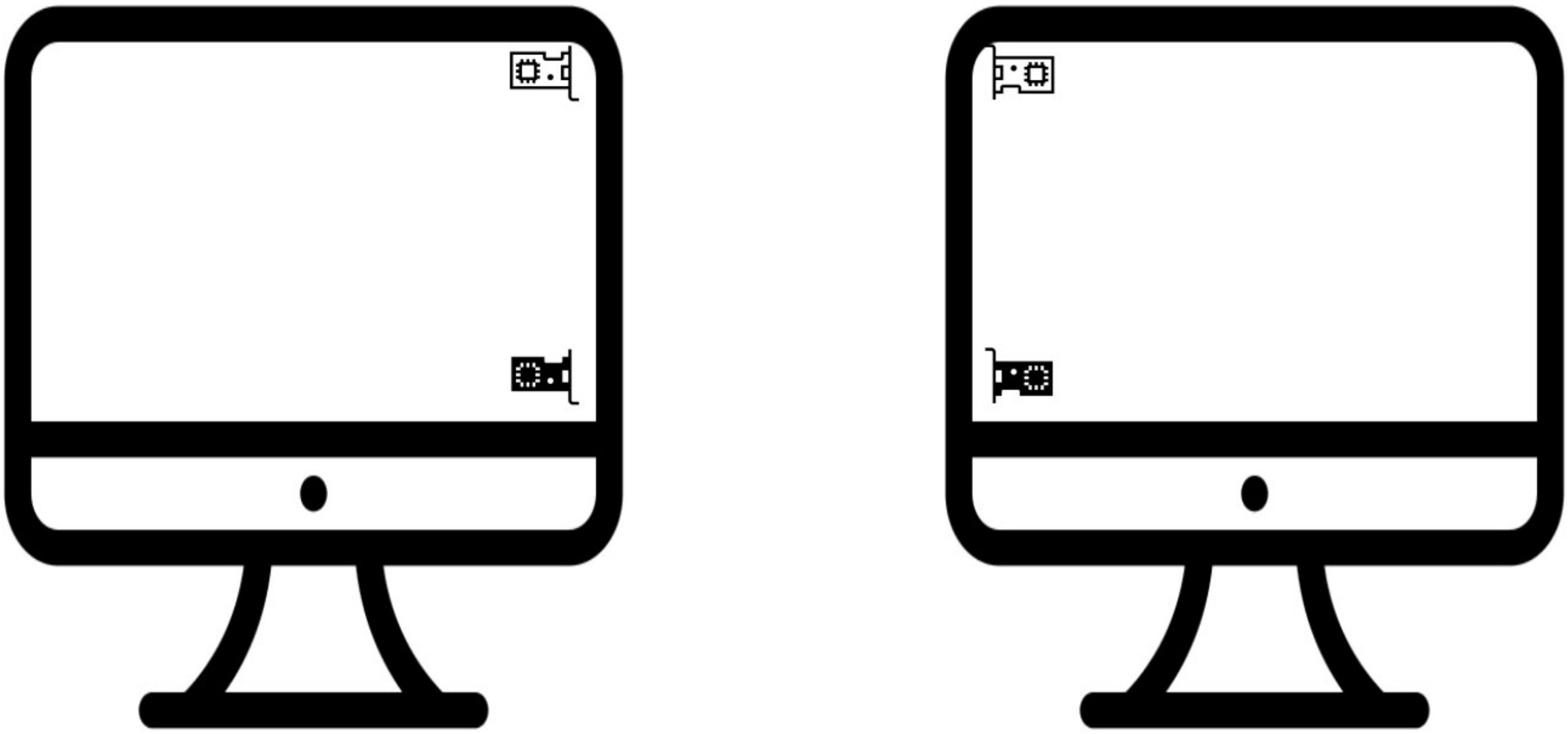


Context - VPN

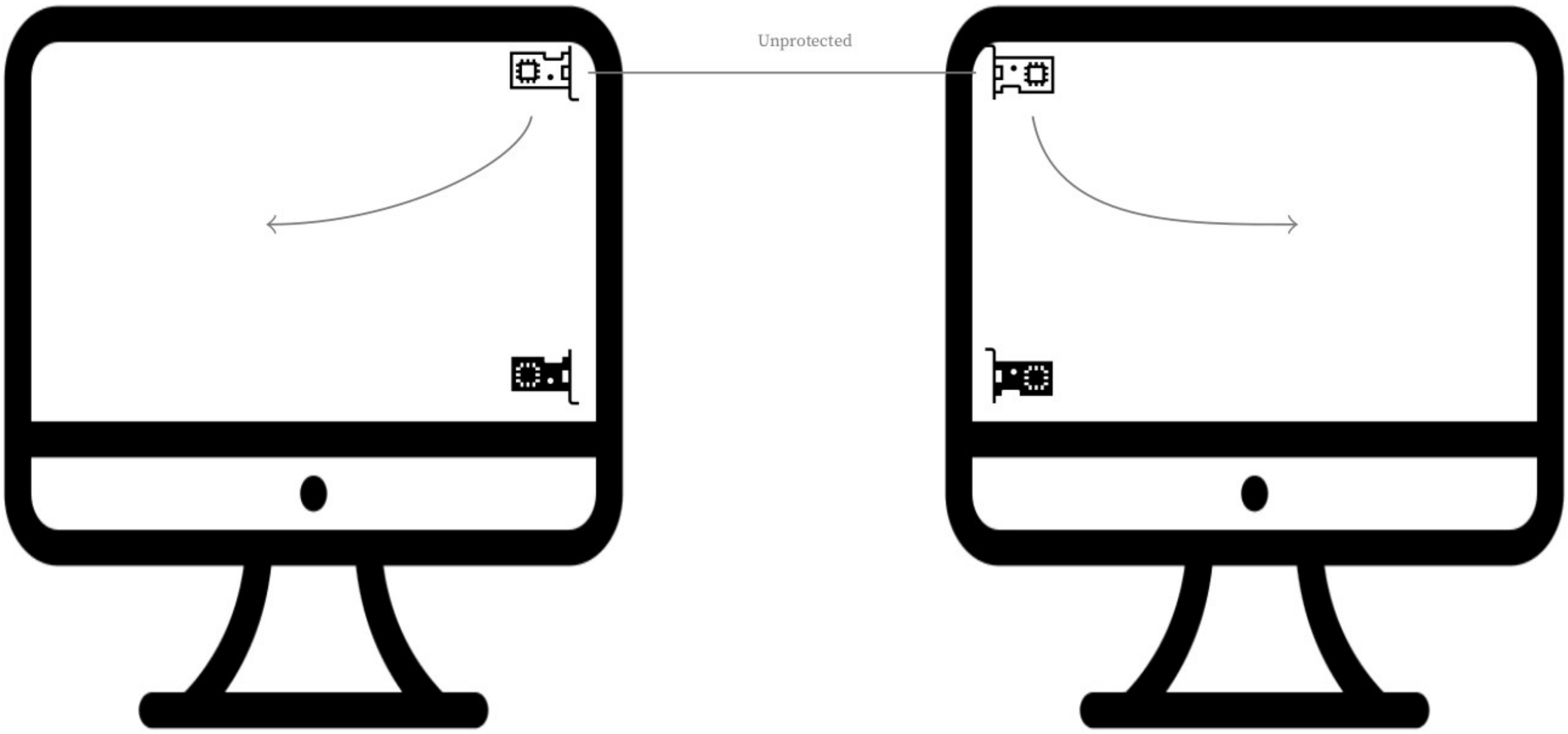


Messages and keys secrecy
Peers agreement
Interface protection
Virtual Private Network

