Easterhegg20
2023-04-09

Ein VPN zum Schutz vor Quantencomputern

Wanja Zaeske, Stephan Ajuvo, Marei Peischl, Benjamin Lipp, Lisa Schmidt, Karolin Varner

Danke an NLNet für die finanzielle Unterstützung!

https://rosenpass.eu

Was passiert im Talk?

- Was ist Rosenpass?
- Wozu braucht es Post-Quanten-Kryptographie?
- Rosenpass-Demo!
- Wie funktioniert Rosenpass?
- Bunte Checkmarks: Kryptobeweise im CI-Setup
- Integration mit WireGuard & Chiffren

Rosenpass ist...

Software

- Ein Add-On für WireGuard um Post-Quanten-Sicherheit zu ermöglichen
- Eine stand-alone Schlüsselaustausch-Applikation, die mit allen möglichen Systemen integriert werden kann

Kryptographie

- Ein kryptographisches Protokoll
- Ein Schlüsseltauschverfahren
- post-quanten-sicher
- formal verifiziert

Kommunikation und Projekt

- Eine WissKomm-Initiative um Kryptographie der breiten Öffentlichkeit zugänglich zu machen
- Community-nahe Initiative um Forschung zu ermöglichen

Angriffe von Quantencomputern: Shors[4] Algorithmus

Jargon: Löst einige mathematische Probleme effizient, auf denen moderne Krypto basiert:

- RSA[1] (das *Faktorisierungsproblem* – Primzahlzerlegung)
- DH[2] (Berechnen des *Diskreten Logarithmus*)
- ECDH[3] (Berechnen des *Diskreten Logarithmus auf Elliptischen Kurven*)

Weniger Jargon: Bricht so ziemlich alle moderne, asymmetrische Kryptographie.

[1] „Rivest-Shamir-Adleman“ – Ron Rivest, Adi Shamir, Leonard Adleman
[2] „Diffie-Hellmann“ – Whitfield Diffie, Martin Hellmann
[3] Elliptic Curve Diffie-Hellmann
[4] Peter Shor

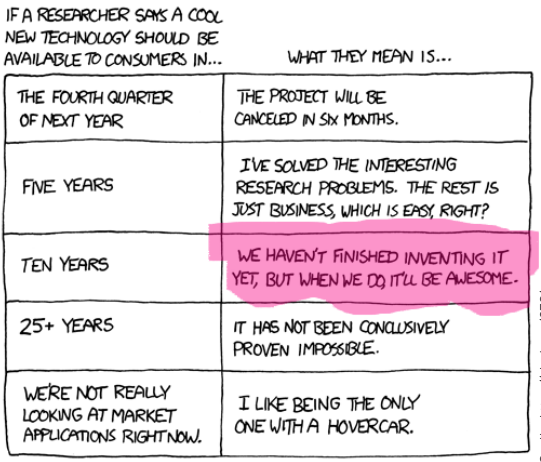
Angriffe von Quantencomputern: Grovers[5] Algorithmus

Jargon: Suche durch ungeordnete Listen in $O(\sqrt{n})$ statt klassisch $O(n)$ im Durchschnitt.

Weniger Jargon: Mostly harmless („im wesentlichen harmlos“); symmetrische Kryptographie ist kaum betroffen.

[5] Lov Grover

Quantencomputer: Ein ganz heißes Eisen



IF A RESEARCHER SAYS A COOL NEW TECHNOLOGY SHOULD BE AVAILABLE TO CONSUMERS IN...	WHAT THEY MEAN IS...
THE FOURTH QUARTER OF NEXT YEAR	THE PROJECT WILL BE CANCELED IN SIX MONTHS.
FIVE YEARS	I'VE SOLVED THE INTERESTING RESEARCH PROBLEMS. THE REST IS JUST BUSINESS, WHICH IS EASY, RIGHT?
TEN YEARS	WE HAVEN'T FINISHED INVENTING IT YET, BUT WHEN WE DO, IT'LL BE AWESOME.
25+ YEARS	IT HAS NOT BEEN CONCLUSIVELY PROVEN IMPOSSIBLE.
WE'RE NOT REALLY LOOKING AT MARKET APPLICATIONS RIGHT NOW.	I LIKE BEING THE ONLY ONE WITH A HOVERCAR.

