Håck ma's 2024, Schloss in Ottenschlag
2024-08-29

The Rosenpass protocol and attacks we found against WireGuard while developing it

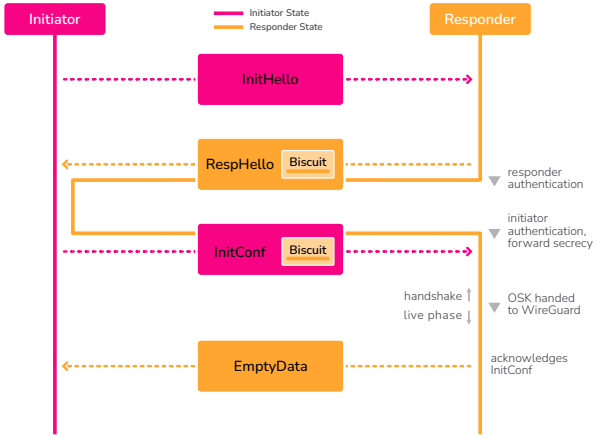
Benjamin Lipp, **Karolin Varner**
with support from Alice Bowman, Marei Peischl, and Lisa Schmidt
https://rosenpass.eu

This is the Plan

1. **Introducing Rosenpass,** briefly!
2. **The design of Rosenpass** and basics about post-quantum protocols!
3. **Hybrid Security** how it can be done and how we do it!
4. **Security trade-offs** in Rosenpass and WireGuard (aka attacks)!
5. **Protocol proofs** what they are and why they give me headaches!

Introducing Rosenpass, briefly

- A post-quantum secure key exchange **protocol** based on the paper Post-Quantum WireGuard[PQWG]
- An open-source Rust **implementation** of that protocol, already in use
- A way to secure WireGuard setups against quantum attacks
- A **post-quantum secure VPN**
- A governance **organization** to facilitate development, maintenance and adoption of said protocol
- A translation research organization



The design of Rosenpass

and how to build post-quantum protocols

In the following slides, you will learn...

...that most cryprographic applications today are susceptible against attacks from quantum computers.

...that this is not fundamental to cryptography, but that pre-quantum protocols are simply a more efficient.

...that – cryptographically speaking – the difference between pre-quantum and post-quantum crypto is about a subtle difference in function interface.

Glossary: Post-quantum security

Pre-quantum Cryptography is...

...susceptible to attacks from quantum computers.

- Specifically, to *Shor's Algorithm*
- Quite fast
- Widely used
- Widely widely trusted

Post-quantum cryptography is...

...not susceptible to attacks from quantum computers.

- generally less efficient.
- much bigger ciphertexts.
- hard to use on embedded devices.
- adoped in the last decade.
- less analyzed.

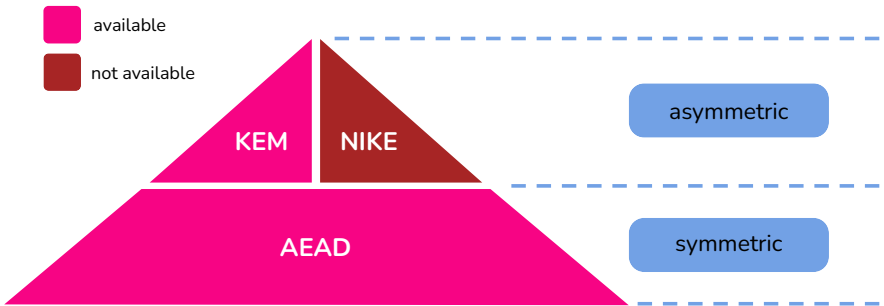
Hybrid cryptography combines...

...the quantum-resistance of post-quantum cryptography and the trustedness of pre-quantum cryptography.

It is...

- about as inefficient as post-quantum cryptography.
- not widely adopted. Which is a major problem.

What post-quantum got



NIKEs and KEMs

Key Encapsulation Method

```
trait Kem {
  // Secret, Public, Symmetric, Ciphertext
  type Sk; type Pk; type Shk; type Ct;
  fn genkey() -> (Sk, Pk);
  fn encaps(pk: Pk) -> (Shk, Ct);
  fn decaps(sk: Pk, ct: Ct) -> Shk;
}
#[test]
fn test<K: Kem>() {
  let (sk, pk) = K::genkey();
  let (shk1, ct) = K::encaps(pk);
  let shk2 = K::decaps(sk, ct);
  assert_eq!(shk1, shk2);
}
```

Non Interactive Key Exchange

```
trait Nike {
  // Secret, Public, Symmetric
  type Sk; type Pk; type Shk;
  fn genkey() -> (Sk, Pk);
  fn nike(sk: Sk, pk: Pk) -> Shk;
}
#[test]
fn test<N: Nike>() {
  let (sk1, pk1) = N::genkey();
  let (sk2, pk2) = N::genkey();
  let ct1 = N::nike(sk1, pk2);
  let ct2 = N::nike(sk2, pk1);
  assert_eq!(ct1, ct2);
}
```

NIKEs and KEMs

Key Encapsulation Method

```
fn Kem::encaps(Pk) -> (Shk, Ct);
fn Kem::decaps(Pk, Ct) -> Shk;

(shk, ct) = encaps(pk);
assert!(decaps(sk, ct) = shk)
```

Think of it a encrypting a key and sending it to the partner. Aka. Diffie-Hellman.

- Secrecy
- Implicit authentication of recipient (assuming they have the shared key, they must also have their secret key)

Non Interactive Key Exchange

```
fn nike(sk: Sk, pk: Pk) -> Shk;

assert!(nike(sk1, pk2) = nike(pk2, sk1));
```

I don't know a good analogy, but ote how the keypairs are *crossing over* to each other.

- Secrecy
- Mutual authentication (for each party: assuming they have the shared key, they must also have their secret key)

NIKEs and KEMs: Key exchange

Key Encapsulation Method

Responder Authentication: Initiator encapsulates key under the responder public key.

Initiator Authentication: Responder encapsulates key under the initiator public key.

Forward-secrecy: In case the secret keys get stolen, either party generates a temporary and has the other party encapsulate a secret under that keypair.