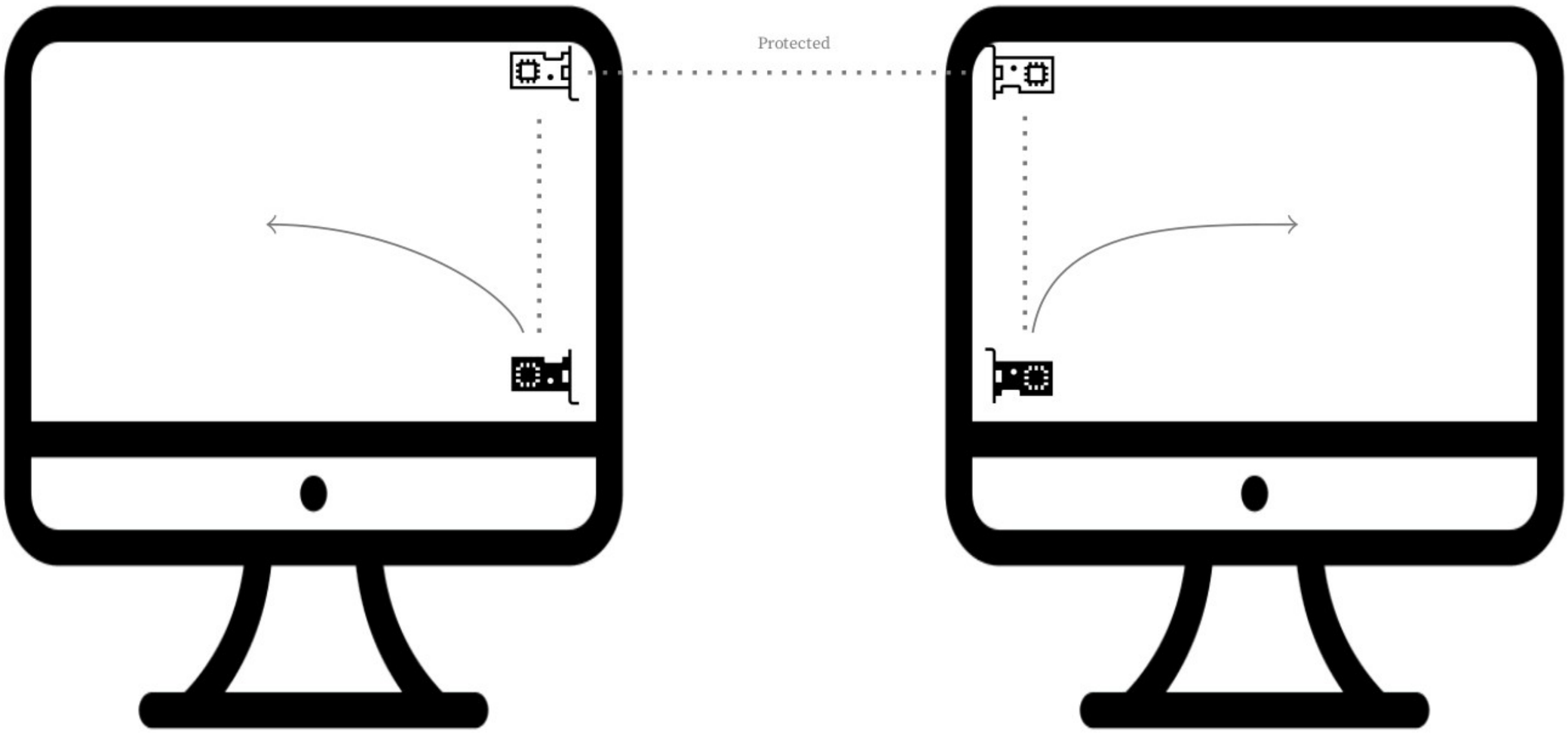
Context - VPN



Context - VPN



Context - VPN



WireGuard, a simple protocol

Main features

- New VPN solution (tunnel mode)
- Integrated in Linux Kernel
- Simple, light
- Open source
- IOS, Android, Windows, Go, ...

WireGuard, a simple protocol

Main features

- New VPN solution (tunnel mode)
- Integrated in Linux Kernel
- Simple, light
- Open source
- IOS, Android, Windows, Go, ...

WireGuard, a simple protocol

Main features

- New VPN solution (tunnel mode)
- Integrated in Linux Kernel
- Simple, light
- Open source
- IOS, Android, Windows, Go, ...

```
InitHello   ⟶
            ⟵   RecHello
TransData   ⟷   TransData
```

WireGuard, a simple protocol

Main features

- New VPN solution (tunnel mode)
- Integrated in Linux Kernel
- Simple, light
- Open source
- IOS, Android, Windows, Go, ...

No crypto-agility

- Curve25519
- `AEAD_CHACHA20_POLY1305`
- Blake2s
- KDF_1, KDF_2, KDF_3
- `IKpsk2`

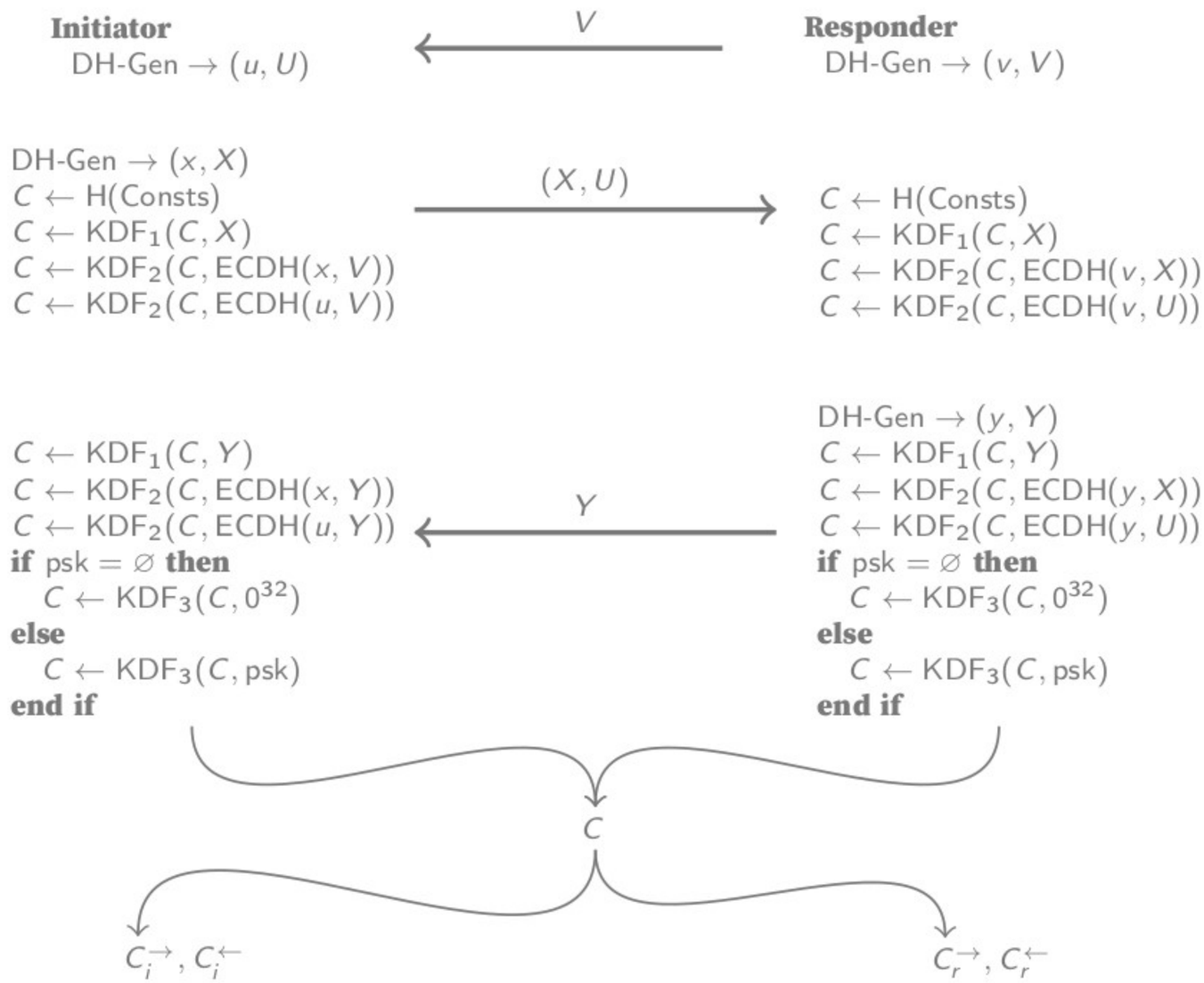
$$\texttt{InitHello} \longrightarrow$$
$$\longleftarrow \quad \texttt{RecHello}$$
$$\texttt{TransData} \longleftrightarrow \texttt{TransData}$$

Key agreement - IKpsk2 in action

```
IKpsk2
  <- s
  ...
  -> e, es, s, ss
  <- e, ee, se, psk
  <-
  ->
```