Quelle: https://xkcd.com/678/

Post-Quanten-Kryptographie: Munch now decrypt later

- Post-Quanten-Kryptographie ist auf dem Weg der Standardisierung
- Wir müssen sehr früh deployen;
  wenn die Krypto kaputt ist, dann ist es zu spät.

Quelle: https://foto.wuestenigel.com/gray-hamster-eating-sunflower-seed/

„Munch now decrypt later“[6]

[6] „Jetzt speichern später entschlüsseln“. Warnung: Geheimdienste sind nicht so cute wie dieser Hamster.

Post-Quanten-Kryptographie: Wird bereits standardisiert

Durch NIST[7] zur Standardisierung ausgewählt [1]:

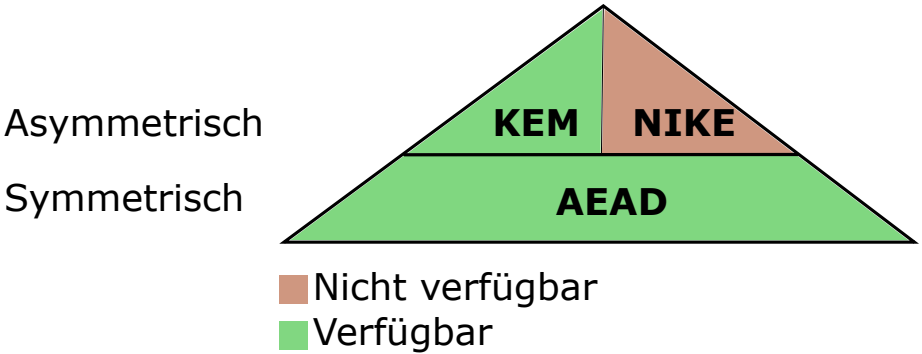
- Crystals-Kyber (Verschlüsselung)
- Crystals-Dilithium (Signatur)
- Falcon (Signatur)
- Sphincs+ (Signatur)

Das BSI[8] empfiehlt [2]:

- Frodo (Verschlüsselung)
- Classic McEliece (Verschlüsselung)

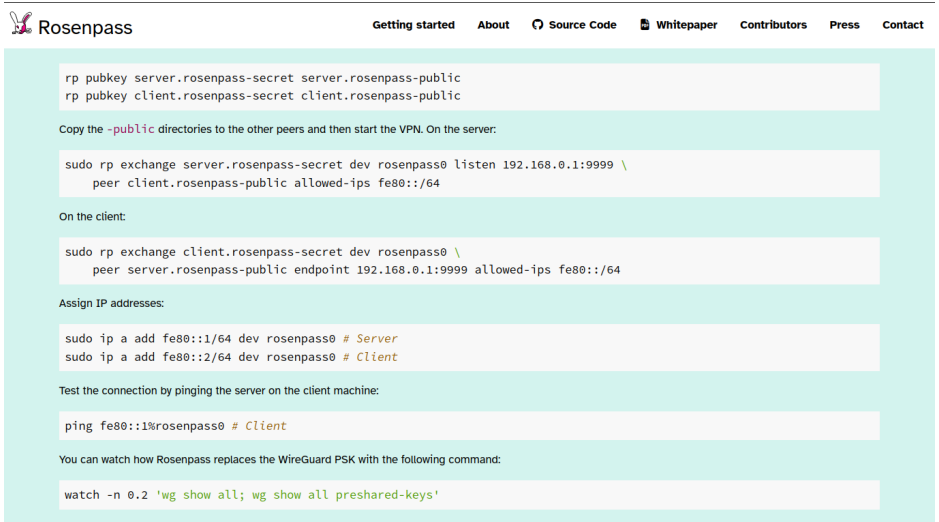
[7] National Institut for Standards and Technology – US-Amerikanische Standardisierungsbehörde
[8] Bundesamt für Sicherheit in der Informationstechnik

Verschlüsselung im Angesicht von Quantencomputern



Die meisten Schlüsselaustausch-Protokolle inklusive WireGuard nutzen NIKEs

Rosenpass Demo!



