Post-quantum WireGuard

Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Florian Weber, Philip R. Zimmermann[a]

October 25, 2021

[a] authors in alphabetic order

WireGuard

- Modern Virtual Private Network (VPN) protocol
- Presented by Donenfeld at NDSS 2017
- Merged into Linux kernel in 2020
- Only ≈4000 lines of code
- Runs over UDP

> *”Compared to horrors that are OpenVPN and IPSec, WireGuard is a work of art”*
>
> —Linus Torvalds

“Cryptographically opinionated”

- No “crypto agility”
- Fixed suite of cryptographic primitives:
  - X25519 as Diffie-Hellman routine
  - ChaCha20-Poly1305 as AEAD
  - Blake2s for hashing and keyed hashing
  - HKDF for key derivation

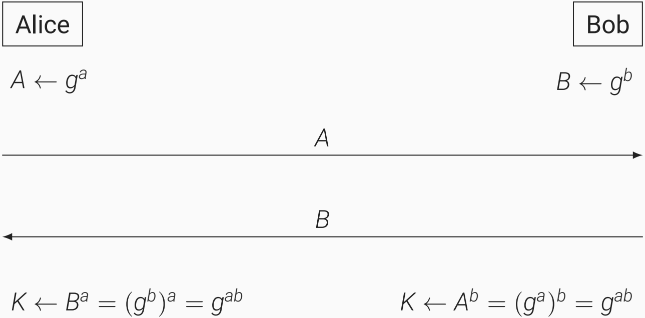
“Cryptographically opinionated”

- No “crypto agility”
- Fixed suite of cryptographic primitives:
  - X25519 as Diffie-Hellman routine
  - ChaCha20-Poly1305 as AEAD
  - Blake2s for hashing and keyed hashing
  - HKDF for key derivation
- Focus today: the WireGuard **handshake**
  - Authenticate parties to each other
  - Establish a session key to encrypt payload data

“Cryptographically opinionated”

- No “crypto agility”
- Fixed suite of cryptographic primitives:
  - X25519 as Diffie-Hellman routine
  - ChaCha20-Poly1305 as AEAD
  - Blake2s for hashing and keyed hashing
  - HKDF for key derivation
- Focus today: the WireGuard **handshake**
  - Authenticate parties to each other
  - Establish a session key to encrypt payload data

Diffie-Hellman reminder



Alice

Bob

$A \leftarrow g^a$

$B \leftarrow g^b$

A

B

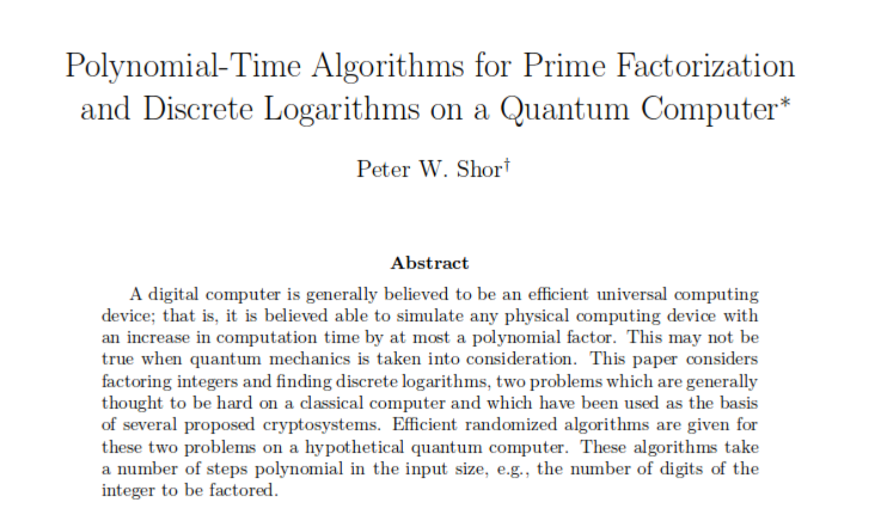
$K \leftarrow B^a = (g^b)^a = g^{ab}$

$K \leftarrow A^b = (g^a)^b = g^{ab}$

Diffie-Hellman reminder

Attacker who can compute *x* given g^x breaks Diffie-Hellman

This is known as **Discrete-Logarithm Problem**



Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer[*]

Peter W. Shor[†]

Abstract

A digital computer is generally believed to be an efficient universal computing device; that is, it is believed able to simulate any physical computing device with an increase in computation time by at most a polynomial factor. This may not be true when quantum mechanics is taken into consideration. This paper considers factoring integers and finding discrete logarithms, two problems which are generally thought to be hard on a classical computer and which have been used as the basis of several proposed cryptosystems. Efficient randomized algorithms are given for these two problems on a hypothetical quantum computer. These algorithms take a number of steps polynomial in the input size, e.g., the number of digits of the integer to be factored.