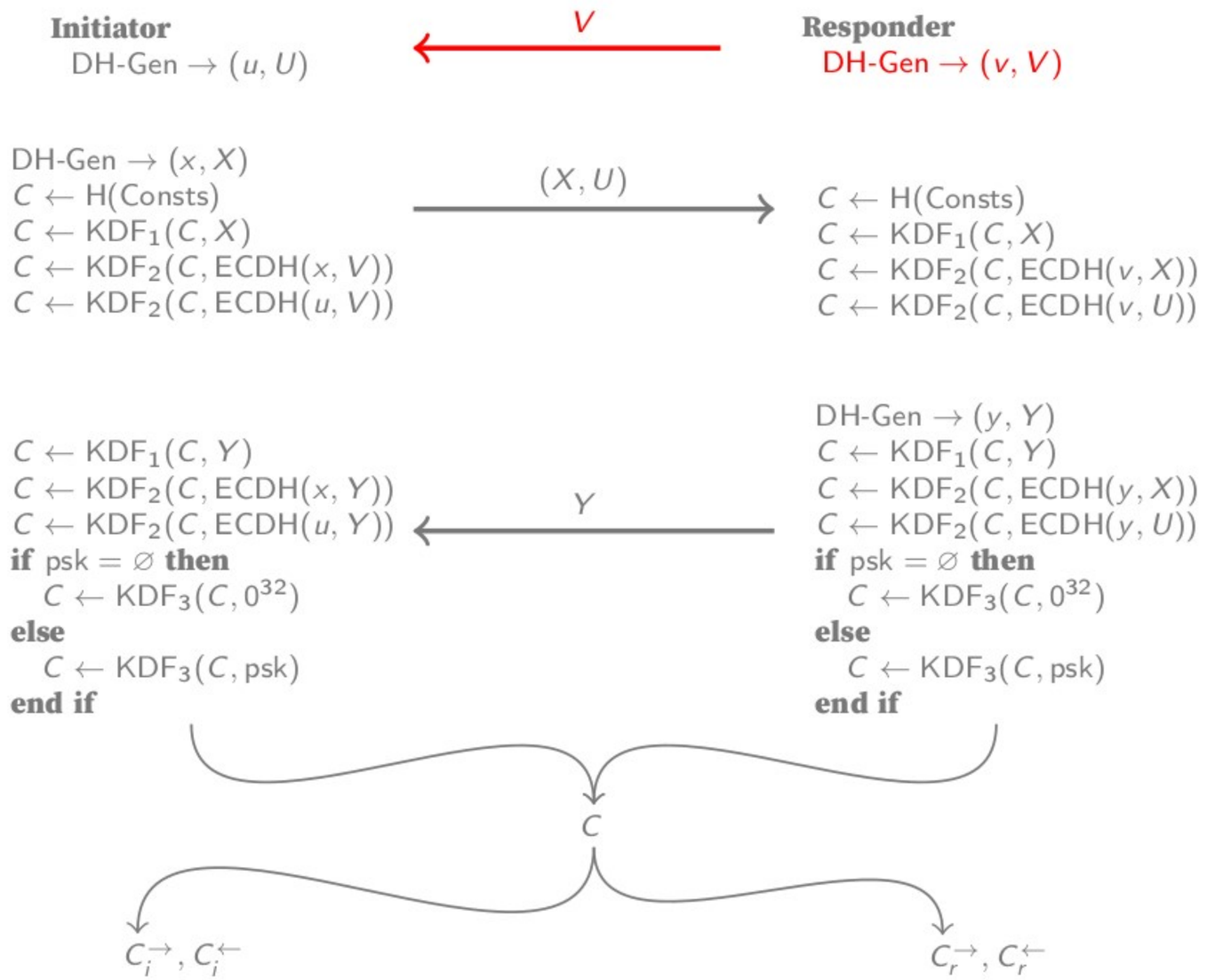
Key agreement - `IKpsk2` in action

```
IKpsk2
  <- s
  ...
  -> e, es, s, ss
  <- e, ee, se, psk
  <-
  ->
```



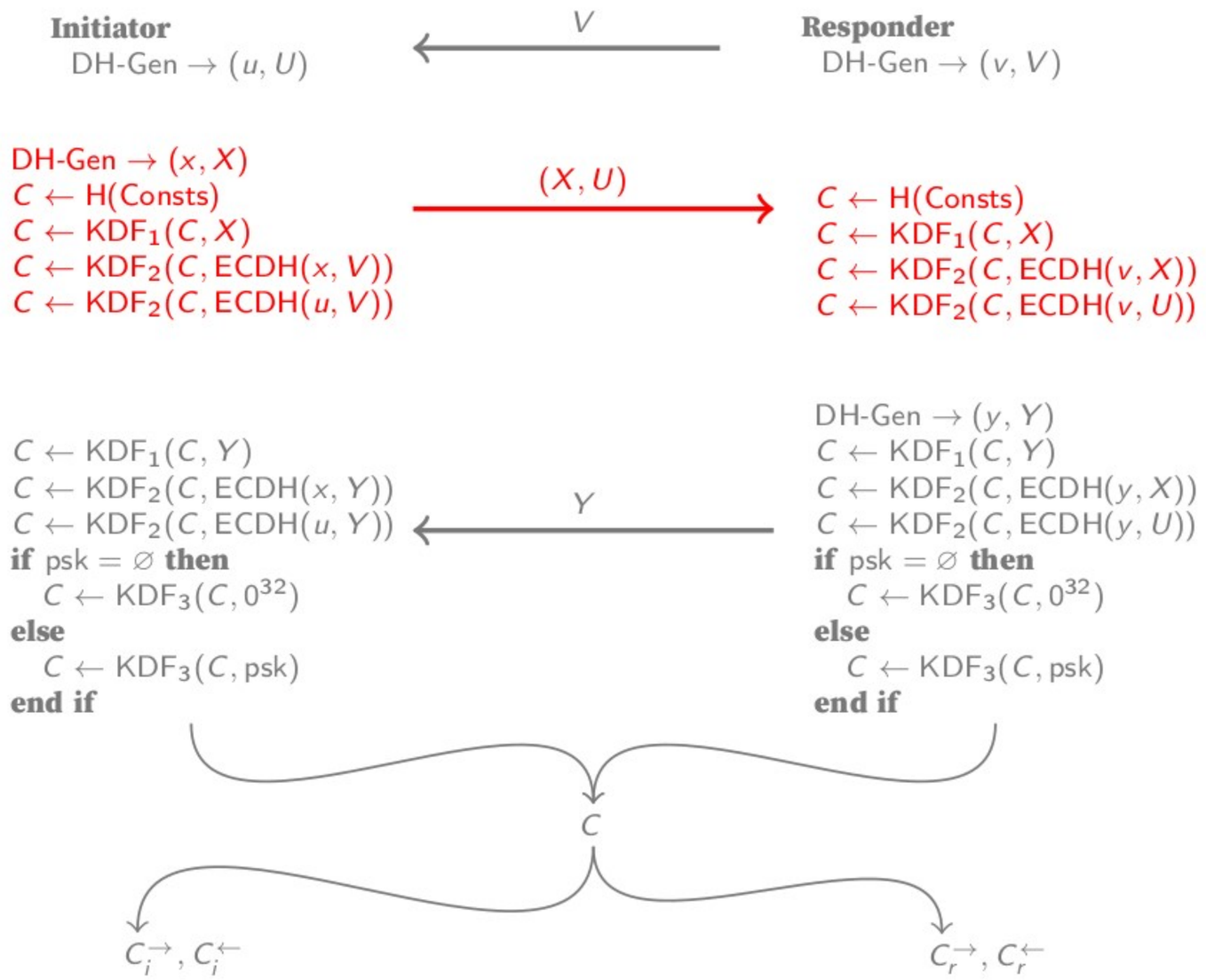
Key agreement - `IKpsk2` in action

```
IKpsk2
  <- s
  ...
  -> e, es, s, ss
  <- e, ee, se, psk
  <-
  ->
```