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
```

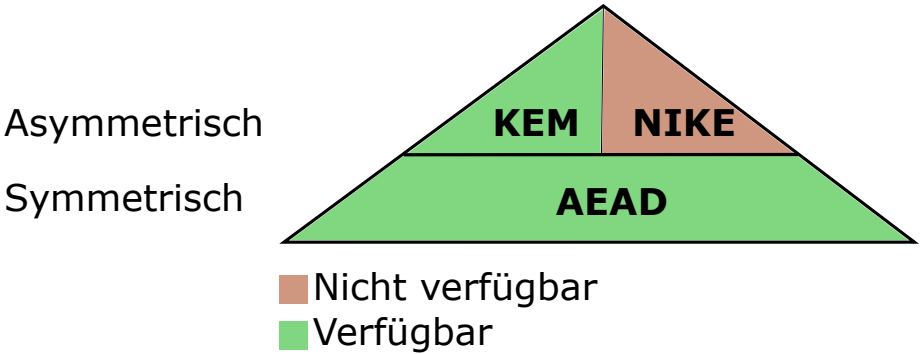
Test the connection by pinging the server on the client machine:

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

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

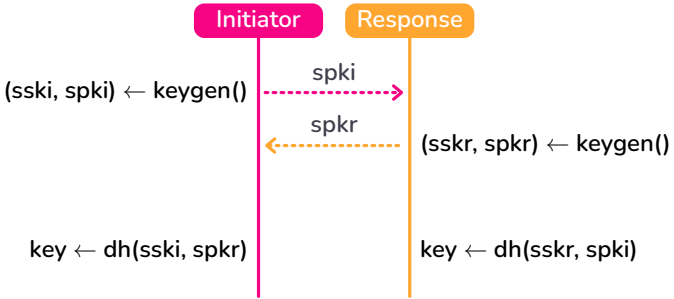
```
watch -n 0.2 'wg show all; wg show all preshared-keys'
```

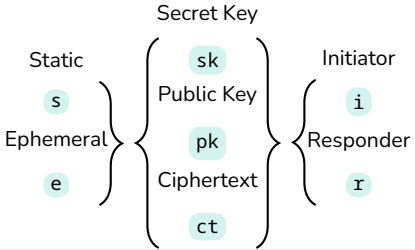
Verschlüsselung im Angesicht von Quantencomputern



Die meisten Schlüsselaustausch-Protokolle inklusive WireGuard nutzen NIKEs

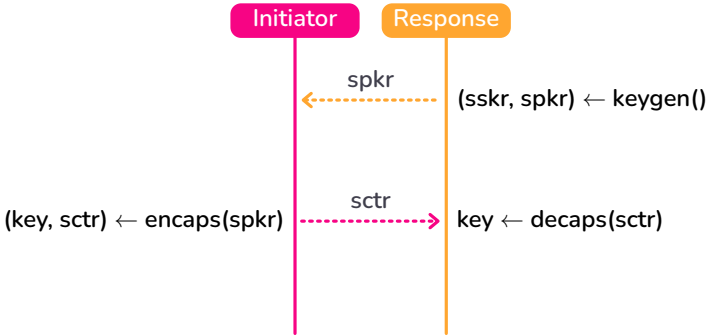
Schlüsselaustauschmethoden: Static-static Schlüsselaustausch mit NIKEs[9]

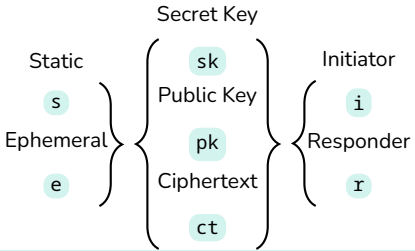




[9] „Non-Interactive Key Exchange“ – Nichtinteraktiver Schlüsselaustausch

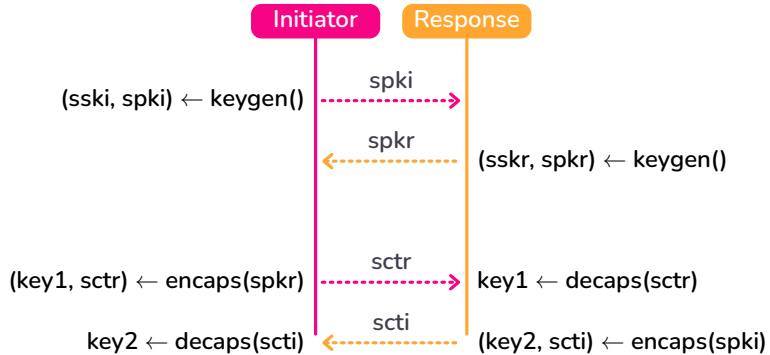
Einfachst-möglicher Schlüsseltausch mit KEMs[10]





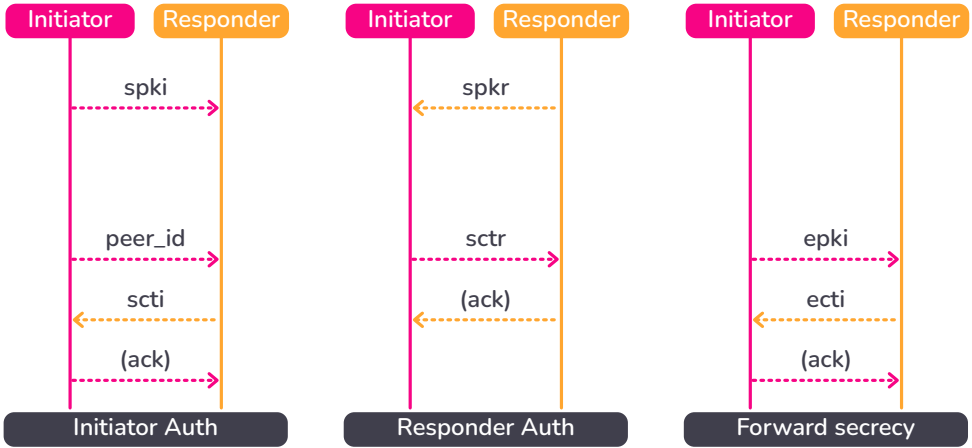
[10] „Key-Encapsulation Method“ – Schlüsseltransportmethode