> *“In the past, people have said, maybe it’s 50 years away, it’s a dream, maybe it’ll happen sometime. I used to think it was 50. Now I’m thinking like it’s 15 or a little more. It’s within reach. It’s within our lifetime. It’s going to happen.”*
>
> —Mark Ketchen (IBM), Feb. 2012, about quantum computers

Post-quantum crypto

Definition

Post-quantum crypto is (asymmetric) crypto that resists attacks using classical *and quantum* computers.

Post-quantum crypto

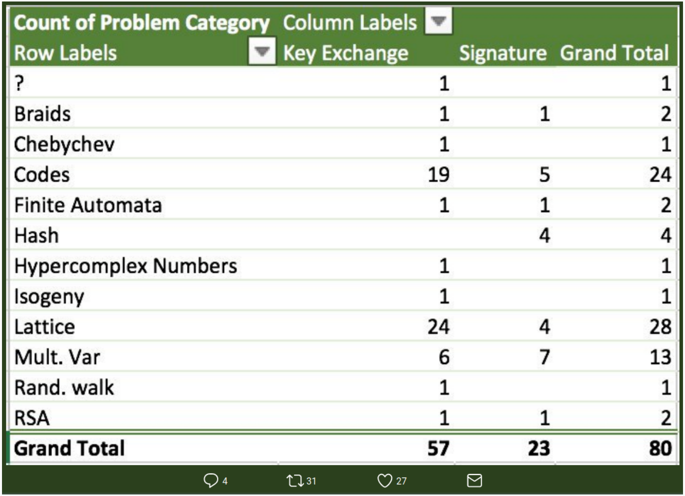
Definition

Post-quantum crypto is (asymmetric) crypto that resists attacks using classical *and quantum* computers.

5 main directions

- Lattice-based crypto (PKE and Sigs)
- Code-based crypto (mainly PKE)
- Multivariate-based crypto (mainly Sigs)
- Hash-based signatures (only Sigs)
- Isogeny-based crypto (so far, mainly PKE)

NISTPQC



Count of Problem Category	Column Labels		
Row Labels	Key Exchange	Signature	Grand Total
?	1		1
Braids	1	1	2
Chebychev	1		1
Codes	19	5	24
Finite Automata	1	1	2
Hash		4	4
Hypercomplex Numbers	1		1
Isogeny	1		1
Lattice	24	4	28
Mult. Var	6	7	13
Rand. walk	1		1
RSA	1	1	2
Grand Total	**57**	**23**	**80**

Overview tweeted by Jacob Alperin-Sheriff on Dec 4, 2017.

State of NIST PQC

- Nov 2017: 69 “complete and proper” submissions

State of NIST PQC

- Nov 2017: 69 “complete and proper” submissions
- Jan. 2019: Round-2 announcement planned at RWC

State of NIST PQC

- Nov 2017: 69 “complete and proper” submissions
- Jan. 2019: Round-2 announcement planned at RWC
- Feb. 2019: 26 round-2 candidates

State of NIST PQC

- Nov 2017: 69 “complete and proper” submissions
- Jan. 2019: Round-2 announcement planned at RWC
- Feb. 2019: 26 round-2 candidates
- Jun. 2020: Round-3 announcement planned

State of NIST PQC

- Nov 2017: 69 “complete and proper” submissions
- Jan. 2019: Round-2 announcement planned at RWC
- Feb. 2019: 26 round-2 candidates
- Jun. 2020: Round-3 announcement planned
- Jul. 2020: Round-3 announcement:
  - 7 finalists
  - 8 alternate schemes

What now?

- NIST is expected to announce winners in late 2021
- $\approx$ one year later get standards

What now?

- NIST is expected to announce winners in late 2021
- ≈ one year later get standards
- Replace existing crypto with new crypto

What now?

- NIST is expected to announce winners in late 2021
- $\approx$ one year later get standards
- Replace existing crypto with new crypto

Mission accomplished – The world is safe again!

What now?