How to do this properly? See

Non Interactive Key Exchange

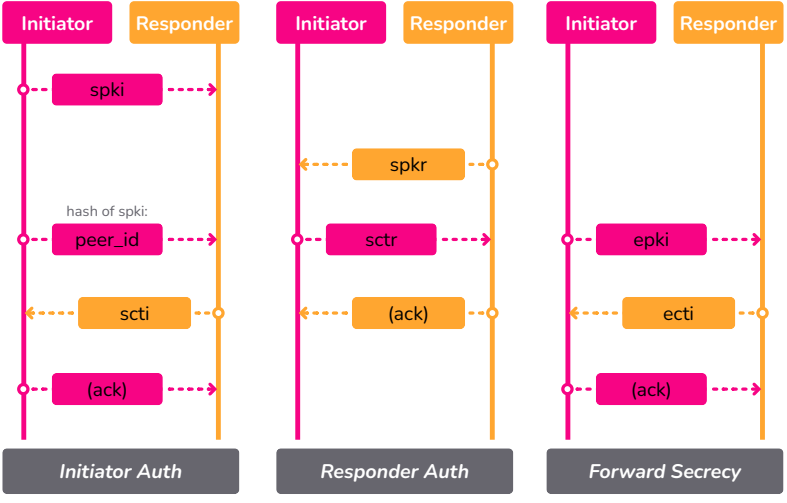
Responder Authentication: Static-static NIKE since NIKE gives mutual authentication.

Initiator Authentication: Static-static NIKE since NIKE gives mutual authentication.

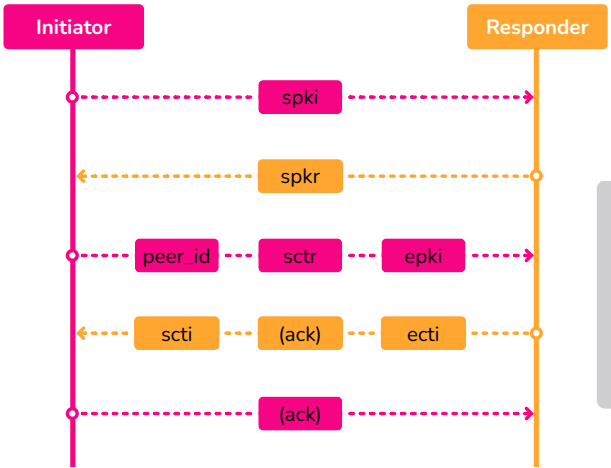
Forward-secrecy: Another nike, involving a temporary keypair.

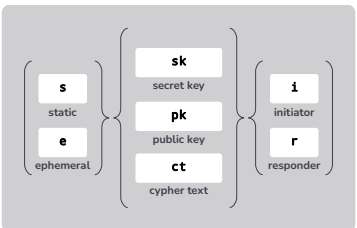
How to do this properly? See the Noise Protocol Framework.

Rosenpass Kex Exchange Parts

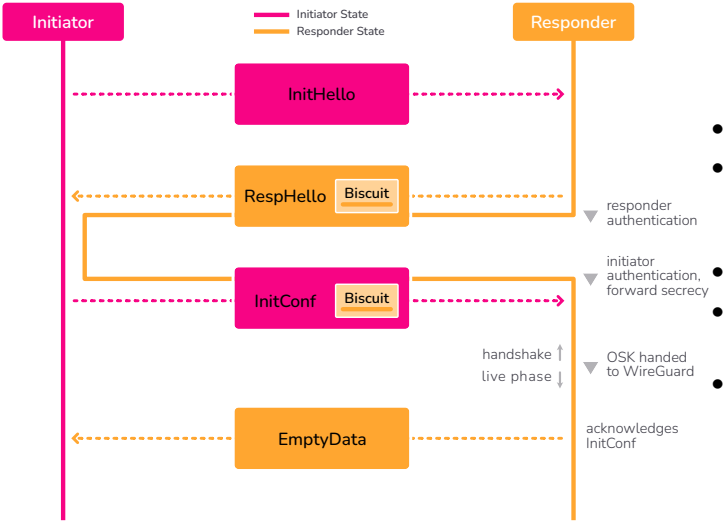


Rosenpass Unified Key Exchange



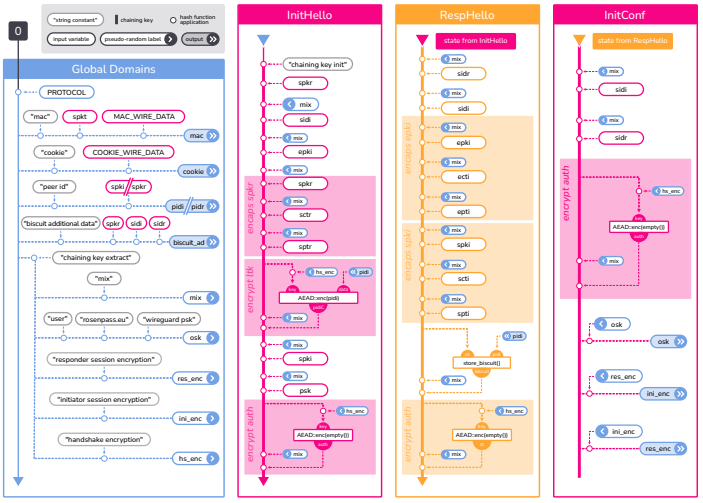


Rosenpass Protocol Features

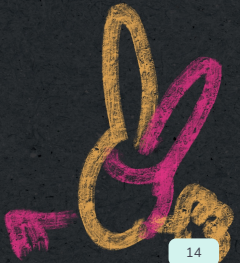


- Authenticated key exchange
- Three KEM operations interleaved to achieve mutual authentication and forward secrecy
- No use of signatures
- First package (InitHello) is unauthenticated
- Stateless responder to avoid disruption attacks

Rosenpass Key Derivation Chain



Hybridization



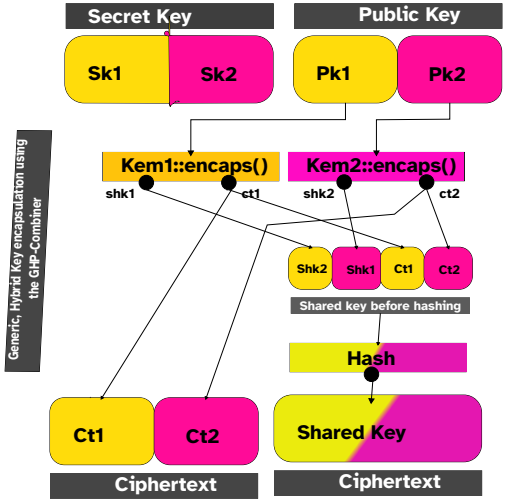
In the following slides, you will learn...

...that hybrid, practical, post-quantum cryptography is built by combining pre-quantum and post-quantum primitives.

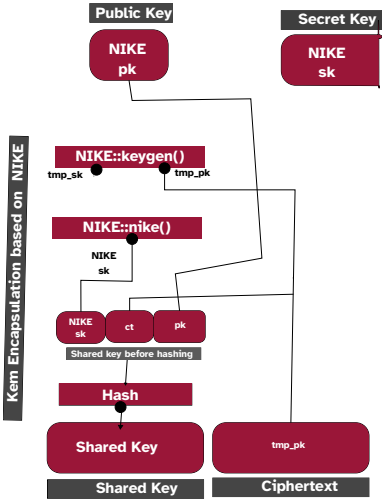
...that key encapsulation methods can be combined, rendering a protocol like Rosenpass useful in pre-quantum as well as post-quantum settings.