Schlüsselaustauschmethoden: Mit KEMs wird es komplizierter

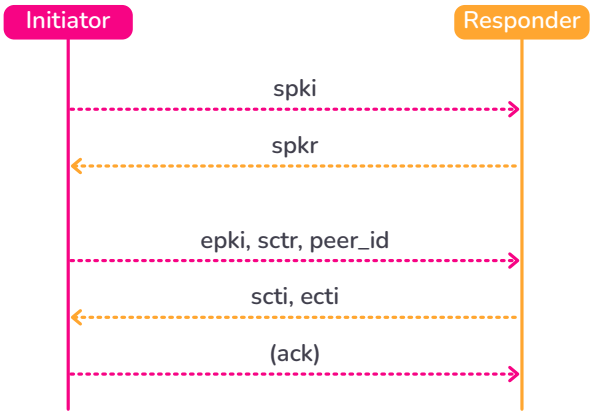


Static-static Schlüsselaustausch mit KEMs.

Post-Quanten-WireGuard: 3 Schlüsseltransporte [5]

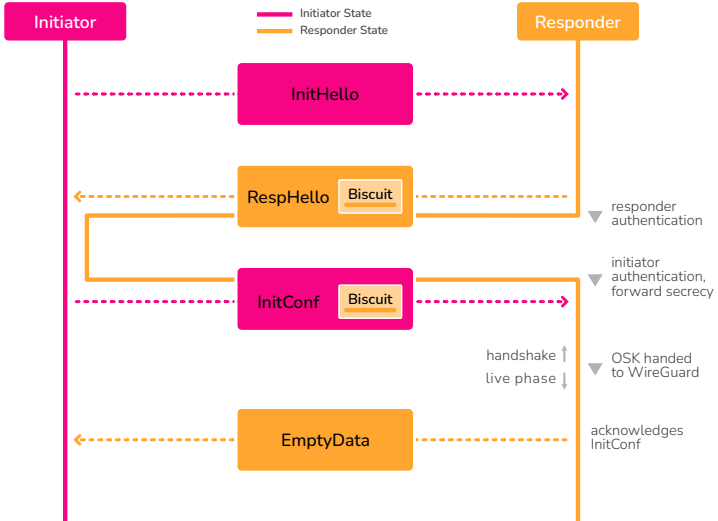


Alle 3 Schlüsseltransporte in einem Protokoll



Der Initiator ist erst authentifiziert, nachdem „(ack)“ empfangen wurde.

Das Rosenpass-Protokoll



Sicherheitsanalyse

Symbolische Protokoll-Analyse

- kann automatisiert logische Fehler im Protokoll finden.
- Genauer: Kommunikationsabläufe, die Sicherheitseigenschaften brechen

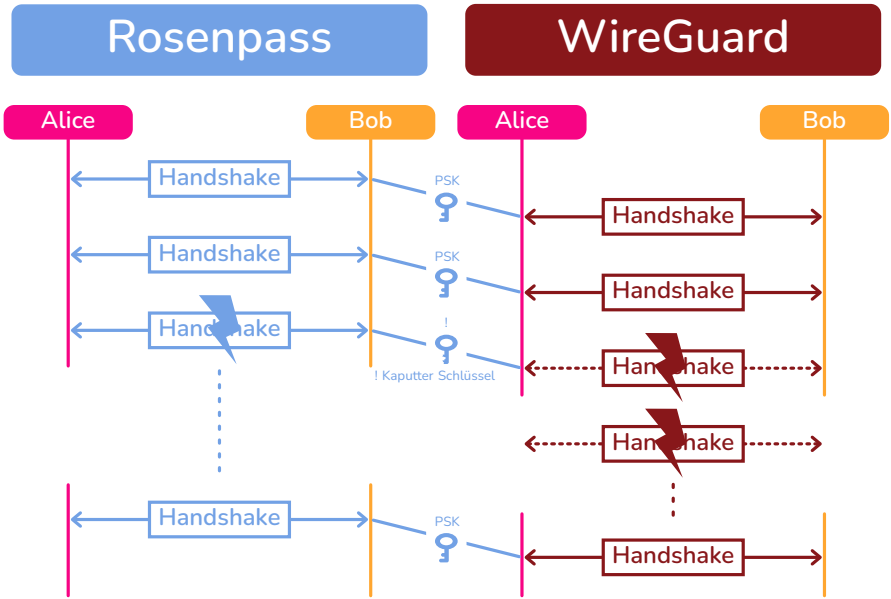
In unserem Fall:

- Wir nutzen ProVerif [3] als Tool um Protokoll-Bugs auszuschließen
- Wir haben die Laufzeit optimiert; symbolische Analyse läuft in fünf Minuten
- Beweise sind Teil des Software-Repositories; laufen in der CI

Wir arbeiten an Beweisen in einem stärkeren Angreifermodell: kryptographische Beweise (mit CryptoVerif [4])

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
```