- NIST is expected to announce winners in late 2021
- ≈ one year later get standards
- Replace existing crypto with new crypto

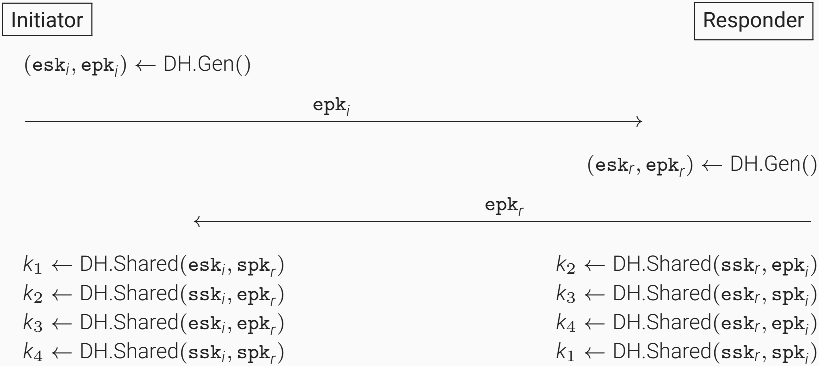
Mission accomplished – The world is safe again!

. . . or is it?

The WireGuard handshake (basic idea: “4DH”)

Initiator has long-term DH key-pair $(\mathtt{ssk}_i, \mathtt{spk}_i)$

Responder has long-term DH key-pair $(\mathtt{ssk}_r, \mathtt{spk}_r)$



Initiator

Responder

$(\mathtt{esk}_i, \mathtt{epk}_i) \leftarrow$ DH.Gen()

$\xrightarrow{\mathtt{epk}_i}$

$(\mathtt{esk}_r, \mathtt{epk}_r) \leftarrow$ DH.Gen()

$\xleftarrow{\mathtt{epk}_r}$

Initiator	Responder
$k_1 \leftarrow \text{DH.Shared}(\mathtt{esk}_i, \mathtt{spk}_r)$	$k_2 \leftarrow \text{DH.Shared}(\mathtt{ssk}_r, \mathtt{epk}_i)$
$k_2 \leftarrow \text{DH.Shared}(\mathtt{ssk}_i, \mathtt{epk}_r)$	$k_3 \leftarrow \text{DH.Shared}(\mathtt{esk}_r, \mathtt{spk}_i)$
$k_3 \leftarrow \text{DH.Shared}(\mathtt{esk}_i, \mathtt{epk}_r)$	$k_4 \leftarrow \text{DH.Shared}(\mathtt{esk}_r, \mathtt{epk}_i)$
$k_4 \leftarrow \text{DH.Shared}(\mathtt{ssk}_i, \mathtt{spk}_r)$	$k_1 \leftarrow \text{DH.Shared}(\mathtt{ssk}_r, \mathtt{spk}_i)$

Derive session key from k_1, k_2, k_3, and k_4

The WireGuard handshake (high-level)

Initiator

Responder

```
1: $(\mathtt{esk}_i, \mathtt{epk}_i) \leftarrow \mathsf{DH.Gen}()$
2: $\mathtt{sid}_i \xleftarrow{\$} \{0,1\}^{32}$
3: $\mathtt{ltk} \leftarrow \mathsf{AEAD.Enc}(\kappa_3, 0, \mathtt{spk}_i, H_3)$
4: $\mathit{now} \leftarrow \mathsf{Timestamp}()$
5: $\mathtt{time} \leftarrow \mathsf{AEAD.Enc}(\kappa_4, 0, H_4, \mathit{now})$
6: $\mathtt{m1} \leftarrow \mathsf{MAC}(\mathsf{H}(\mathtt{lbl}_3 \| \mathtt{spk}_r), \mathtt{type} \| 0^3 \| \mathtt{sid}_i \| \mathtt{epk}_i \| \mathtt{ltk} \| \mathtt{time})$
7: $\mathtt{m2} \leftarrow \mathsf{MAC}(\mathit{cookie}, \mathtt{type} \| 0^3 \| \mathtt{sid}_i \| \mathtt{epk}_i \| \mathtt{ltk} \| \mathtt{time} \| \mathtt{m1})$
8: $\mathtt{InitHello} \leftarrow \mathtt{type} \| 0^3 \| \mathtt{sid}_i \| \mathtt{epk}_i \| \mathtt{ltk} \| \mathtt{time} \| \mathtt{m1} \| \mathtt{m2}$
```