...why combining protocols like WireGuard and Rosenpass directly is still useful to enable code-reuse and to avoid loosing trust in established systems like WireGuard.

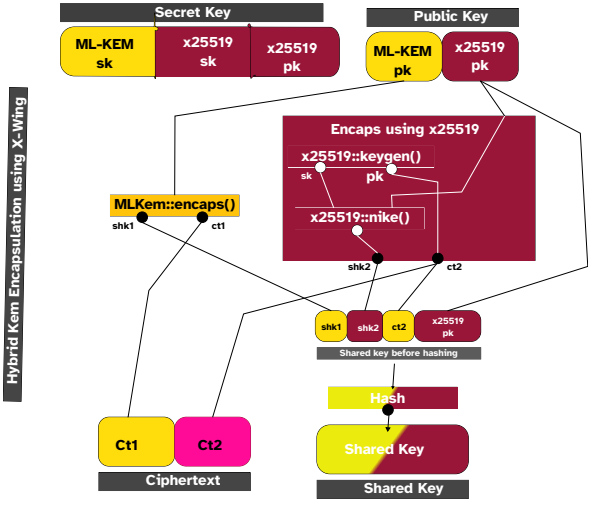
Combining two KEMs with the GHP-Combiner



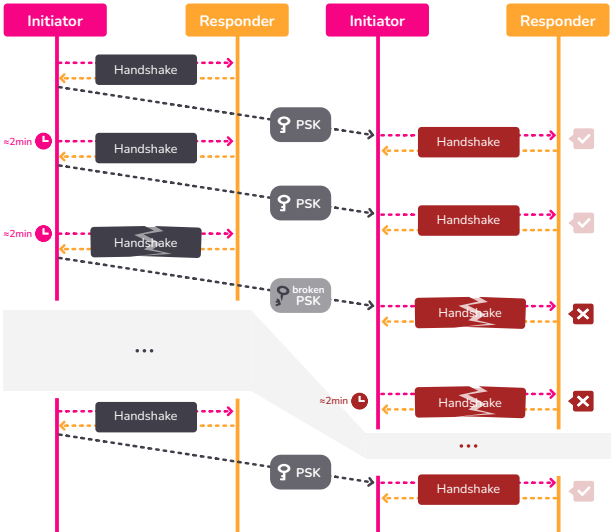
Turning a NIKE into a KEM



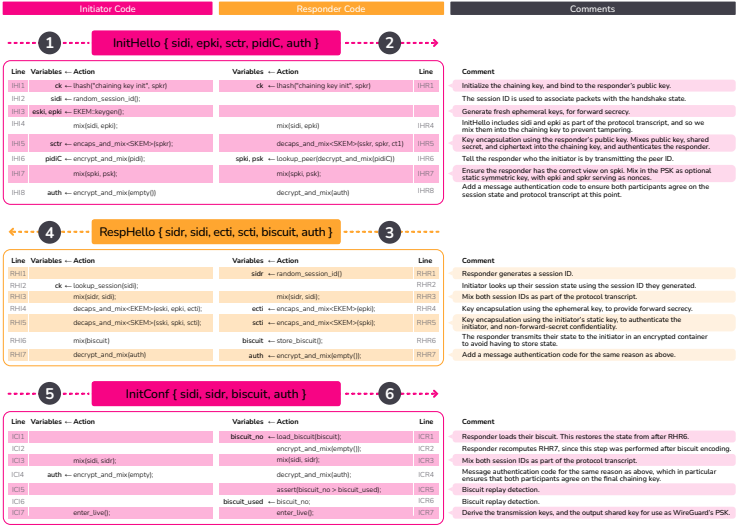
X-Wing


Secret Key
ML-KEM sk
x25519 sk
x25519 pk
Public Key
ML-KEM pk
x25519 pk
Hybrid Kem Encapsulation using X-Wing
Encaps using x25519
x25519::keygen()
sk
pk
x25519::nike()
MLKem::encaps()
shk1
ct1
shk2
ct2
shk1
shk2
ct2
x25519 pk
Shared key before hashing
Hash
Ct1
Ct2
Ciphertext
Shared Key
Shared Key

Rosenpass & WireGuard Hybridization



Full Protocol Reference in the Whitepaper



Attack-time

Rosenpass and WireGuard: Advanced Security

Limited Stealth:

- Protocol should not respond without pre-auth.
- Proof of IP ownership (cookie mechanism) prevents full stealth
- Adv. needs to know responder public key

Limited Identity Hiding:

- Adversary cannot recognize peers unless their public key is known
- This is incomplete!

CPU DOS mitigation:

- Attacker should not easily trigger public key operations
- Preventing CPU exhaustion using network amplification
- Proof of IP ownership