Rosenpass
WireGuard
Alice
Bob
Alice
Bob
Handshake
PSK
Handshake
Handshake
PSK
Handshake
!
Handshake
! Kaputter Schlüssel
Handshake
Handshake
Handshake
PSK
Handshake

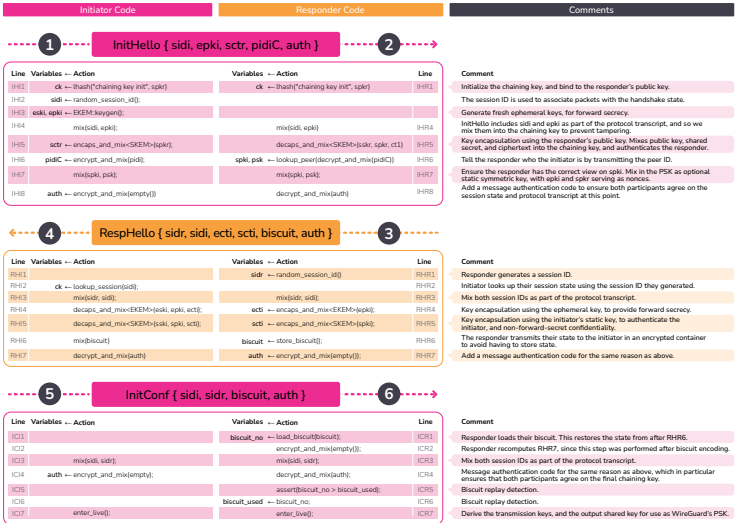
Verwendete Chiffren

- Authentifikation und Vertraulichkeit: **Classic McEliece**
  (erfunden 1978, codebasiert)
- Forward Secrecy: **Kyber**
  (von NIST zur Standardisierung ausgewählt, gitterbasiert)
- Kryptoagilität: Wir planen die Möglichkeit einzubauen, die Chiffren zu wechseln
  (das ist *nicht* ciphersuite negotiation)