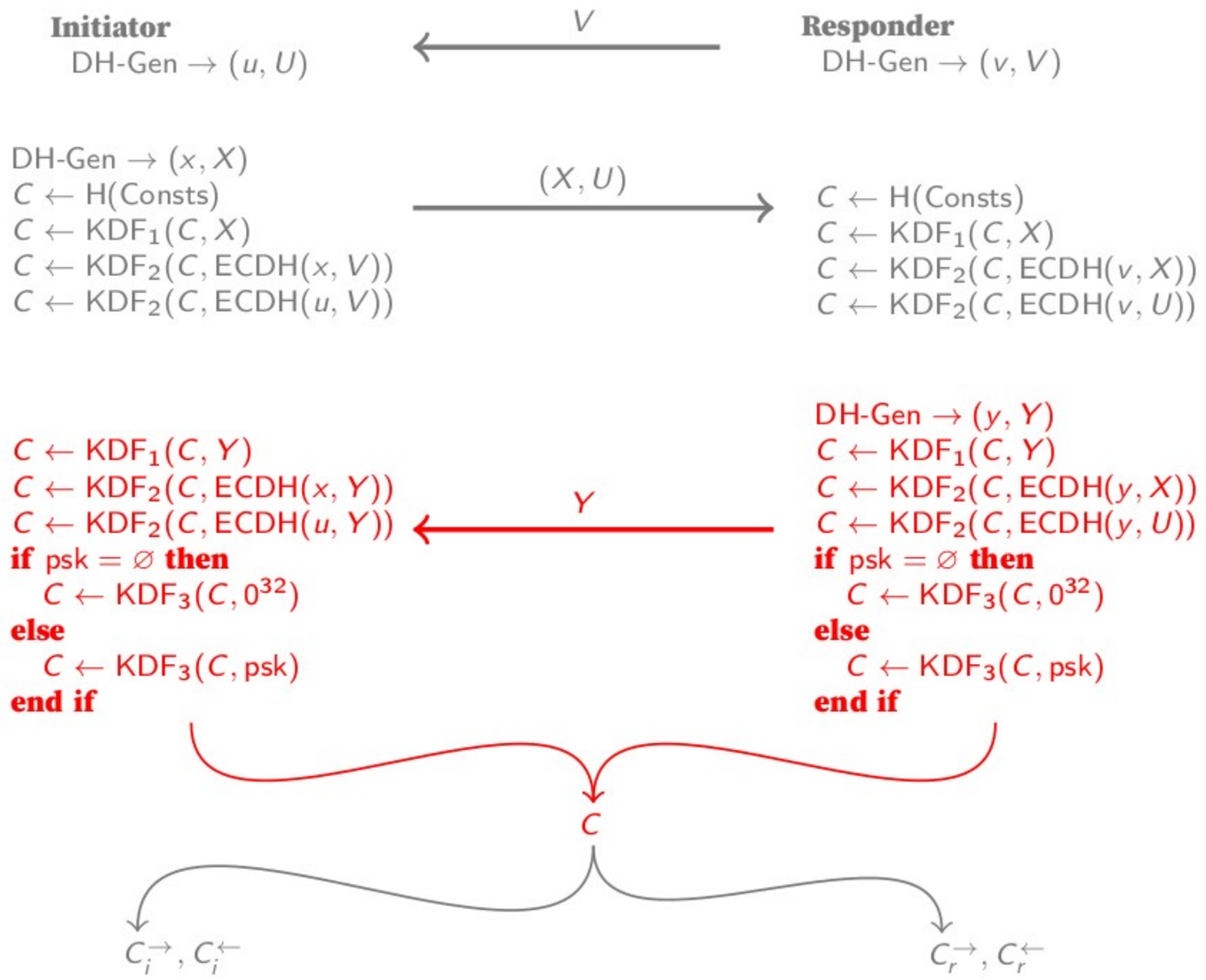
Key agreement - `IKpsk2` in action

```
IKpsk2
  <- s
  ...
  -> e, es, s, ss
  <- e, ee, se, psk
  <-
  ->
```



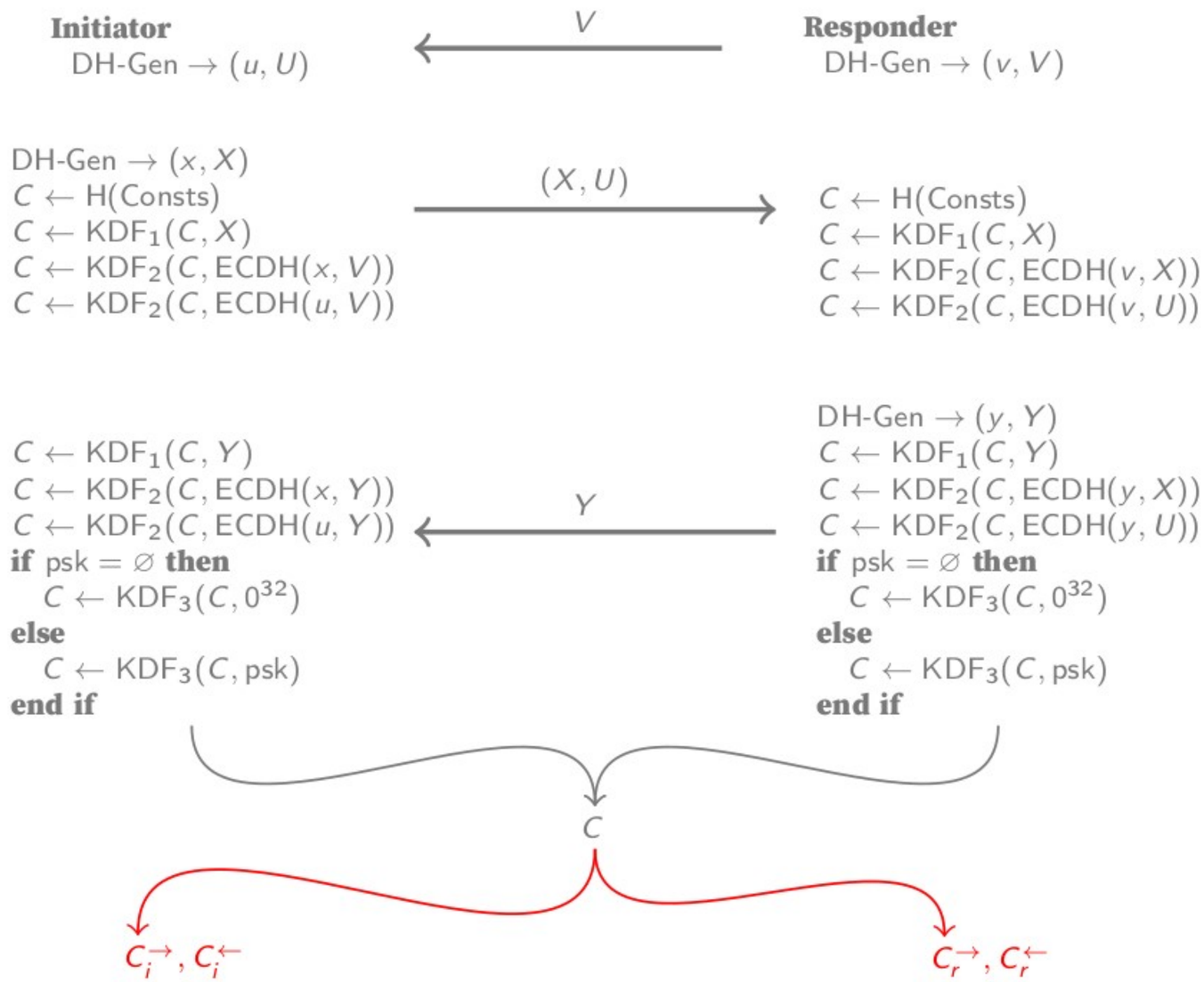
Key agreement - `IKpsk2` in action

```
IKpsk2
  <- s
  ...
  -> e, es, s, ss
  <- e, ee, se, psk
  <-
  ->
```



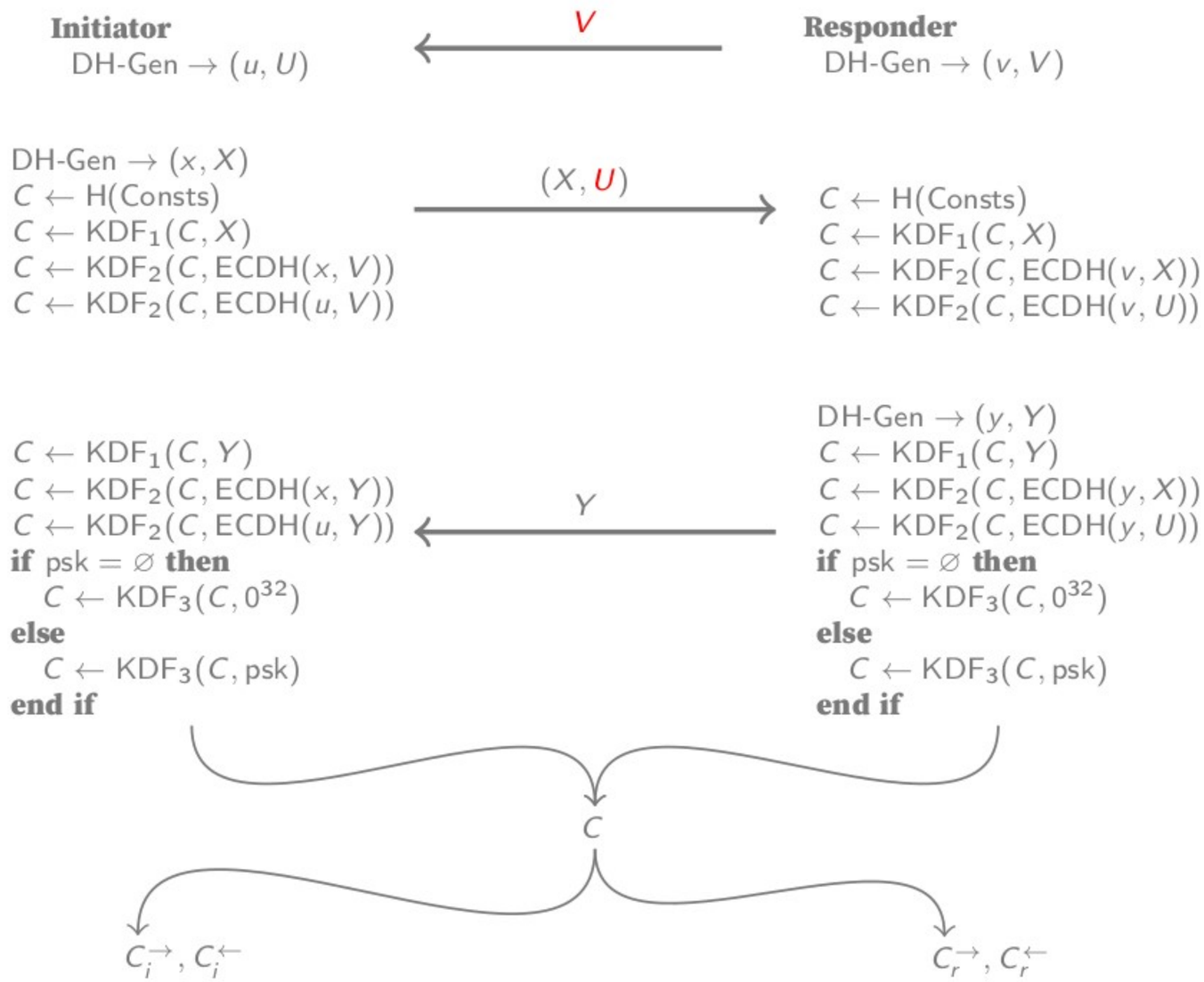
Key agreement - `IKpsk2` in action

```
IKpsk2
  <- s
  ...
  -> e, es, s, ss
  <- e, ee, se, psk
  <-
  ->
```



