arrows.meta >=Latex[round]
dateplot
commentEnv
cryptoverif morekeywords=collision, const, crypto, define, defined, do, else, end, equation, equiv, event,
event$_{a}bort, expand, find, forall, foreach, fun, get, implementation, in, if, inj, insert, length, let, l$
$< -, < -R,, sensitive = true, morecomment = [s](**), morestring =$
$[b]", cvoutputmorekeywords = , otherkeywords = , sensitive = true, morecomment = [s](**), mo$

Rosenpass

Securing & Deploying Post-Quantum WireGuard

Karolin Varner, with Benjamin Lipp, Wanja Zaeske, Lisa Schmidt

RWPQC23 | https://rosenpass.eu/whitepaper.pdf

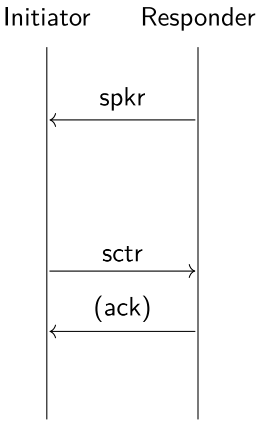
November 23, 2023

Structure of the talk

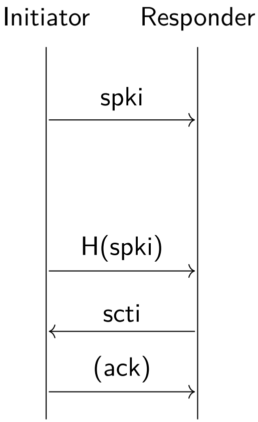
- Post-quantum WireGuard[1]: How to build an interactive key exchange from KEMs
- Contribution: State Disruption Attacks & cookies as a defense
- Contribution: Symbolic analysis of the Rosenpass protocol
- Contribution: Noise-like specification
- Contribution: New hashing & domain separation scheme
- Contribution: Reference implementation – Securing WireGuard in practice

[1] Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Florian Weber, and Philip R. Zimmermann. "Post-quantum WireGuard". In: 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021. Full version: https://eprint.iacr.org/2020/379

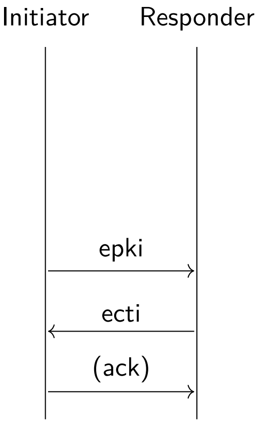
Post-quantum WireGuard: Three encapsulations