Triumphs ~ Secrecy & Non-Interruptability

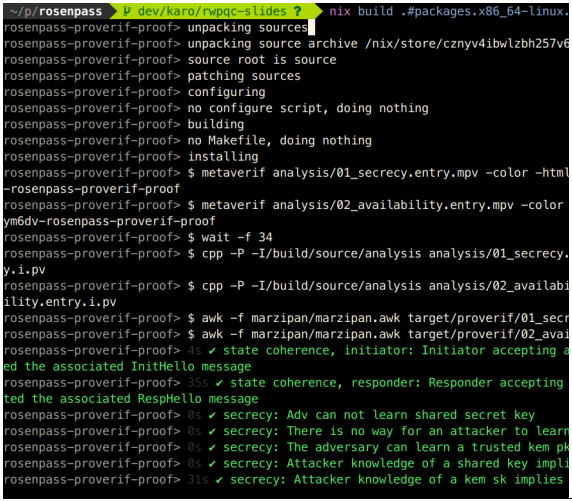
Modeling of Rosenpass

Using ProVerif

Symbolic Modeling of Rosenpass

```
~/p/rosenpass  dev/karo/rwpqc-slides ?  nix build .#packages.x86_64-linux.proof-proverif --print-build-logs [17/17]
rosenpass-proverif-proof> unpacking sources
rosenpass-proverif-proof> unpacking source archive /nix/store/cznyv4ibwlzbh257v6lzx8r8al4cb0v0-source
rosenpass-proverif-proof> source root is source
rosenpass-proverif-proof> patching sources
rosenpass-proverif-proof> configuring
rosenpass-proverif-proof> no configure script, doing nothing
rosenpass-proverif-proof> building
rosenpass-proverif-proof> no Makefile, doing nothing
rosenpass-proverif-proof> installing
rosenpass-proverif-proof> $ metaverif analysis/01_secrecy.entry.mpv -color -html /nix/store/gidm68r04lkpanvkgz48527qf6nym6dv
-rosenpass-proverif-proof
rosenpass-proverif-proof> $ metaverif analysis/02_availability.entry.mpv -color -html /nix/store/gidm68r04lkpanvkgz48527qf6n
ym6dv-rosenpass-proverif-proof
rosenpass-proverif-proof> $ wait -f 34
rosenpass-proverif-proof> $ cpp -P -I/build/source/analysis analysis/01_secrecy.entry.mpv -o target/proverif/01_secrecy.entr
y.i.pv
rosenpass-proverif-proof> $ cpp -P -I/build/source/analysis analysis/02_availability.entry.mpv -o target/proverif/02_availab
ility.entry.i.pv
rosenpass-proverif-proof> $ awk -f marzipan/marzipan.awk target/proverif/01_secrecy.entry.i.pv
rosenpass-proverif-proof> $ awk -f marzipan/marzipan.awk target/proverif/02_availability.entry.i.pv
rosenpass-proverif-proof> 4s ✔ state coherence, initiator: Initiator accepting a RespHello message implies they also generat
ed the associated InitHello message
rosenpass-proverif-proof> 35s ✔ state coherence, responder: Responder accepting an InitConf message implies they also genera
ted the associated RespHello message
rosenpass-proverif-proof> 0s ✔ secrecy: Adv can not learn shared secret key
rosenpass-proverif-proof> 0s ✔ secrecy: There is no way for an attacker to learn a trusted kem secret key
rosenpass-proverif-proof> 0s ✔ secrecy: The adversary can learn a trusted kem pk only by using the reveal oracle
rosenpass-proverif-proof> 0s ✔ secrecy: Attacker knowledge of a shared key implies the key is not trusted
rosenpass-proverif-proof> 31s ✔ secrecy: Attacker knowledge of a kem sk implies the key is not trusted
```

Symbolic Modeling of Rosenpass



- Symbolic modeling using ProVerif
- Proofs treated as part of the codebase
- Uses a model internally that is based on a fairly comprehensive Maximum Exposure Attacks (MEX) variant
- Covers non-interruptability (resistance to disruption attacks)
- Mechanized proof in the computational model is an open issue

What are Non-Interruptability and State Disruption Attacks?

- Attacker trying to perform a protocol-level DOS attack
- Attacker may observe messages
- Attacker may insert messages, but they may not drop or modify messages
- Halfway between an active and passive attacker:
  - For a fully active attacker state disruption is trivial; they can just drop messages

Non-Interruptability: More Formally

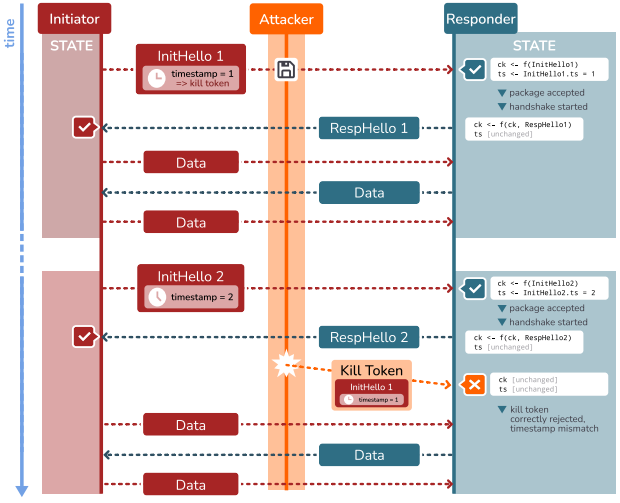
For every pair of traces `tmin`, `tmax` where trace `tmax` can be formed by insertion of messages/oracle calls into `tmin`, the result of `tmin` and `tmax` should remain the same.

- Let `Result` be the set of possible protocol results
- Let `Trace` be the set of possible protocol traces
- Let $\texttt{res}(\texttt{t}) : \texttt{Trace} \to \texttt{Result}$ determine the protocol result given $\texttt{t} : \texttt{Trace}$
- Let $\texttt{t1} \supseteq \texttt{t2} : \texttt{Trace} \to \texttt{Trace} \to \texttt{Prop}$ denote that $\texttt{t2}$ can be formed by insertion of elements into $\texttt{t1}$
- $\forall(\texttt{tmin}, \texttt{tmax}) : \texttt{Trace} \times \texttt{Trace}; \texttt{tmin} \supseteq \texttt{tmax} \to \texttt{res}(\texttt{tmin}) = \texttt{res}(\texttt{tmax})$

Trials ~ Attacks found

ChronoTrigger

Retransmission Protection in WireGuard



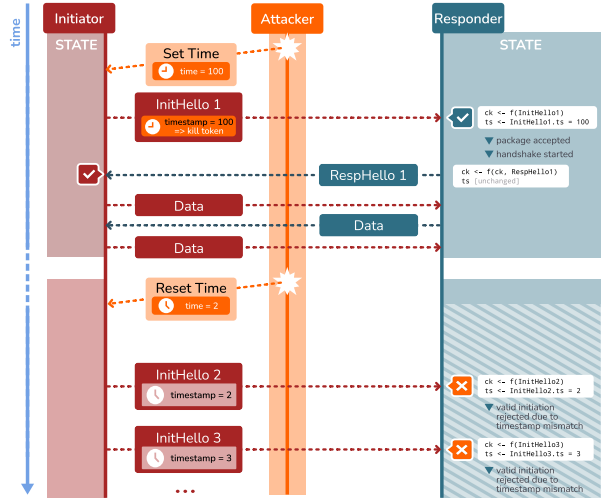
- Replay attacks thwarted by counter
- Counter is based on real-time clock
- Responder is semi-stateful (one retransmission at program start may be accepted, but this does not affect protocol security)

⇒ WG requires *either* reliable real-time clock *or* stateful initiator

⇒ Adversary can attempt replay, but this cannot interrupt a valid handshake by the initiator

! Assumption of reliable system time is invalid in practice!

ChronoTrigger Attack: Immediate Execution



A. Preparation phase:

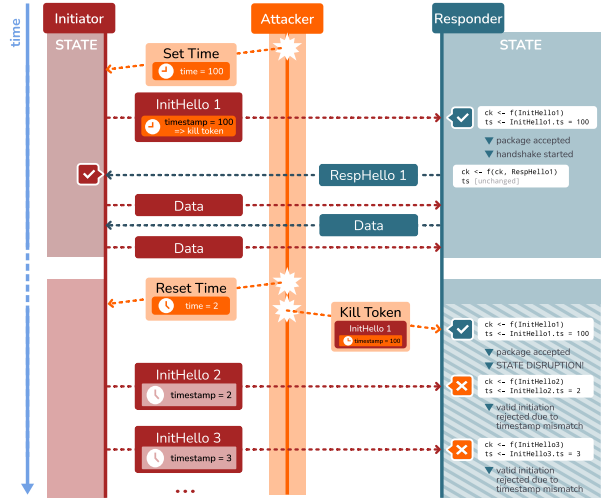
1. **Attacker** sets *initiator system time* to a future value
2. **Attacker** waits while both peers are performing a valid handshake