Key agreement - `IKpsk2` in action

```
IKpsk2
  <- s
  ...
  -> e, es, s, ss
  <- e, ee, se, psk
  <-
  ->
```

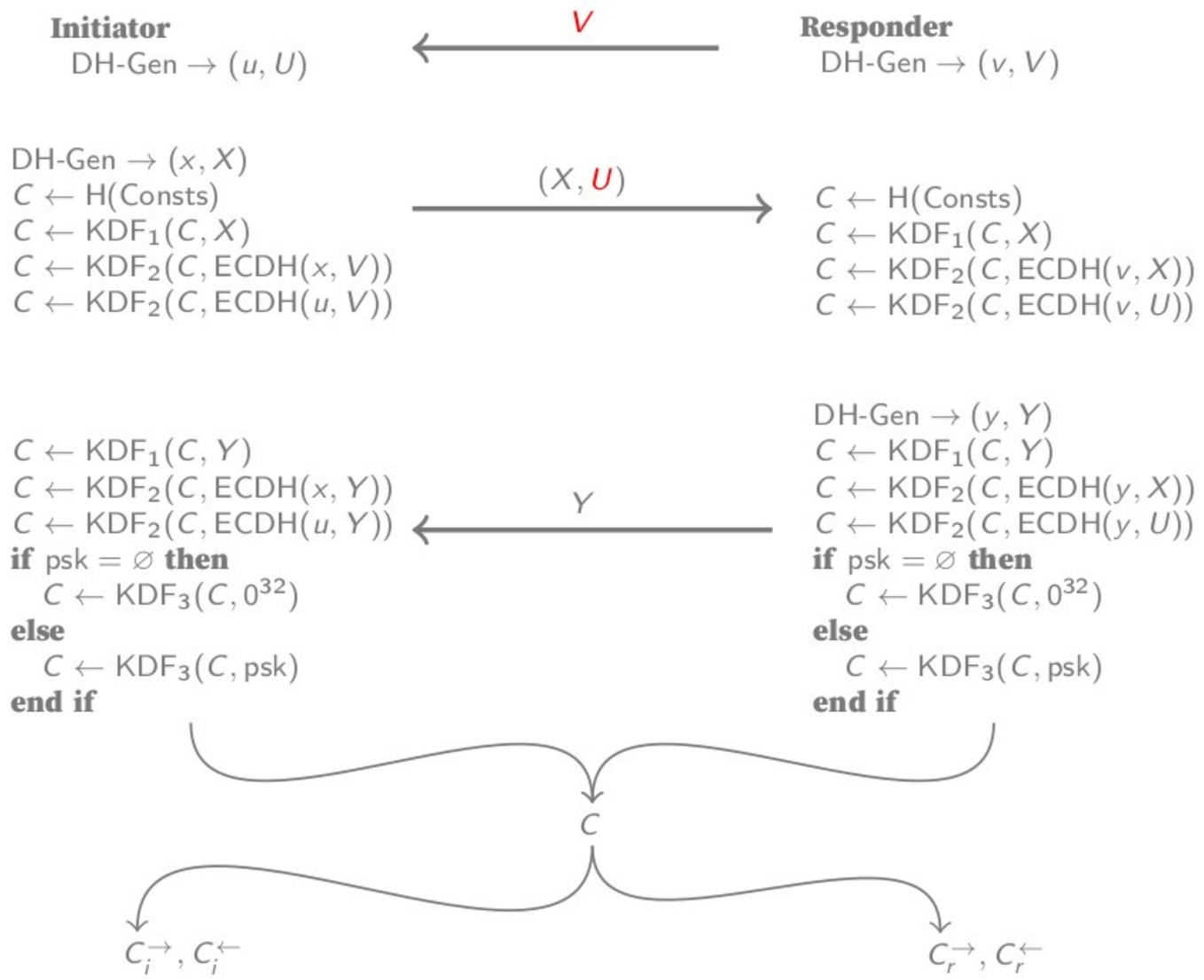


Static keys validation

- Noise specification: *it's up to the application to determine whether the remote party's static public key is acceptable.*

Key agreement - `IKpsk2` in action

```
IKpsk2
  <- s
  ...
  -> e, es, s, ss
  <- e, ee, se, psk
  <-
  ->
```



Static keys validation

- Noise specification: *it's up to the application to determine whether the remote party's static public key is acceptable.*
- WireGuard specification: *WireGuard rests upon peers exchanging static public keys with each other.*

$\Longrightarrow$ **Not done in WireGuard ...**

From simple to complete protocol

At first specified WireGuard protocol is simple ...

```
InitHello  ⟶
           ⟵   RecHello
TransData  ⟷   TransData
```

From simple to complete protocol

... initial static key exchange needs to be taken into account ...

From simple to complete protocol

... key exchange is considered complete *after* first transport message ...

From simple to complete protocol

... cookie messages are used to protect against DOS attacks.

Current analyses

Specifications

- 2017: J.A. Donenfeld, "WireGuard: Next generation kernel network tunnel"
- 2018: T. Perrin, "The noise protocol framework"

Two types of proofs

- Computationnal:
  - Manual
  - Computer-aided (CryptoVerif)
- Symbolic, computer-aided:
  - ProVerif
  - Tamarin

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*

Current analyses

What is the scope of *WireGuard* analyses ?

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

Are `IKpsk2` analyses applicable to WireGuard ?

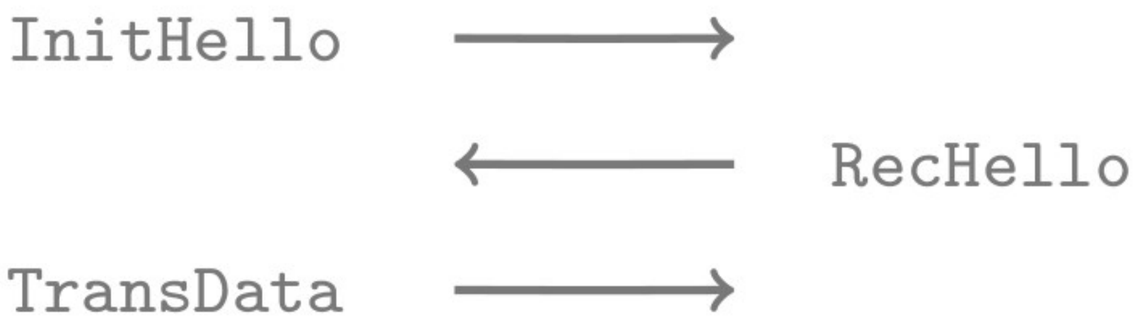
- Lazy answer: yes !
- Correct answer: should be studied !

Are threat model equivalent ?

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

Current analyses

What is the scope of *WireGuard* analyses ?



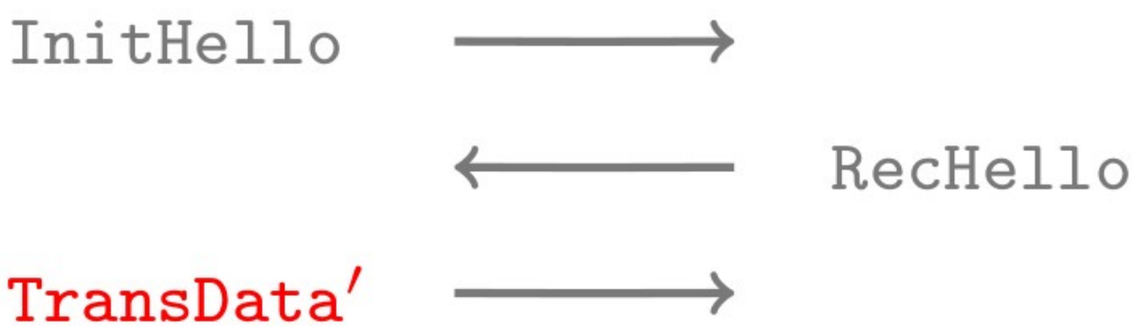
Symbolic *WireGuard*

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

- "Simple" version of the protocol (with only confirmation message) ✗
- Initial key distribution not assessed ✗
- No Cookie messages ✗

Current analyses

What is the scope of *WireGuard* analyses ?