$$\xrightarrow{\quad\quad\quad\quad\quad\quad \mathtt{InitHello} \quad\quad\quad\quad\quad\quad}$$

```
9:                               $(\mathtt{esk}_r, \mathtt{epk}_r) \leftarrow \mathsf{DH.Gen}()$
10:                              $\mathtt{sid}_r \xleftarrow{\$} \{0,1\}^{32}$
11:                              $\mathtt{zero} \leftarrow \mathsf{AEAD.Enc}(\kappa_9, 0, H_9, \emptyset)$
12:                              $\mathtt{m1} \leftarrow \mathsf{MAC}(\mathsf{H}(\mathtt{lbl}_3 \| \mathtt{spk}_i), \mathtt{type} \| 0^3 \| \mathtt{sid}_r \| \mathtt{sid}_i \| \mathtt{epk}_r \| \mathtt{zero})$
13:                              $\mathtt{m2} \leftarrow \mathsf{MAC}(\mathit{cookie}, \mathtt{type} \| 0^3 \| \mathtt{sid}_r \| \mathtt{sid}_i \| \mathtt{epk}_r \| \mathtt{zero} \| \mathtt{m1})$
14:                              $\mathtt{RespHello} \leftarrow \mathtt{type} \| 0^3 \| \mathtt{sid}_r \| \mathtt{sid}_i \| \mathtt{epk}_r \| \mathtt{zero} \| \mathtt{m1} \| \mathtt{m2}$
```

$$\xleftarrow{\quad\quad\quad\quad\quad\quad \mathtt{RespHello} \quad\quad\quad\quad\quad\quad}$$

```
15:              $\mathit{tk}_i \leftarrow \mathsf{KDF}_1(C_9, \emptyset)$
16:              $\mathit{tk}_r \leftarrow \mathsf{KDF}_2(C_9, \emptyset)$
```

$$\xrightarrow{\quad\quad \mathsf{AEAD.Enc}(\mathit{tk}_i, \cdot, \emptyset, \text{application data}) \quad\quad}$$

Handshake security

- Key confidentiality
- Entity authentication

Handshake security

- Key confidentiality
- Entity authentication
- Key uniqueness
- Identity hiding
- Replay attack resistance
- Unknown key-share (UKS) attack resistance
- DoS attack resistance (early reject)

WireGuard security proofs

- Computational: Dowling and Paterson, 2018
  - eCK-PFS-PSK
  - Assumes additional key-confirmation message
  - Missing: key uniqueness, identity hiding, DoS mitigation
- Symbolic: partially by Donenfeld and Milner, 2017
  - Missing: perfect forward secrecy, replay attack resistance, DoS mitigation

Post-quantum security of WireGuard

- The optional PSK provides confidentiality against quantum attacks.
- Assumption: PSK cannot be recovered by quantum attackers
- Post-quantum cryptography: Donenfeld claimed ’not practical for use here’
- Applebaum, Martindale, Wu, 2019:
  - Tweak to WireGuard protocol
  - Send H(pk) instead of pk
  - Quantum attacker does not easily get pk
  - Resistance against mass-surveillance attackers

PQ-WireGuard – our goals

- Post-quantum confidentiality **and authentication**
- NIST security level 3 (≈AES-192)
- Retain all security properties of WireGuard
- Efficient 1-round-trip handshake

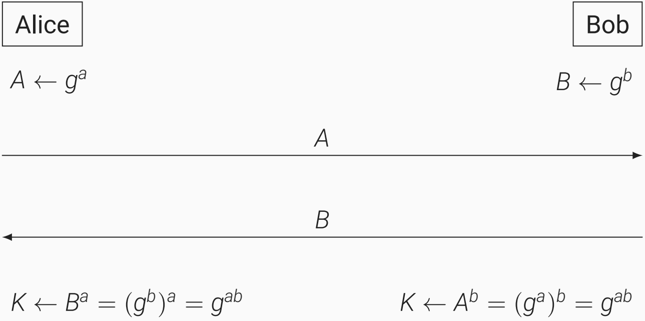
PQ-WireGuard – our goals

- Post-quantum confidentiality **and authentication**
- NIST security level 3 (≈AES-192)
- Retain all security properties of WireGuard
- Efficient 1-round-trip handshake
- No fragmentation
  - Remember: WireGuard uses UDP
  - Lost packets, filtering ⇒ more complex state machine
- Packet-size constraint:
  - IPv6 guarantee: no fragmentation of packets ≤ 1280 bytes
  - Fit WireGuard messages into 1232 bytes