Responder Auth

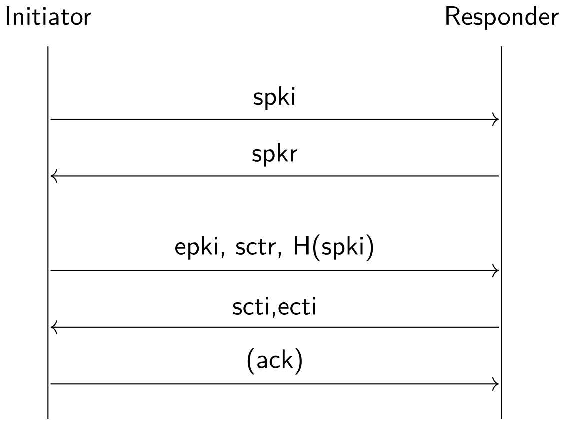


Initiator Auth



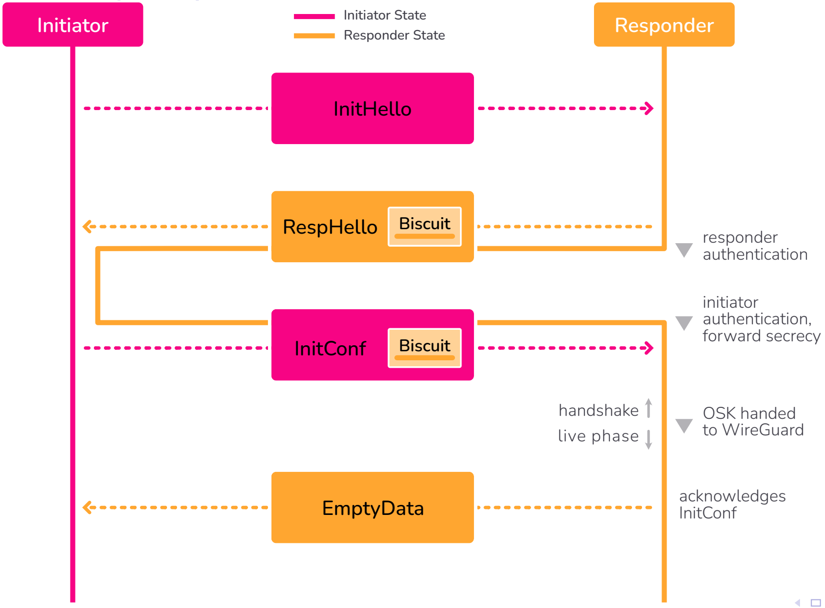
Forward secrecy

Combining the three encapsulations in one protocol



Note that the initiator is not authenticated until they send "(ack)".

The Rosenpass protocol



CVE-2021-46873 – DOS against WireGuard through NTP

- The replay protection in classic WireGuard assumes a monotonic counter
- But the system time is attacker controlled because NTP is insecure
- This generates a kill packet that abuses replay protection and renders the initiator's key-pair useless
- Attack is possible in the real world!
- Similar attack in post-quantum WireGuard is worse since InitHello is unauthenticated
- Solution: Biscuits

Security analysis of rosenpass

- CryptoVerif in progress
- Symbolic analysis using ProVerif
- Code is part of the software repository & build system
- Symbolic analysis is fast (about five minutes), runs in parallel and is

Proverif in technicolor

```
~/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
```

Noise-like specification (easier for engineers)

Initiator Code | Responder Code | Comments

1 InitHello { sidi, epki, sctr, pidiC, auth } 2

Line	Variables ← Action	Variables ← Action	Line	Comment
IHI1	ck ← lhash("chaining key init", spkr)	ck ← lhash("chaining key init", spkr)	IHR1	Initialize the chaining key, and bind to the responder's public key.
IHI2	sidi ← random_session_id();			The session ID is used to associate packets with the handshake state.
IHI3	eski, epki ← EKEM::keygen();			Generate fresh ephemeral keys, for forward secrecy.
IHI4	mix(sidi, epki);	mix(sidi, epki)	IHR4	InitHello includes sidi and epki as part of the protocol transcript, and so we mix them into the chaining key to prevent tampering.
IHI5	sctr ← encaps_and_mix<SKEM>(spkr);	decaps_and_mix<SKEM>(sskr, spkr, ct1)	IHR5	Key encapsulation using the responder's public key. Mixes public key, shared secret, and ciphertext into the chaining key, and authenticates the responder.
IHI6	pidiC ← encrypt_and_mix(pidi);	spki, psk ← lookup_peer(decrypt_and_mix(pidiC))	IHR6	Tell the responder who the initiator is by transmitting the peer ID.
IHI7	mix(spki, psk);	mix(spki, psk);	IHR7	Ensure the responder has the correct view on spki. Mix in the PSK as optional static symmetric key, with epki and spkr serving as nonces.
IHI8	auth ← encrypt_and_mix(empty())	decrypt_and_mix(auth)	IHR8	Add a message authentication code to ensure both participants agree on the session state and protocol transcript at this point.

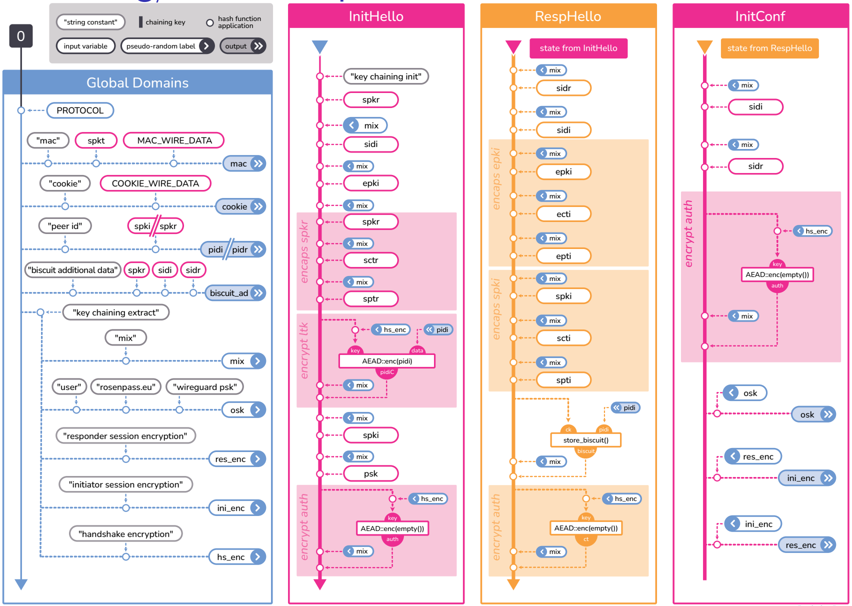
4 RespHello { sidr, sidi, ecti, scti, biscuit, auth } 3