Computationnal *WireGuard*

- 2018: B. Dowling and K. G. Paterson, “A cryptographic analysis of the WireGuard protocol”

- "Simple" version of the protocol (with *modified* confirmation message) ✗
- Initial key distribution not assessed ✗
- No Cookie messages ✗

Current analyses

What is the scope of *WireGuard* analyses ?

Computationnal *WireGuard*

- 2019: B. Lipp, B. Blanchet, and K. Bhargavan, “A mechanised cryptographic proof of the WireGuard VPN protocol”

- "Complete" protocol (with confirmation message and transport messages) ✓
- Initial key distribution ✓
- Cookie messages not included in full protocol, but in a separate model ✗

Current analyses

Are `IKpsk2` analyses applicable to WireGuard ?

InitHello
type(1)
reserved
sender
ephemeral
static
timestamp
mac1
mac2

RecHello
type(2)
reserved
sender
receiver
ephemeral
empty
mac1
mac2

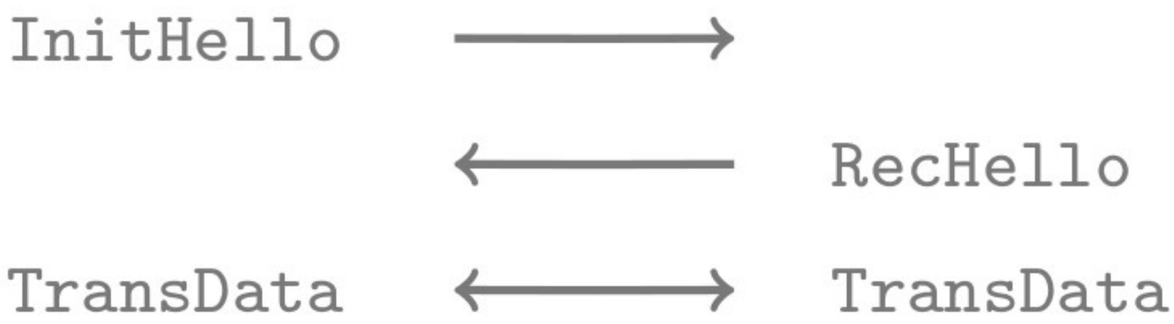
TransData
type(3)
reserved
receiver
counter
packet

Cookie
type(4)
reserved
receiver
nonce
cookie

- Fields `static`, `timestamp`, `empty` are specific to WireGuard
- WireGuard includes
  - session identifiers `sender`, `receiver`
  - MAC computations `mac1`, `mac2`
  - `Cookie` messages

Current analyses

Are `IKpsk2` analyses applicable to WireGuard ?



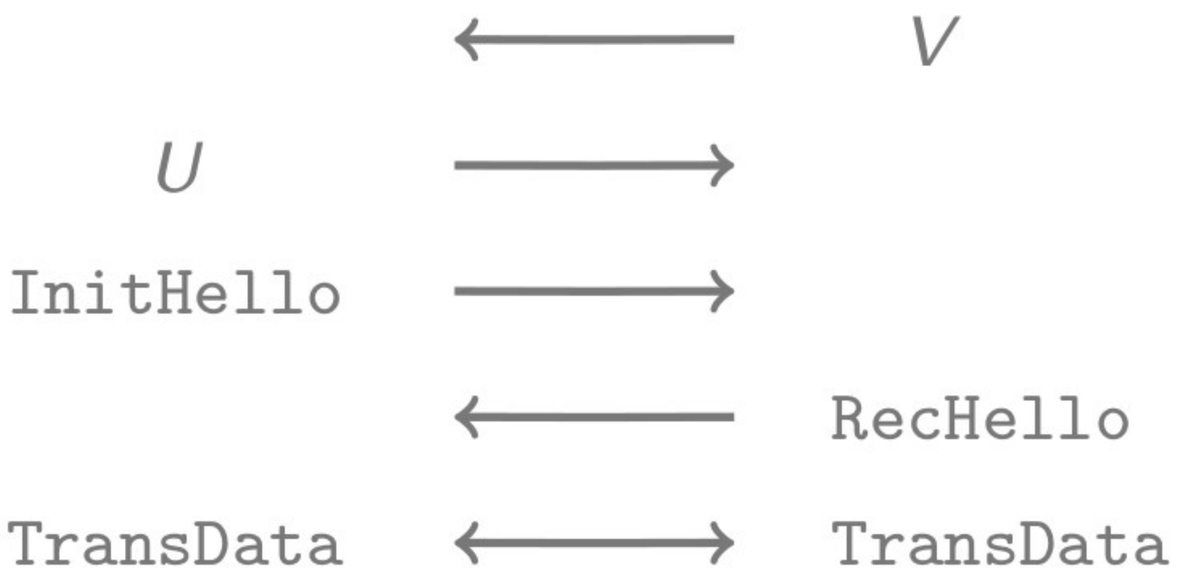
Symbolic *`IKpsk2`*

- 2019: N. Kobeissi, G. Nicolas, and K. Bhargavan, “Noise explorer: Fully automated modeling and verification for arbitrary noise protocols”

- "Simple" version of the protocol ✗
- No confirmation message ✗
- Initial key distribution not assessed ✗

Current analyses

Are `IKpsk2` analyses applicable to WireGuard ?



Symbolic *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”

- "Simple" version of the protocol ✗
- No confirmation message ✗
- Initial key distribution assessed ✓

Current analyses

Threat models also differ !

Threats

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

Current analyses

Threat models also differ !

Threats

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

Symbolic *WireGuard*

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

Proof

- Computer-aided (TAMARIN)

Current analyses

Threat models also differ !

Threats

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

Computationnal *WireGuard*

- 2018: B. Dowling and K. G. Paterson, “A cryptographic analysis of the WireGuard protocol”

Proof

- Manual

Current analyses

Threat models also differ !

Threats

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

Computationnal *WireGuard*

- 2019: B. Lipp, B. Blanchet, and K. Bhargavan, "A mechanised cryptographic proof of the WireGuard VPN protocol"

Proof

- Computer-aided (CryptoVerif)

Current analyses

Threat models also differ !

Threats

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

Symbolic *IKpsk2*

- 2019: N. Kobeissi, G. Nicolas, and K. Bhargavan, “Noise explorer: Fully automated modeling and verification for arbitrary noise protocols”

Proof

- Computer-aided (ProVerif)

Current analyses

Threat models also differ !

Threats

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

Symbolic *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"

Proof

- Computer-aided (TAMARIN)

New model

Objective

- Be as complete as possible !
  - Key distribution
  - Confirmation message
  - Include `Cookie` message in full protocol
- Focus on Symbolic analysis.
- Should be verifiable with TAMARIN and PROVERIF.