PQ-WireGuard – the idea

1. Replace DH with key-encapsulation mechanisms (KEMs)
2. Instantiate with PQ KEMs achieving desired security

Diffie-Hellman



Alice

Bob

$A \leftarrow g^a$

$B \leftarrow g^b$

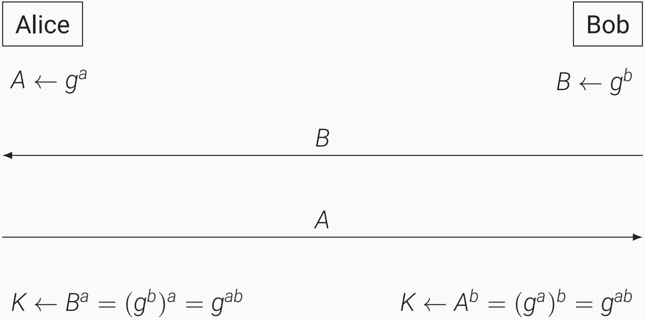
A

B

$K \leftarrow B^a = (g^b)^a = g^{ab}$

$K \leftarrow A^b = (g^a)^b = g^{ab}$

Diffie-Hellman



Alice

Bob

$A \leftarrow g^a$

$B \leftarrow g^b$

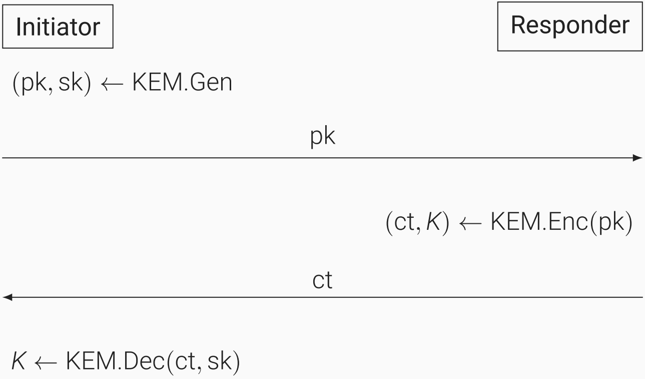
B

A

$K \leftarrow B^a = (g^b)^a = g^{ab}$

$K \leftarrow A^b = (g^a)^b = g^{ab}$

KEMs: as close as you'll get to DH



Initiator

Responder

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KEM.Gen}$

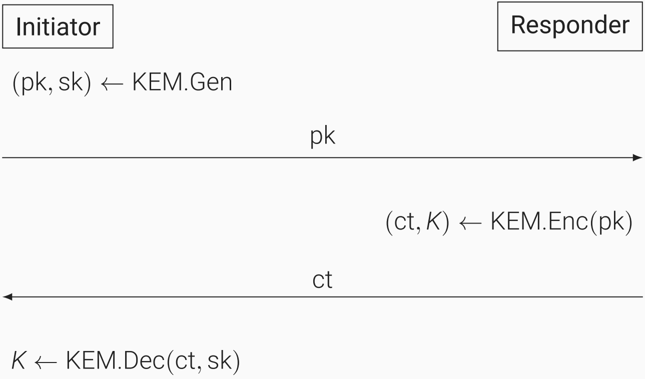
pk

$(\mathsf{ct}, K) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk})$

ct

$K \leftarrow \mathsf{KEM.Dec}(\mathsf{ct}, \mathsf{sk})$

KEMs: as close as you'll get to DH*



Initiator

Responder

$(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{KEM.Gen}$

pk

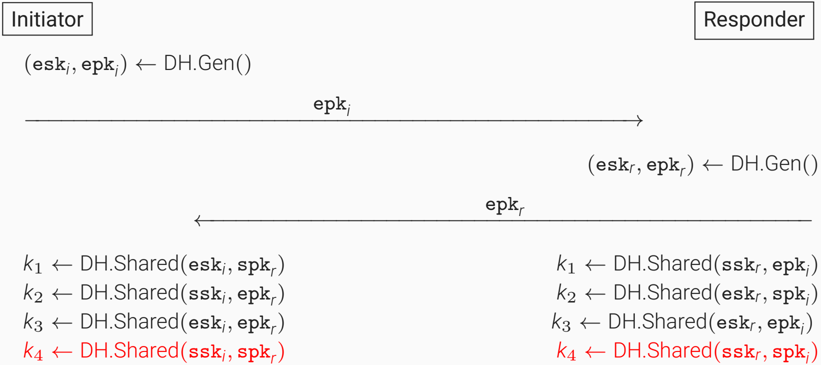
$(\mathrm{ct}, K) \leftarrow \mathrm{KEM.Enc}(\mathrm{pk})$

ct

$K \leftarrow \mathrm{KEM.Dec}(\mathrm{ct}, \mathrm{sk})$

*Except with CSIDH (Castryck, Lange, Martindale, Renes, Panny, 2018)

What can KEMs (not) do?