B. Direct execution phase:

1. **Attacker** lets system time on initiator reset

=> Initiation now fails due to counter mismatch

ChronoTrigger Attack: Delayed Execution



A. Preparation phase:

1. **Attacker** sets *initiator system time* to a future value
2. **Attacker** records *InitHello* as *KillToken* while both peers are performing a valid handshake

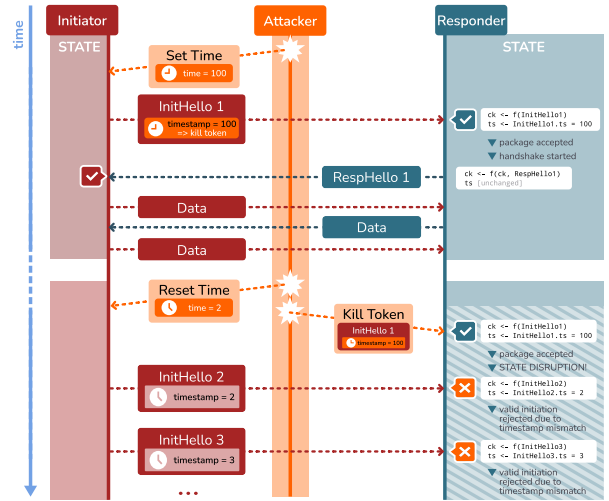
... both peers are being reset ...

B. Delayed execution phase:

1. **Attacker** sends *KillToken* to responder, setting their timestamp to a future value

⇒ Initiation now fails again due to timestamp mismatch

ChronoTrigger Attack: Delayed Execution



Attacker gains

- Extremely cheap protocol-level DOS

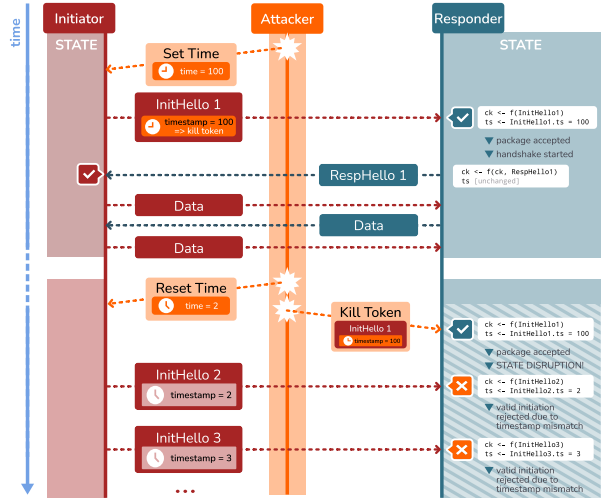
Preparation phase, attacker needs:

- Eavesdropping of initiator packets
- Access to system time

Delayed execution, attacker needs:

- No access beyond message transmission to responder

ChronoTrigger Attack: Delayed Execution

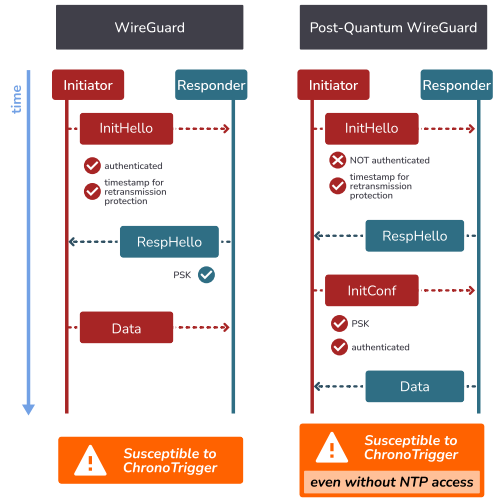


Gaining access to system time:

- Network Time Protocol is insecure, Mitigations are of limited use

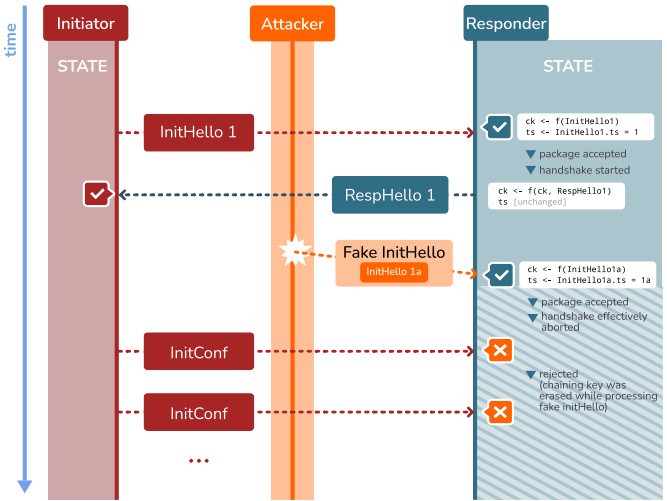
⇒ Break NTP *once*; kill token lasts forever

ChronoTrigger: Changes in Post-Quantum WG



- *InitHello* is unauthenticated
- Retransmission counter is kept
- PQWG assumes a pre-shared key to authenticate InitHello instead (the authors recommend deriving the PSK from both public keys)
- PSK evaluated twice, during InitHello *and* InitConf processing

ChronoTigger against Post-Quantum WireGuard



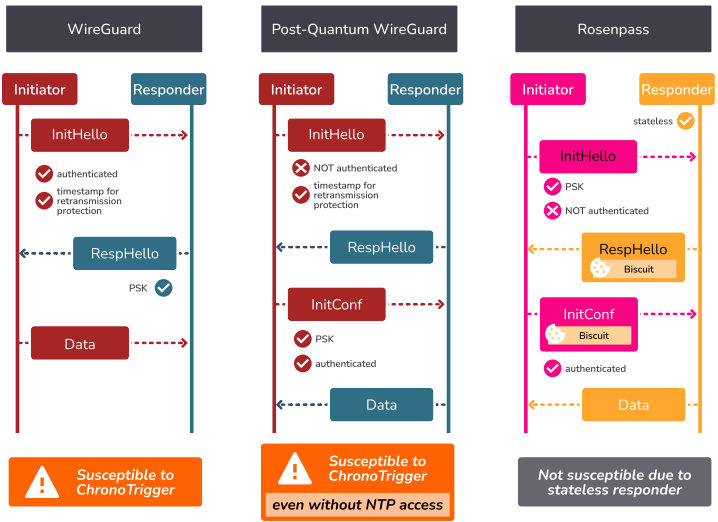
No PSK/Public keys as PSK

- Attacker needs access to public keys
- The attack is trivial (attacker just forges *InitHello*)