Line	Variables ← Action	Variables ← Action	Line	Comment
RHI1		sidr ← random_session_id()	RHR1	Responder generates a session ID.
RHI2	ck ← lookup_session(sidi);		RHR2	Initiator looks up their session state using the session ID they generated.
RHI3	mix(sidr, sidi);	mix(sidr, sidi);	RHR3	Mix both session IDs as part of the protocol transcript.
RHI4	decaps_and_mix<EKEM>(eski, epki, ecti);	ecti ← encaps_and_mix<EKEM>(epki);	RHR4	Key encapsulation using the ephemeral key, to provide forward secrecy.
RHI5	decaps_and_mix<SKEM>(sski, spki, scti);	scti ← encaps_and_mix<SKEM>(spki);	RHR5	Key encapsulation using the initiator's static key, to authenticate the initiator, and non-forward-secret confidentiality.
RHI6	mix(biscuit)	biscuit ← store_biscuit();	RHR6	The responder transmits their state to the initiator in an encrypted container to avoid having to store state.
RHI7	decrypt_and_mix(auth)	auth ← encrypt_and_mix(empty());	RHR7	Add a message authentication code for the same reason as above.

5 InitConf { sidi, sidr, biscuit, auth } 6

Line	Variables ← Action	Variables ← Action	Line	Comment
ICI1		biscuit_no ← load_biscuit(biscuit);	ICR1	Responder loads their biscuit. This restores the state from after RHR6.
ICI2		encrypt_and_mix(empty());	ICR2	Responder recomputes RHR7, since this step was performed after biscuit encoding.
ICI3	mix(sidi, sidr);	mix(sidi, sidr);	ICR3	Mix both session IDs as part of the protocol transcript.
ICI4	auth ← encrypt_and_mix(empty);	decrypt_and_mix(auth);	ICR4	Message authentication code for the same reason as above, which in particular ensures that both participants agree on the final chaining key.
ICI5		assert(biscuit_no > biscuit_used);	ICR5	Biscuit replay detection.
ICI6		biscuit_used ← biscuit_no;	ICR6	Biscuit replay detection.
ICI7	enter_live();	enter_live();	ICR7	Derive the transmission keys, and the output shared key for use as WireGuard's PSK.

New Hashing/Domain separation scheme



Reference implementation in rust, deploying post-quantum-secure WireGuard

Rosenpass

Getting started About Source Code Whitepaper Contributors Press Contact

```
rp pubkey server.rosenpass-secret server.rosenpass-public
rp pubkey client.rosenpass-secret client.rosenpass-public
```

Copy the `-public` directories to the other peers and then start the VPN. On the server:

```
sudo rp exchange server.rosenpass-secret dev rosenpass0 listen 192.168.0.1:9999 \
    peer client.rosenpass-public allowed-ips fe80::/64
```

On the client:

```
sudo rp exchange client.rosenpass-secret dev rosenpass0 \
    peer server.rosenpass-public endpoint 192.168.0.1:9999 allowed-ips fe80::/64
```

Assign IP addresses:

```
sudo ip a add fe80::1/64 dev rosenpass0 # Server
sudo ip a add fe80::2/64 dev rosenpass0 # Client
```

Test the connection by pinging the server on the client machine:

```
ping fe80::1%rosenpass0 # Client
```

You can watch how Rosenpass replaces the WireGuard PSK with the following command:

```
watch -n 0.2 'wg show all; wg show all preshared-keys'
```