Ausblick

- Rosenpass in Kubernetes
- Isolation, Micro-VMs, Docker
- Formal verifizierte Implementierung
- Mehr WissKomm zu Kryptographie. Kryptographie braucht verständliche Erklärungen!
- Wir suchen High-Assurance-Kryptographieprojekte um mit uns zusammenzuarbeiten. Rosenpass ist klein und kann als Demonstrator dienen.

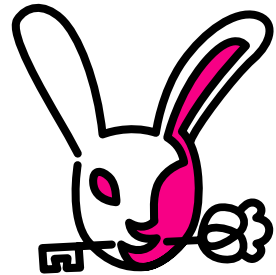
Zum Nachbauen… aus dem Whitepaper:



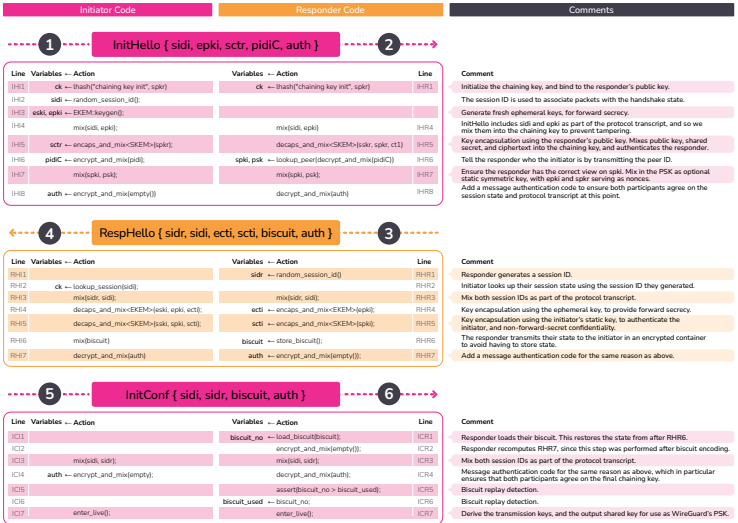
Initiator Code			Responder Code			Comments
		1 InitHello { sidi, epki, sctr, pidiC, auth } 2				
Line	Variables	← Action	Variables	← Action	Line	Comment
IHI1	ck	← thash("chaining key init", spkr)	ck	← thash("chaining key init", spkr)	IHR1	Initialize the chaining key, and bind to the responder's public 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, pskr	← lookup_peer(decrypt_and_mix(pidiC))	IHR6	Tell the responder who the initiator is by transmitting the peer ID.
IHI7		mix(spki, pskr);		mix(spki, pskr);	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.

Konversationsstarter

- Zurzeit wird Rosenpass via CLI konfiguriert
- Das lässt zu wünschen:
  - hinzufügen/entfernen von peers **ohne** neustart
  - <23 CLI Argumente für exchange mit einem peer
  - simple integration mit anderen Programmiersprachen
- Lösungsansätze:
  - Konfigurationsdatei: `rosenpass rp-config.toml`
  - Unix Domain Socket: `add peer /opt/peer-pub.key rosenpass.eu:9999`


ROSENPASS

Zum Nachbauen… aus dem Whitepaper:



Initiator Code			Responder Code			Comments
		1 InitHello { sidi, epki, sctr, pidiC, auth } 2				
Line	Variables	← Action	Variables	← Action	Line	Comment
IH1	ck	← thash("chaining key init", spkr)	ck	← thash("chaining key init", spkr)	IHR1	Initialize the chaining key, and bind to the responder's public key.
IH2	sidi	← random_session_id();				The session ID is used to associate packets with the handshake state.
IH3	eski, epki	← EKEM.keygen();				Generate fresh ephemeral keys, for forward secrecy.
IH4		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.
IH5	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.
IH6	pidiC	← encrypt_and_mix(pidi);	spki, pskr	← lookup_peer(decrypt_and_mix(pidiC))	IHR6	Tell the responder who the initiator is by transmitting the peer ID.
IH7		mix(spki, pskr);		mix(spki, pskr);	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.
IH8	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
RH1			sidr	← random_session_id()	RHR1	Responder generates a session ID.
RH2	ck	← lookup_session(sidi);			RHR2	Initiator looks up their session state using the session ID they generated.
RH3		mix(sidr, sidi);		mix(sidr, sidi);	RHR3	Mix both session IDs as part of the protocol transcript.
RH4		decaps_and_mix<EKEM>(eski, epki, ecti);	ecti	← encaps_and_mix<EKEM>(epki);	RHR4	Key encapsulation using the ephemeral key, to provide forward secrecy.
RH5		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.
RH6		mix(biscuit)	biscuit	← store_biscuit();	RHR6	The responder transmits their state to the initiator in an encrypted container to avoid having to store state.
RH7		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
IC1			biscuit_no	← load_biscuit(biscuit);	ICR1	Responder loads their biscuit. This restores the state from after RHR6.
IC2				encrypt_and_mix(empty());	ICR2	Responder recomputes RHR7, since this step was performed after biscuit encoding.
IC3		mix(sidi, sidr);		mix(sidi, sidr);	ICR3	Mix both session IDs as part of the protocol transcript.
IC4	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.
IC5				assert(biscuit_no > biscuit_used);	ICR5	Biscuit replay detection.
IC6			biscuit_used	← biscuit_no;	ICR6	Biscuit replay detection.
IC7		enter_live();		enter_live();	ICR7	Derive the transmission keys, and the output shared key for use as WireGuard's PSK.

Sicherheitsanalyse

Symbolische Protokoll-Analyse

- kann automatisiert logische Fehler im Protokoll finden.
- Genauer: Kommunikationsabläufe, die Sicherheitseigenschaften brechen

In unserem Fall:

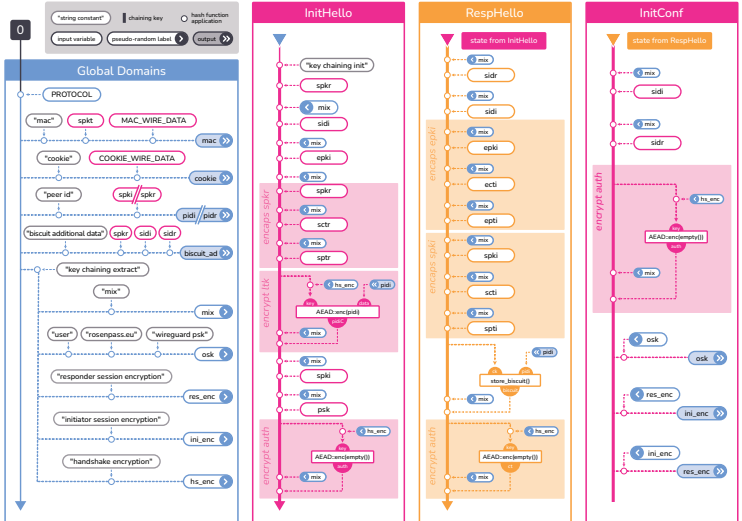
- Wir nutzen ProVerif [3] als Tool um Protokoll-Bugs auszuschließen
- Wir haben die Laufzeit optimiert; symbolische Analyse läuft in fünf Minuten
- Beweise sind Teil des Software-Repositories; laufen in der CI

Wir arbeiten an Beweisen in einem stärkeren Angreifermodell: kryptographische Beweise (mit CryptoVerif [4])

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

New Hashing/Domain separation scheme



[1] URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[2] URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Broschueren/Kryptografie-quantensicher-gestalten.pdf.

[3] URL: https://proverif.inria.fr/.

[4] URL: https://cryptoverif.inria.fr/.

[5] Andreas Hülsing u. a. „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. IEEE, 2021, S. 304–321. DOI: 10.1109/SP40001.2021.00030. URL: https://doi.org/10.1109/SP40001.2021.00030.