With PSK

- Replay attack with NTP access from classic WireGuard still applies

ChronoTrigger: Changes in Rosenpass



- InitHello remains unauthenticated
- PSK used in InitHello exclusively
- Responder state is moved into a cookie called *Biscuit*

⇒ Stateless responder prevents ChronoTrigger attack

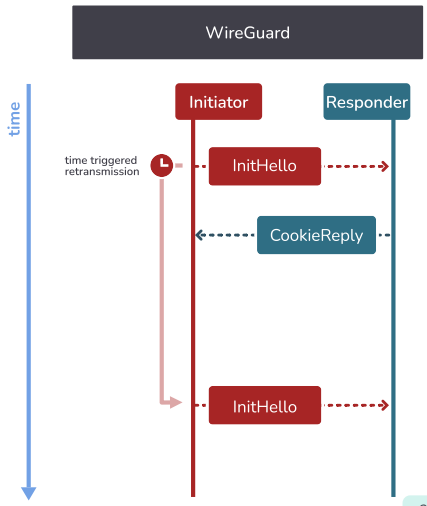
Trials ~ Attacks found

CookieCutter

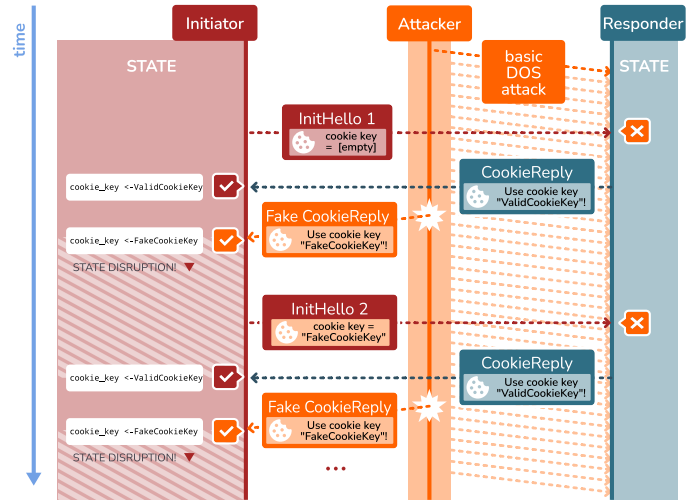
CookieCutter Attack

ROSENPASS

> I am under load. Prove that you are not using IP address impersonation before I process your handshake!
> This message contains a *cookie key*. Use it to prove that you can receive messages sent to your address when retransmitting your *InitHello* packet.
>
> **A WireGuard CookieReply**, ca. 2014



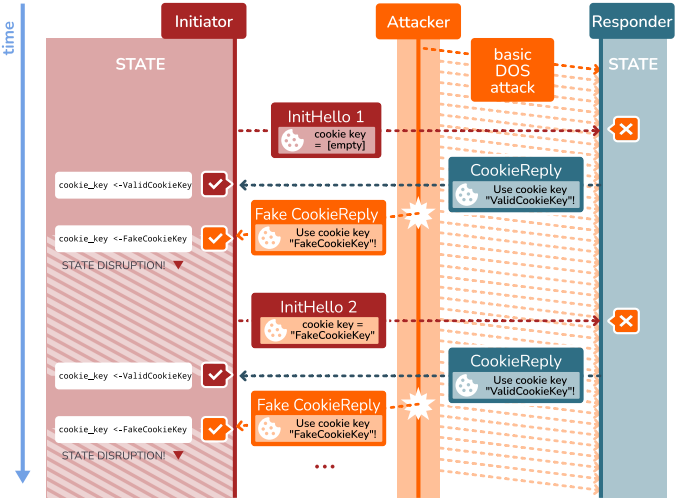
CookieCutter Attack



1. **Attacker** begins continuous DOS attack against responder
2. **Initiator** begins handshake, sends *InitHello*
3. **Responder** replies with *CookieReply*
   > **CookieReply:** I am under load. Prove you are not using an IP spoofing attack with this *cookie key*.
4. **Initiator** Initiator stores cookie key and waits for their retransmission timer
5. **Attacker** forges a cookie reply with a fake *cookie key*
6. **Initiator** Initiator overwrites the valid cookie key with the fake one

... Repeat ad nauseam

CookieCutter Attack



Attacker gains:

- Cheap protocol-level DOS

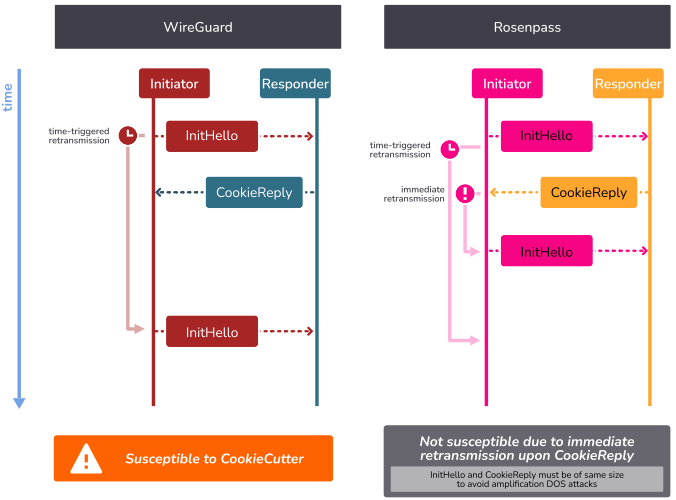
Attacker needs:

- Knowledge of public keys
- Good timing

Role switching:

- WireGuard sometimes uses role switching
- To account for that, the attack can be performed against both peers

CookieCutter: Post-Quantum WG & Rosenpass



Post-Quantum WireGuard

- No change.

Rosenpass

- Immediate retransmission of *InitHello* upon receiving *CookieReply*
- *CookieReply* and *InitHello* must be of same size to prevent DOS amplication attacks

⇒ Rosenpass is protected from CookieCutter attacks

Trials ~ Advanced Security Properties

Knock Patterns

Rosenpass and WireGuard: Advanced Security

CPU DOS mitigation:

- No change on the protocol level.
- 😐 Slightly worsened in practice because PQ operations are more expensive than elliptic curves

Limited Stealth:

- No change in Rosenpass, but we should have **full stealth**!

⇒ Remove cookie mechanism?

😐 This would affect the CPU DOS mitigation too much.

Limited Identity Hiding:

- No change in Rosenpass, but we should have **full identity hiding**!

⇒ Do not use pre-authentication with public key?

😐 This would affect the CPU DOS mitigation, possibly too much.

Choose Two: Stealth, Identity Hiding, CPU DOS Mit.