Initiator

Responder

$(\mathtt{esk}_i, \mathtt{epk}_i) \leftarrow \mathsf{DH.Gen}()$

$$\xrightarrow{\mathtt{epk}_i}$$

$(\mathtt{esk}_r, \mathtt{epk}_r) \leftarrow \mathsf{DH.Gen}()$

$$\xleftarrow{\mathtt{epk}_r}$$

$k_1 \leftarrow \mathsf{DH.Shared}(\mathtt{esk}_i, \mathtt{spk}_r)$

$k_2 \leftarrow \mathsf{DH.Shared}(\mathtt{ssk}_i, \mathtt{epk}_r)$

$k_3 \leftarrow \mathsf{DH.Shared}(\mathtt{esk}_i, \mathtt{epk}_r)$

$k_4 \leftarrow \mathsf{DH.Shared}(\mathtt{ssk}_i, \mathtt{spk}_r)$

$k_1 \leftarrow \mathsf{DH.Shared}(\mathtt{ssk}_r, \mathtt{epk}_i)$

$k_2 \leftarrow \mathsf{DH.Shared}(\mathtt{esk}_r, \mathtt{spk}_i)$

$k_3 \leftarrow \mathsf{DH.Shared}(\mathtt{esk}_r, \mathtt{epk}_i)$

$k_4 \leftarrow \mathsf{DH.Shared}(\mathtt{ssk}_r, \mathtt{spk}_i)$

A first approach with KEMs

Initiator

Responder

$(\mathtt{esk}_i, \mathtt{epk}_i) \leftarrow \text{CPAKEM.Gen}()$

$r_1 \xleftarrow{\$} \{0,1\}^\lambda, (c_1, k_1) \leftarrow \text{CCAKEM.Enc}(\mathtt{spk}_r, r_1)$

$$\xrightarrow{\qquad\qquad\qquad \mathtt{epk}_i, c_1 \qquad\qquad\qquad}$$

$r_2 \xleftarrow{\$} \{0,1\}^\lambda, (c_2, k_2) \leftarrow \text{CCAKEM.Enc}(\mathtt{spk}_i, r_2)$

$r_3 \xleftarrow{\$} \{0,1\}^\lambda, (c_3, k_3) \leftarrow \text{CPAKEM.Enc}(\mathtt{epk}_i, r_3)$

$$\xleftarrow{\qquad\qquad\qquad c_2, c_3 \qquad\qquad\qquad}$$

$k_1 \leftarrow \text{CCAKEM.Dec}(\mathtt{ssk}_r, c_1)$

$k_2 \leftarrow \text{CCAKEM.Dec}(\mathtt{ssk}_i, c_2)$

$k_3 \leftarrow \text{CPAKEM.Dec}(\mathtt{esk}_i, c_3)$

What are we lacking?

DoS resistance

- First initiator message is unauthenticated
- Solution: Use (optional) pre-shared key for early rejects

What are we lacking?

DoS resistance

- First initiator message is unauthenticated
- Solution: Use (optional) pre-shared key for early rejects

“MEX” resistance

- Some security also if all RNGs are insecure
- Static-static DH for confidentiality from long-term keys
- Solution: Use “NAXOS trick”

What are we lacking?

DoS resistance

- First initiator message is unauthenticated
- Solution: Use (optional) pre-shared key for early rejects

“MEX” resistance

- Some security also if all RNGs are insecure
- Static-static DH for confidentiality from long-term keys
- Solution: Use “NAXOS trick”

UKS-attack resistance

- WireGuard does not hash public keys into session key
- UKS resistance derived from static-static DH
- Solution: Use default PSK as $\mathrm{H}(\mathtt{spk}_i \oplus \mathtt{spk}_r)$

(Most of) the PQ-WireGuard handshake

Initiator

Responder

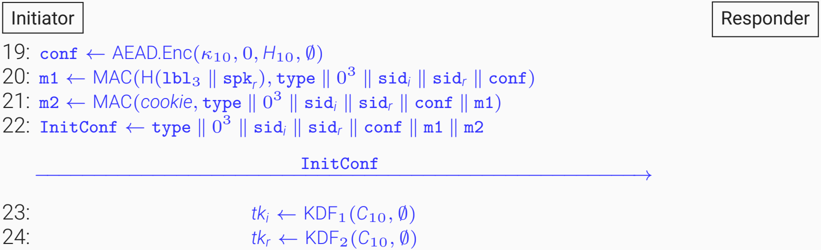
```
$(\mathtt{esk}_i, \mathtt{epk}_i) \leftarrow \text{CPAKEM.Gen}()$
$\mathtt{sid}_i \xleftarrow{\$} \{0,1\}^{32}$
$r_i \leftarrow \{0,1\}^{\lambda}$
$(\mathtt{ct1}, \mathtt{shk1}) \leftarrow \text{CCAKEM.Enc}(\mathtt{spk}_r, \text{KDF}_1(\sigma_i, r_i))$
$\mathtt{ltk} \leftarrow \text{AEAD.Enc}(\kappa_3, 0, \text{H}(\mathtt{spk}_i), H_3)$
$\mathit{now} \leftarrow \text{Timestamp}()$
$\mathtt{time} \leftarrow \text{AEAD.Enc}(\kappa_4, 0, H_4, \mathit{now})$
$\mathtt{m1} \leftarrow \text{MAC}(\text{H}(\mathtt{lbl}_3 \| \mathtt{spk}_r), \mathtt{type} \| 0^3 \| \mathtt{sid}_i \| \mathtt{epk}_i \| \mathtt{ct1} \| \mathtt{ltk} \| \mathtt{time})$
$\mathtt{m2} \leftarrow \text{MAC}(\mathit{cookie}, \mathtt{type} \| 0^3 \| \mathtt{sid}_i \| \mathtt{epk}_i \| \mathtt{ct1} \| \mathtt{ltk} \| \mathtt{time} \| \mathtt{m1})$
$\mathtt{InitHello} \leftarrow \mathtt{type} \| 0^3 \| \mathtt{sid}_i \| \mathtt{epk}_i \| \mathtt{ct1} \| \mathtt{ltk} \| \mathtt{time} \| \mathtt{m1} \| \mathtt{m2}$
```