New model

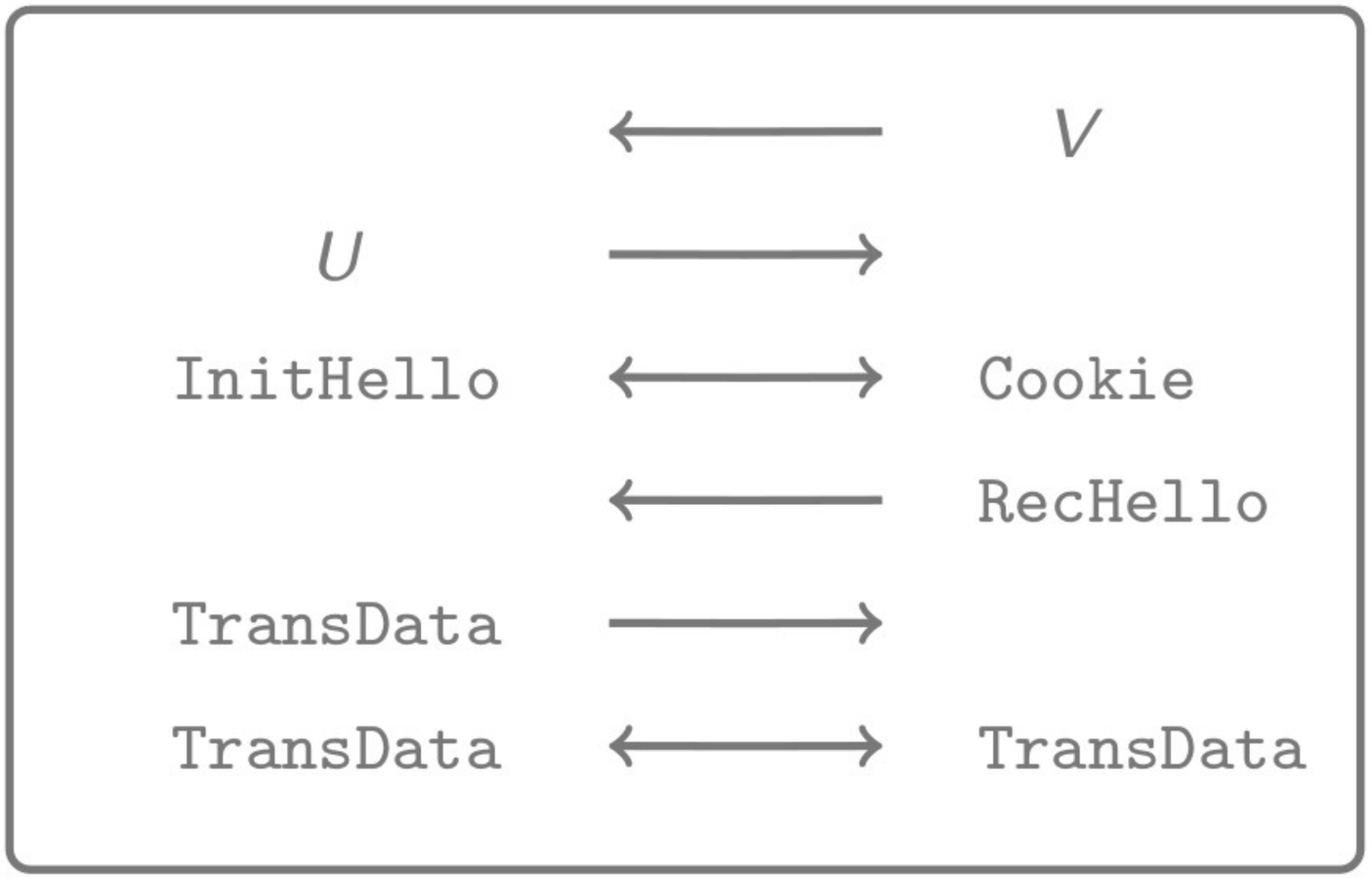
Objective

- Be as complete as possible !
  - Key distribution
  - Confirmation message
  - Include `Cookie` message in full protocol
- Focus on Symbolic analysis.
- Should be verifiable with TAMARIN and PROVERIF.

New proof assistant: SAPIC$^+$

- 2022: V. Cheval, C. Jacomme, S. Kremer, and R. Künnemann, "SAPIC$^+$: protocol verifiers of the world, unite!"

Target model



- "Complete" protocol (with confirmation message and transport messages) ✓
- Initial key distribution ✓
- Cookie messages included in full protocol ✓

Target threat model

Threats

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

Precomputation ?

- WireGuard implements static key ECDH precomputation.
- That is
  - Initiator implements $\mathsf{ECDH}(u, V)$
  - Responder implements $\mathsf{ECDH}(v, U)$

  before session begins, hence WireGuard maintains it.

Compromise of ECDH precomputation is *weaker* than compromise of private static keys:

- if adv. has v and g^u, adv. has g^{uv},
- however if adv. has g^u and g^{uv}, adv. *does not* have v.

Methodology

- Start with a reference model with all "Gets" threats in SAPIC$^+$
- Derivate PROVERIF models for "Sets" threats
- Verify queries on all derivated models in PROVERIF

How many models ? How many queries ?

- $2^9 = 512$ models, as adversary
  - Can set u, v, psk, x, y, g^{uv} (for Initiator and Responder)
  - Can corrupt U and V distribution
  - And can combine !
- Up to $2^6 = 64$ queries per model, as adversary
  - Can get u, v, psk, x, y, g^{uv}
  - And can combine !
- Can be simplified (e.g no need to test adversary that gets and sets u.)

$\Longrightarrow$ **4860** queries for each security property (instead of $2^{15} = 32768$) !

Illustration - reference model

```
process:

new ~ltkI;
new ~ltkR;
new ~psk;

new empty;

out(<'initiator', 'g'^~ltkI>);
out(<'responder', 'g'^~ltkR>);
out(empty);

!(
    Initiator(~ltkI, 'g'^~ltkI, 'g'^~ltkR, ('g'^~ltkR)^~ltkI, ~psk, empty, 'zero')
  | Responder(~ltkR, 'g'^~ltkI, 'g'^~ltkR, ('g'^~ltkI)^~ltkR, ~psk, empty, 'zero')
  | RevealPsk(~psk)
  | RevealPri(~ltkI)
  | RevealPri(~ltkR)
  | RevealPre(~ltkI, ~ltkR)

)
```

```
let Initiator(~ltkI, pkI, pkR, sisr, ~psk, empty, zero_1) =
    ...
    new ~ekI;
    ...
    let pekI  = 'g'^~ekI in
    (
    ...
    )
    | (RevealPri(~ekI))
```

```
let Responder(~ltkR, pkI, pkR, srsi, ~psk, empty, zero_1) =
    ...
    new ~ekR;
    ...
    let pekR  = 'g'^~ekR in
    (
    ...
    )
    | (RevealPri(~ekR))
```

Illustration - model derivation (adversary sets psk and x and y)

```
process:

new ~ltkI;
new ~ltkR;
in(~psk);

new empty;

out(<'initiator', 'g'^~ltkI>);
out(<'responder', 'g'^~ltkR>);
out(empty);

!(
    Initiator(~ltkI, 'g'^~ltkI, 'g'^~ltkR, ('g'^~ltkR)^~ltkI, ~psk, empty, 'zero')
  | Responder(~ltkR, 'g'^~ltkI, 'g'^~ltkR, ('g'^~ltkI)^~ltkR, ~psk, empty, 'zero')
  // | RevealPsk(~psk)
  | RevealPri(~ltkI)
  | RevealPri(~ltkR)
  | RevealPre(~ltkI, ~ltkR)

)
```

```
let Initiator(~ltkI, pkI, pkR, sisr, ~psk, empty, zero_1) =
    ...
    in(~ekI);
    ...
    let pekI  = 'g'^~ekI in
    (
    ...
    )
    // | (RevealPri(~ekI))
```

```
let Responder(~ltkR, pkI, pkR, srsi, ~psk, empty, zero_1) =
    ...
    in(~ekR);
    ...
    let pekR  = 'g'^~ekR in
    (
    ...
    )
    // | (RevealPri(~ekR))
```

Methodology

- Verify queries on all derivated models in ProVerif
- Extract all "true" properties
- Compute a combination of all "true" properties

Combination of all "true" properties ?

- Compute conjunction.
- Compute DNF.

$\Longrightarrow$ compact result !

Inspired by:

Symbolic *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"

Illustration with agreement on 3^{rd} message

Verify all 4860 queries ...

```
query i:time,j:time, pki:bitstring, pkr:bitstring, peki:bitstring, pekr:bitstring, psk:bitstring, ck:bitstring;
(event(eRConfirm(pki, pkr, peki, pekr, psk, ck))@i)
==> (((event(eIConfirm(pki, pkr, peki, pekr, psk, ck))@j) && (j < i))).
```

...

```
query i:time,j:time, pki:bitstring, pkr:bitstring, peki:bitstring, pekr:bitstring, psk:bitstring, ck:bitstring,
j1:time, j2:time, j3:time, j4:time, j5:time, j6:time;
(event(eRConfirm(pki, pkr, peki, pekr, psk, ck))@i)
==> (((event(eIConfirm(pki, pkr, peki, pekr, psk, ck))@j) && (j < i))
||((event(eRevPri(pki))@j1) && (j1 < i))
||((event(eRevPri(pkr))@j2) && (j2 < i))
||((event(eRevPri(peki))@j3) && (j3 < i))
||((event(eRevPri(pekr))@j4) && (j4 < i))
||((event(eRevPre(pki, pkr))@j5) && (j5 < i))
||((event(eRevPsk(psk))@j6) && (j6 < i))).
```

- **1824** queries are "true"
- **3036** queries are "false"
- **0** queries are "cannot be proved"

Illustration with agreement on 3^{rd} message

Extract and compute DNF requires the use of symbols:

- `Du`, `Dv`: adversary corrupts public keys distribution
- `Ru`, `Rv`, `Rx`, `Ry`, `Rs`, `Rc`: adversary gets private keys (`u`, `v`, `x`, `y`), psk (`s`) or precomp. value (`c`)
- `Mu`, `Mv`, `Mx`, `My`, `Ms`, `Mi`, `Mr`: adversary sets private keys (`u`, `v`, `x`, `y`), psk (`s`) or precomp value (`i` for Initiator, `r` for Responder)

```
DNF = (Du & Rs) | (Ms & Ru) | (Mu & Rs) | (Rs & Ru) | (Mi & Ms & Ry) | (Mi & My & Rs) | (Mi & Rs & Ry) |
(Mr & Ms & Ry) | (Mr & My & Rs) | (Mr & Rs & Ry) | (Ms & Mv & Ry) | (Ms & My & Rc) | (Ms & My & Rv) |
(Ms & Rc & Ry) | (Ms & Rv & Ry) | (Mv & My & Rs) | (Mv & Rs & Ry) | (My & Rc & Rs) | (My & Rs & Rv) |
(Rc & Rs & Ry) | (Rs & Rv & Ry)
```

Interpretation

We have agreement on 3^{rd} message ***unless*** adversary

- compromises initiator's static key distribution **AND** gets psk
- **OR** sets psk **AND** gets initiator's static private key *u*
- **OR** ...
- **OR** gets psk **AND** gets responder 's static private key *v* **AND** gets responder 's ephemeral private key *y*

Illustration with agreement on 3^{rd} message

Extract, compute DNF. Requires the use of symbols:

- `Du`, `Dv`: adversary corrupts public keys distribution
- `Ru`, `Rv`, `Rx`, `Ry`, `Rs`, `Rc`: adversary gets private keys (`u`, `v`, `x`, `y`), psk (`s`) or precomp. value (`c`)
- `Mu`, `Mv`, `Mx`, `My`, `Ms`, `Mi`, `Mr`: adversary sets private keys (`u`, `v`, `x`, `y`), psk (`s`) or precomp value (`i` for Initiator, `r` for Responder)

```
DNF = (Du & Rs) | (Ms & Ru) | (Mu & Rs) | (Rs & Ru) | (Mi & Ms & Ry) | (Mi & My & Rs) | (Mi & Rs & Ry) |
(Mr & Ms & Ry) | (Mr & My & Rs) | (Mr & Rs & Ry) | (Ms & Mv & Ry) | (Ms & My & Rc) | (Ms & My & Rv) |
(Ms & Rc & Ry) | (Ms & Rv & Ry) | (Mv & My & Rs) | (Mv & Rs & Ry) | (My & Rc & Rs) | (My & Rs & Rv) |
(Rc & Rs & Ry) | (Rs & Rv & Ry)
```

Key distribution corruption

We have agreement on 3^{rd} message ***unless*** adversary:

- compromises initiator's static key distribution **AND** gets psk

$\Longrightarrow$ **Key distribution has an impact on security !**

Illustration with agreement on 3^{rd} message

Extract, compute DNF. Requires the use of symbols:

- `Du`, `Dv`: adversary corrupts public keys distribution
- `Ru`, `Rv`, `Rx`, `Ry`, `Rs`, `Rc`: adversary gets private keys (`u`, `v`, `x`, `y`), psk (`s`) or precomp. value (`c`)
- `Mu`, `Mv`, `Mx`, `My`, `Ms`, `Mi`, `Mr`: adversary sets private keys (`u`, `v`, `x`, `y`), psk (`s`) or precomp value (`i` for Initiator, `r` for Responder)

```
DNF = (Du & Rs) | (Ms & Ru) | (Mu & Rs) | (Rs & Ru) | (Mi & Ms & Ry) | (Mi & My & Rs) | (Mi & Rs & Ry) |
(Mr & Ms & Ry) | (Mr & My & Rs) | (Mr & Rs & Ry) | (Ms & Mv & Ry) | (Ms & My & Rc) | (Ms & My & Rv) |
(Ms & Rc & Ry) | (Ms & Rv & Ry) | (Mv & My & Rs) | (Mv & Rs & Ry) | (My & Rc & Rs) | (My & Rs & Rv) |
(Rc & Rs & Ry) | (Rs & Rv & Ry)
```

Precomputation

We have agreement on 3^{rd} message ***unless*** adversary:

- sets initiator's precomputation **AND** sets psk **AND** gets responder's ephemeral private key
- **OR** ...
- **OR** gets precomputation value **AND** gets psk **AND** gets responder's ephemeral private key

Illustration with agreement on 3^{rd} message

Extract, compute DNF. Requires the use of symbols:

- Du, Dv: adversary corrupts public keys distribution
- Ru, Rv, Rx, Ry, Rs, Rc: adversary gets private keys (u, v, x, y), psk (s) or precomp. value (c)
- Mu, Mv, Mx, My, Ms, Mi, Mr: adversary sets private keys (u, v, x, y), psk (s) or precomp value (i for Initiator, r for Responder)

```
DNF = (Du & Rs) | (Ms & Ru) | (Mu & Rs) | (Rs & Ru) | (Mi & Ms & Ry) | (Mi & My & Rs) | (Mi & Rs & Ry) |
(Mr & Ms & Ry) | (Mr & My & Rs) | (Mr & Rs & Ry) | (Ms & Mv & Ry) | (Ms & My & Rc) | (Ms & My & Rv) |
(Ms & Rc & Ry) | (Ms & Rv & Ry) | (Mv & My & Rs) | (Mv & Rs & Ry) | (My & Rc & Rs) | (My & Rs & Rv) |
(Rc & Rs & Ry) | (Rs & Rv & Ry)
```

Precomputation

In some cases, Rc has the same impact as Mv or Rv, although Rc is *weaker* than Mv or Rv.
$\Longrightarrow$ **Precomputation has a negative impact on security !**

Illustration with agreement on 3^{rd} message

Extract, compute DNF. Requires the use of symbols:

- `Du`, `Dv`: adversary corrupts public keys distribution
- `Ru`, `Rv`, `Rx`, `Ry`, `Rs`, `Rc`: adversary gets private keys (`u`, `v`, `x`, `y`), psk (`s`) or precomp. value (`c`)
- `Mu`, `Mv`, `Mx`, `My`, `Ms`, `Mi`, `Mr`: adversary sets private keys (`u`, `v`, `x`, `y`), psk (`s`) or precomp value (`i` for Initiator, `r` for Responder)

```
DNF = (Du & Rs) | (Ms & Ru) | (Mu & Rs) | (Rs & Ru) | (Mi & Ms & Ry) | (Mi & My & Rs) | (Mi & Rs & Ry) |
(Mr & Ms & Ry) | (Mr & My & Rs) | (Mr & Rs & Ry) | (Ms & Mv & Ry) | (Ms & My & Rc) | (Ms & My & Rv) |
(Ms & Rc & Ry) | (Ms & Rv & Ry) | (Mv & My & Rs) | (Mv & Rs & Ry) | (My & Rc & Rs) | (My & Rs & Rv) |
(Rc & Rs & Ry) | (Rs & Rv & Ry)
```

Comparison with previous work

- 2018: J. A. Donenfeld and K. Milner, "Formal verification of the WireGuard protocol" (*Symbolic WireGuard*)

$\Longrightarrow$ **More accurate analysis !**

Illustration with agreement on 3^{rd} message

Extract, compute DNF. Requires the use of symbols:

- `Du`, `Dv`: adversary corrupts public keys distribution
- `Ru`, `Rv`, `Rx`, `Ry`, `Rs`, `Rc`: adversary gets private keys (`u`, `v`, `x`, `y`), psk (`s`) or precomp. value (`c`)
- `Mu`, `Mv`, `Mx`, `My`, `Ms`, `Mi`, `Mr`: adversary sets private keys (`u`, `v`, `x`, `y`), psk (`s`) or precomp value (`i` for Initiator, `r` for Responder)

```
DNF = (Du & Rs) | (Ms & Ru) | (Mu & Rs) | (Rs & Ru) | (Mi & Ms & Ry) | (Mi & My & Rs) | (Mi & Rs & Ry) |
(Mr & Ms & Ry) | (Mr & My & Rs) | (Mr & Rs & Ry) | (Ms & Mv & Ry) | (Ms & My & Rc) | (Ms & My & Rv) |
(Ms & Rc & Ry) | (Ms & Rv & Ry) | (Mv & My & Rs) | (Mv & Rs & Ry) | (My & Rc & Rs) | (My & Rs & Rv) |
(Rc & Rs & Ry) | (Rs & Rv & Ry)
```

Performances

- Evaluate **4860** queries $\approx$ 14,5 hours (with 5 parallel sets)
- Compute DNF: $\approx$ 2 hours (compute first CNF, then DNF(CNF))

Conclusion

- Work still in progress, next steps:
  - Analyse "false" queries
  - Link with TAMARIN prover
  - Mount concrete attacks
- Verified properties:
  - Agreement
  - Key and message secrecy
  - Anonymity
- Contributions:
  - Complete model of WireGuard
  - Precise threat model, including initial key distribution and precomputations
  - Process with SAPIC$^+$, PROVERIF, TAMARIN