CPU DOS Protection

asymmetric functions are not called unless pre-auth has succeeded

KNOCK PATTERNS

proof of IP ownership

interactive proof of work

KNOCK PATTERNS

initiator public key

responder public keys

weaker + more performant form of authentication (e.g. EC-based NIKE)

WireGuard

Stealth

no interactive pre-auth allowed

Identity Hiding

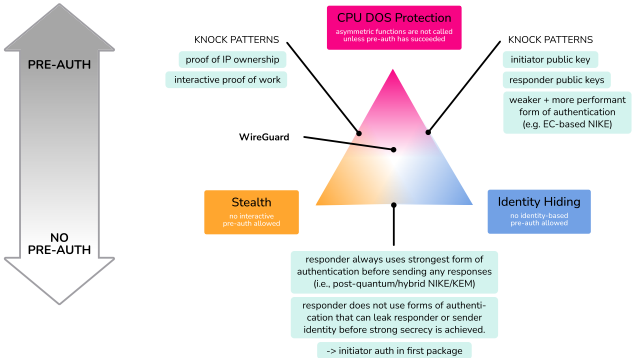
no identity-based pre-auth allowed

responder always uses strongest form of authentication before sending any responses (i.e., post-quantum/hybrid NIKE/KEM)

responder does not use forms of authenti-cation that can leak responder or sender identity before strong secrecy is achieved.

-> initiator auth in first package

WireGuard and Rosenpass Trade-Offs



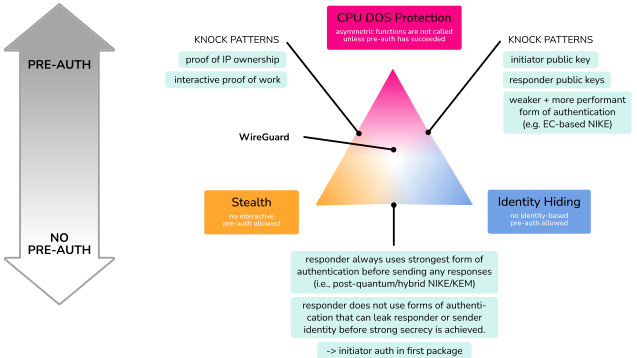
CPU DOS Mitigation:

- There is no clear optimum here.
- CPU DOS mitigation is never calling asymmetric crypto unless we know it succeeds (circular reasoning)

Stealth:

Identity hiding:

WireGuard and Rosenpass Trade-Offs



CPU DOS Mitigation:

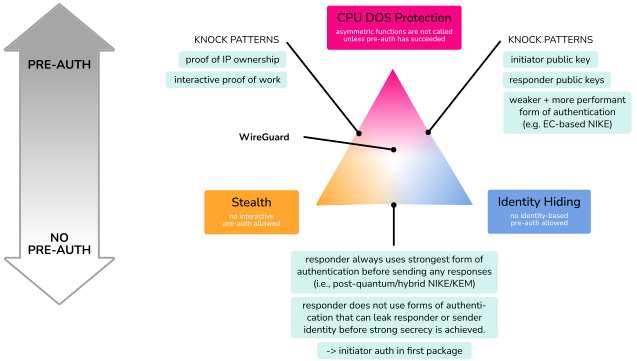
Stealth:

- Broken on DOS attacks assuming recipient is known

⇒ This seems acceptable

Identity hiding:

WireGuard and Rosenpass Trade-Offs



CPU DOS Mitigation:

Stealth:

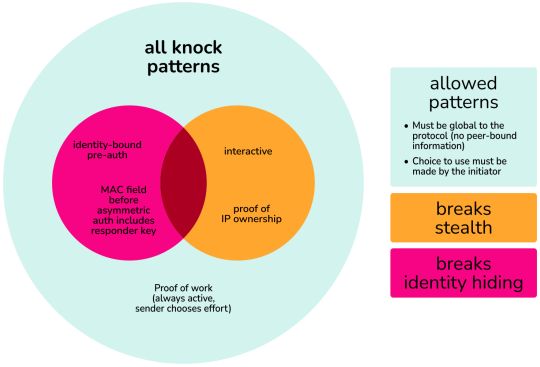
Identity hiding:

- Broken on knowledge of public keys

⇒ This seems inacceptable!

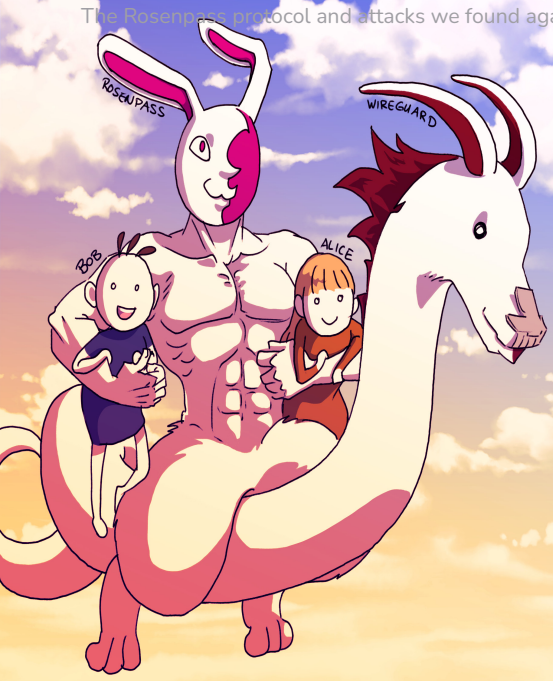
⇒ Investigate proper identity hiding without overly impacting stealth and CPU DOS mitig.

Knock Patterns



- We choose to think of WireGuard's and Rosenpass' pre-auth as "Knock Patterns"
- These knock patterns have severe trade-offs.
- Interactive knock pattern (cookie mechanism) breaks stealth
- Identity-based knock patterns (e.g., knowledge of public key) breaks identity hiding

⇒ Avoid identity-bound knock patterns
⇒ Minimize interactive knock patterns
⇒ Explore other (allowed) knock patterns



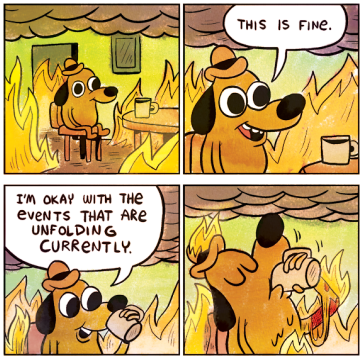
Tribulations ~ Tooling

Oh These Proof Tools

Vive la Révolution! Against the

Bourgeoisie of Proof Assistants!

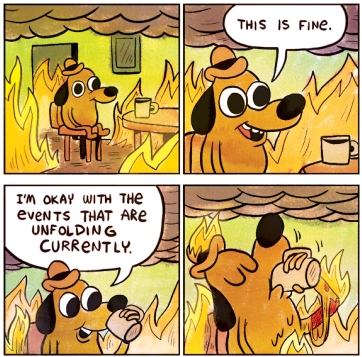
Problematic Parts of Pen-and-Paper Proofs



Bellare and Rogaway: [BR06]
many “essentially unverifiable” proofs, “crisis of rigor”

Halevi: [Hal05]
some reasons are social, but “our proofs are truly complex”

Problematic Parts of Pen-and-Paper Proofs



Bellare and Rogaway: [BR06]
many “essentially unverifiable” proofs, “crisis of rigor”

Halevi: [Hal05]
some reasons are social, but “our proofs are truly complex”

Joseph Jaeger: [ProTeCS 2024, Workshop at Eurocrypt]
technical and social reasons
why and for whom do we write proofs?