$$\xrightarrow{\qquad\qquad\mathtt{InitHello}\qquad\qquad}$$

```
$e, r_r \leftarrow \{0,1\}^{\lambda} \times \{0,1\}^{\lambda}$
$(\mathtt{ct2}, \mathtt{shk2}) \leftarrow \text{CPAKEM.Enc}(\mathtt{epk}_i, e)$
$(\mathtt{ct3}, \mathtt{shk3}) \leftarrow \text{CCAKEM.Enc}(\mathtt{spk}_i, \text{KDF}_1(\sigma_r, r_r))$
$\mathtt{sid}_r \xleftarrow{\$} \{0,1\}^{32}$
$\mathtt{zero} \leftarrow \text{AEAD.Enc}(\kappa_9, 0, H_9, \emptyset)$
$\mathtt{m1} \leftarrow \text{MAC}(\text{H}(\mathtt{lbl}_3 \| \mathtt{spk}_i), \mathtt{type} \| 0^3 \| \mathtt{sid}_r \| \mathtt{sid}_i \| \mathtt{ct2} \| \mathtt{ct3} \| \mathtt{zero})$
$\mathtt{m2} \leftarrow \text{MAC}(\mathit{cookie}, \mathtt{type} \| 0^3 \| \mathtt{sid}_r \| \mathtt{sid}_i \| \mathtt{ct2} \| \mathtt{ct3} \| \mathtt{zero} \| \mathtt{m1})$
$\mathtt{RespHello} \leftarrow \mathtt{type} \| 0^3 \| \mathtt{sid}_r \| \mathtt{sid}_i \| \mathtt{ct2} \| \mathtt{ct3} \| \mathtt{zero} \| \mathtt{m1} \| \mathtt{m2}$
```

$$\xleftarrow{\qquad\qquad\mathtt{RespHello}\qquad\qquad}$$

Adding explicit key confirmation



Initiator

Responder

19: $\mathtt{conf} \leftarrow \mathrm{AEAD.Enc}(\kappa_{10}, 0, H_{10}, \emptyset)$

20: $\mathtt{m1} \leftarrow \mathrm{MAC}(\mathrm{H}(\mathtt{lbl}_3 \parallel \mathtt{spk}_r), \mathtt{type} \parallel 0^3 \parallel \mathtt{sid}_i \parallel \mathtt{sid}_r \parallel \mathtt{conf})$

21: $\mathtt{m2} \leftarrow \mathrm{MAC}(\mathit{cookie}, \mathtt{type} \parallel 0^3 \parallel \mathtt{sid}_i \parallel \mathtt{sid}_r \parallel \mathtt{conf} \parallel \mathtt{m1})$

22: $\mathtt{InitConf} \leftarrow \mathtt{type} \parallel 0^3 \parallel \mathtt{sid}_i \parallel \mathtt{sid}_r \parallel \mathtt{conf} \parallel \mathtt{m1} \parallel \mathtt{m2}$

$$\xrightarrow{\mathtt{InitConf}}$$

23: $tk_i \leftarrow \mathrm{KDF}_1(C_{10}, \emptyset)$

24: $tk_r \leftarrow \mathrm{KDF}_2(C_{10}, \emptyset)$

- Allows proofs to separate handshake from data transmission
- eCK-PFS-PSK proof applies to actual protocol

PQ-WireGuard security proofs

- Computational:
    - Based on Dowling and Paterson (2018)
    - Proof in the eCK-PFS-PSK model
    - Standard model proof
- Symbolic:
    - Based on Donenfeld and Milner (2017)
    - Uses the Tamarin prover
    - Cover all desired security properties