Problematic Parts of Pen-and-Paper Proofs

Bellare and Rogaway: [BR06]
many “essentially unverifiable” proofs, “crisis of rigor”

Halevi: [Hal05]
some reasons are social, but “our proofs are truly complex”

Joseph Jaeger: [ProTeCS 2024, Workshop at Eurocrypt]
technical and social reasons
why and for whom do we write proofs?

We’d like to add:
pen-and-paper proofs are hard to maintain, update, reuse
especially for 3rd parties

Can proofs become part of a continuous engineering effort?

Tools to the Table!

Bellare and Rogaway [BR06], Halevi [Hal05]:
Call for “automated tools, that can help write and verify game-based proofs”

Tools to the Table!

Bellare and Rogaway [BR06], Halevi [Hal05]:
Call for “automated tools, that can help write and verify game-based proofs”

Do the tools *actually help*? And if yes, whom?

ProVerif, Tamarin, CryptoVerif, EasyCrypt:
We like these tools, they are good!

Tools to the Table!

Bellare and Rogaway [BR06], Halevi [Hal05]:
Call for “automated tools, that can help write and verify game-based proofs”

Do the tools *actually help*? And if yes, whom?

ProVerif, Tamarin, CryptoVerif, EasyCrypt:
We like these tools, they are good!

They mostly help the *formal verification experts* to:

- do analyses themselves, write papers
- develop proof methodologies, foundation work for formal methods

Friction+Frustration for the Working Cryptographer

Tooling

Documentation:

Output:

Input Language:

Proof Language:

Friction+Frustration for the Working Cryptographer

Tooling

- Syntax highlighting, favorite editor
- Engineering for large models: syntax rewriting, syntactic sugar, macros
- Comfortable tooling to inspect intermediate games (CryptoVerif)

Documentation:

Output:

Input Language:

Proof Language:

Friction+Frustration for the Working Cryptographer

Tooling

- Syntax highlighting, favorite editor
- Engineering for large models: syntax rewriting, syntactic sugar, macros
- Comfortable tooling to inspect intermediate games (CryptoVerif)

Documentation: Often incomplete; step from example to research too big

Output: hard to understand for non-experts

Input Language: Unintuitive? Inaccessible? Mixed signals!

Proof Language:

Friction+Frustration for the Working Cryptographer

Tooling

- Syntax highlighting, favorite editor
- Engineering for large models: syntax rewriting, syntactic sugar, macros
- Comfortable tooling to inspect intermediate games (CryptoVerif)

Documentation: Often incomplete; step from example to research too big

Output: hard to understand for non-experts

Input Language: Unintuitive? Inaccessible? Mixed signals!

Proof Language: not enough flexibility and leeway compared to pen-and-paper

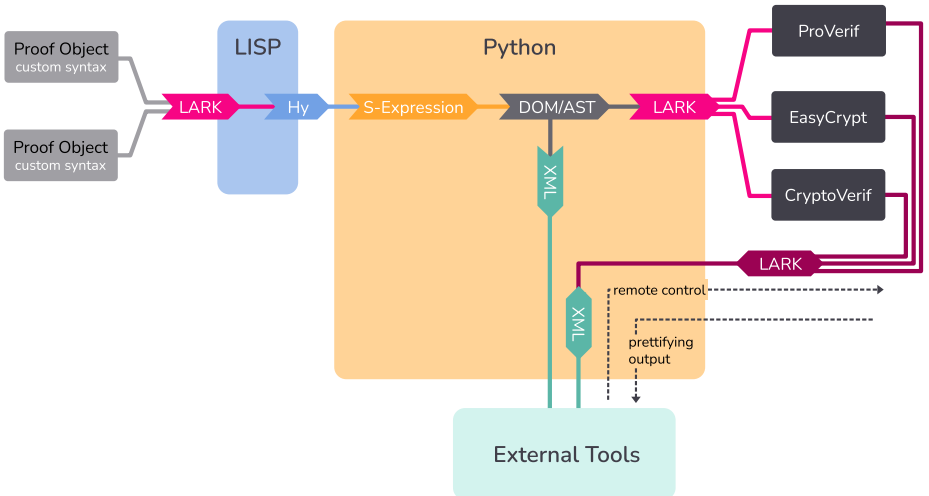
- hand-waving, unsafe blocks
- support for incremental process

Maths is something fundamentally communicative

... and mechanised proofs have not covered this thus far.

> Everything had a name, and each name gave birth to a new thought.
>
> Helen Keller (1880-1968) in The Day Language Came into My Life

Rosenpass going Rube-Goldberg



Idea: Build a framework around existing tools

Keep expressivity and preciseness

Generate and parse their languages

Open them up to ecosystems in Python, Lisp, XML

Epilogue

Conclusion

Rosenpass

- Post-quantum secure AKE
- Same security as WireGuard
- Improved state disruption resistance
- Transfers key to WireGuard for hybrid security

Protocol Findings

- **CookieCutter**: DOS exploiting WireGuard cookie mechanism
- **ChronoTrigger**: DOS exploiting insecure system time to attack WireGuard
- There is a **trade-off** between identity hiding, stealth, and CPU-exhaustion DOS protection

Talk To Us

- About why we should use Tamarin (or SAPIC+?) over ProVerif
- State disruption attacks
- Stealth and Identity hiding
- Adding syntax rewriting to the tool belt of mechanized verification in cryptography

rosenpass.eu

Appendix — Here Be Dragons

Bibliography

[PQWG]: https://eprint.iacr.org/2020/379

Rosenpass going Rube-Goldberg: The Details

- Embed cryptographic proof syntax in Lisp S-Expressions
- Translate Lisp code to Python using the Hy language (Lisp that compiles to Python)
- Translate S-Expression code to AST or DOM
- Translate AST or DOM to ProVerif/Tamarin/CryptoVerif/EasyCrypt code using the LARK code parser/generator
- Remote control ProVerif/Tamarin/CryptoVerif/EasyCrypt by
  - Parsing their command line output using LARK
  - (Possibly using the language server interface for more interactive features)
- Provide custom syntax using
  - Lisp Macros
  - Extending LARK-based syntax parsers (to add custom syntactic elements)
  - AST Rewriting for more complex adaption
- Integrate with external tools by exporting our AST as XML
  - XML is just a convenient grammar for trees
  - We do not need to support the full complexity of XML including XML style sheets and such things