Instantiation

- Long-term IND-CCA-secure KEM: Classic McEliece
    - Smallest ciphertext of all NIST PQC candidates
    - Public-key size does not matter
    - Key-generation time does not matter

Instantiation

- Long-term IND-CCA-secure KEM: Classic McEliece
    - Smallest ciphertext of all NIST PQC candidates
    - Public-key size does not matter
    - Key-generation time does not matter
- Ephemeral IND-CPA-secure KEM requirements:
    - NIST PQC round-2 candidate at level 3
    - High-speed constant-time implementation
    - Pick “conservative” primitives
    - No patent claims by submitters
    - No tweaks that lower security

Instantiation

- Long-term IND-CCA-secure KEM: Classic McEliece
  - Smallest ciphertext of all NIST PQC candidates
  - Public-key size does not matter
  - Key-generation time does not matter
- Ephemeral IND-CPA-secure KEM requirements:
  - NIST PQC round-2 candidate at level 3
  - High-speed constant-time implementation
  - Pick “conservative” primitives
  - No patent claims by submitters
  - No tweaks that lower security
  - Fit into unfragmented IPv6 packet:
    - public key of $\leq$928 bytes
    - ciphertext of $\leq$984 bytes

Dagger

- Only three NIST round-2 candidates within size constraints:
  - SIKE – not high-speed
  - ROLLO – not conservative
  - Round5 – patent encumbered

Dagger

- Only three NIST round-2 candidates within size constraints:
    - SIKE – not high-speed
    - ROLLO – not conservative
    - Round5 – patent encumbered
- Idea: Tweak lattice-based KEM:
    - More public-key and ciphertext compression
    - Increase hardness of lattice problems
    - Increase failure probability (no issue for CPA sec.)

Dagger

- Only three NIST round-2 candidates within size constraints:
  - SIKE – not high-speed
  - ROLLO – not conservative
  - Round5 – patent encumbered
- Idea: Tweak lattice-based KEM:
  - More public-key and ciphertext compression
  - Increase hardness of lattice problems
  - Increase failure probability (no issue for CPA sec.)
- Tweaked (smaller, more lightweight) Saber: **Dagger**

Implementation and Evaluation

- Implement as Linux kernel module
- Use existing high-speed constant-time software for McEliece and Dagger (Saber)

Implementation and Evaluation

- Implement as Linux kernel module
- Use existing high-speed constant-time software for McEliece and Dagger (Saber)
- Metrics for comparison:
    - Amount of traffic
    - Number of packets
    - Handshake latency

Implementation and Evaluation

- Implement as Linux kernel module
- Use existing high-speed constant-time software for McEliece and Dagger (Saber)
- Metrics for comparison:
  - Amount of traffic
  - Number of packets
  - Handshake latency
- Use virtual 10Gbps Ethernet link between two VMs
- Both IPv4 and IPv6: similar results
- Compare with WireGuard, OpenVPN, IPsec, PQCrypto-VPN

Results

VPN Software	Packet Number	Traffic (bytes)	Client Time (milliseconds)	Server Time (milliseconds)
WireGuard	3	458	0.592	0.480
	(0)	(0)	(0.399)	(0.389)
PQ-WireGuard	3	2654	1.015	0.786
	(0)	(0)	(0.618)	(0.621)
IPsec	6	4299	17.188	11.912
(RSA-2048)	(0)	(0)	(0.712)	(0.535)
IPsec	4	2281	5.226	2.822
(Curve25519)	(0)	(0)	(0.575)	(0.436)
OpenVPN	21.003	7955.409	1148.733	1142.650
(RSA-2048)	(0.055)	(7.319)	(250.513)	(243.184)
OpenVPN	19.005	5788.610	1139.140	1133.944
(NIST P-256)	(0.007)	(9.423)	(247.659)	(240.691)
OpenVPN-NL	19.005	6065.700	1162.649	1151.790
(RSA-2048)	(0.072)	(9.665)	(261.078)	(246.363)
OpenVPN-NL	19.001	6061.138	1159.627	1153.949
(NIST P-256)	(0.003)	(4.304)	(252.989)	(247.470)
PQ-OpenVPN	63.006	35608.817	1160.922	1155.713
(Frodo-752 [**BCD+16**])	(0.078)	(10.324)	(259.246)	(245.614)
PQ-OpenVPN	23.005	8996.684	1277.172	1269.074
(SIDHp503)	(0.072)	(9.449)	(251.461)	(257.427)

More online

Paper:

https://cryptojedi.org/papers/#pqwireguard

Code:

https://cryptojedi.org/crypto/#pqwireguard