Crypto protocols for the post-quantum era: PQ-WireGuard and KEMTLS

Peter Schwabe

September 9, 2021

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.

Challenges for post-quantum protocols

1. Massively different performance characteristics than ECC

Challenges for post-quantum protocols

1. Massively different performance characteristics than ECC
2. Much more complex landscape of crypto primitives

Challenges for post-quantum protocols

1. Massively different performance characteristics than ECC
2. Much more complex landscape of crypto primitives
3. No drop-in replacement for DH

Challenges for post-quantum protocols

1. Massively different performance characteristics than ECC
2. Much more complex landscape of crypto primitives
3. No drop-in replacement for DH
4. Most efficient signatures are stateful (and forward secure)

Challenges for post-quantum protocols

1. Massively different performance characteristics than ECC
2. Much more complex landscape of crypto primitives
3. No drop-in replacement for DH
4. Most efficient signatures are stateful (and forward secure)
5. Potentially more complex proofs

Challenges for post-quantum protocols

1. Massively different performance characteristics than ECC
2. Much more complex landscape of crypto primitives
3. No drop-in replacement for DH
4. Most efficient signatures are stateful (and forward secure)
5. Potentially more complex proofs

Post-quantum WireGuard

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

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

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
$(\mathtt{esk}_i, \mathtt{epk}_i) \leftarrow \mathrm{DH.Gen}()$		
	$\xrightarrow{\mathtt{epk}_i}$	
		$(\mathtt{esk}_r, \mathtt{epk}_r) \leftarrow \mathrm{DH.Gen}()$
	$\xleftarrow{\mathtt{epk}_r}$	
$k_1 \leftarrow \mathrm{DH.Shared}(\mathtt{esk}_i, \mathtt{spk}_r)$		$k_1 \leftarrow \mathrm{DH.Shared}(\mathtt{ssk}_r, \mathtt{epk}_i)$
$k_2 \leftarrow \mathrm{DH.Shared}(\mathtt{ssk}_i, \mathtt{epk}_r)$		$k_2 \leftarrow \mathrm{DH.Shared}(\mathtt{esk}_r, \mathtt{spk}_i)$
$k_3 \leftarrow \mathrm{DH.Shared}(\mathtt{esk}_i, \mathtt{epk}_r)$		$k_3 \leftarrow \mathrm{DH.Shared}(\mathtt{esk}_r, \mathtt{epk}_i)$
$k_4 \leftarrow \mathrm{DH.Shared}(\mathtt{ssk}_i, \mathtt{spk}_r)$		$k_4 \leftarrow \mathrm{DH.Shared}(\mathtt{ssk}_r, \mathtt{spk}_i)$

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: $\mathtt{InitHello} \leftarrow \mathtt{type} \| 0^3 \| \mathtt{sid}_i \| \mathtt{epk}_i \| \mathtt{ltk} \| \mathtt{time} \| \mathtt{m1} \| \mathtt{m2}$

$\xrightarrow{\mathtt{InitHello}}$

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: $\mathtt{RespHello} \leftarrow \mathtt{type} \| 0^3 \| \mathtt{sid}_r \| \mathtt{sid}_i \| \mathtt{epk}_r \| \mathtt{zero} \| \mathtt{m1} \| \mathtt{m2}$

$\xleftarrow{\mathtt{RespHello}}$

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

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

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: 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
    - Resistance against 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 $\Rightarrow$ 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
$A \leftarrow g^a$		$B \leftarrow g^b$
	$\xleftarrow{\quad B \quad}$	
	$\xrightarrow{\quad A \quad}$	
$K \leftarrow B^a = (g^b)^a = g^{ab}$		$K \leftarrow A^b = (g^a)^b = g^{ab}$

KEMs: as close as you'll get to DH

Initiator | **Responder**

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

$\xrightarrow{\quad \mathsf{pk} \quad}$

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

$\xleftarrow{\quad \mathsf{ct} \quad}$

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

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

Initiator | **Responder**

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

$\xrightarrow{\mathrm{pk}}$

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

$\xleftarrow{\mathrm{ct}}$

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

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

What can KEMs (not) do?

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

A first approach with KEMs

Initiator | **Responder**

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

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

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

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

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

$\xleftarrow{c_2, c_3}$

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

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

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

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**

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

$\mathtt{sid}_i \xleftarrow{\$} \{0,1\}^{32}$

$r_i \leftarrow \{0,1\}^{\lambda}$

$(\mathtt{ct1}, \mathtt{shk1}) \leftarrow \text{CCAKEM.Enc}(\mathtt{spk}_r, \text{KDF}_1(\sigma_i, r_i))$

$\mathtt{ltk} \leftarrow \text{AEAD.Enc}(\kappa_3, 0, \text{H}(\mathtt{spk}_i), H_3)$

$now \leftarrow \text{Timestamp}()$

$\mathtt{time} \leftarrow \text{AEAD.Enc}(\kappa_4, 0, H_4, now)$

$\mathtt{m1} \leftarrow \text{MAC}(\text{H}(\mathtt{lbl}_3 \,\|\, \mathtt{spk}_r), \mathtt{type} \,\|\, 0^3 \,\|\, \mathtt{sid}_i \,\|\, \mathtt{epk}_i \,\|\, \mathtt{ct1} \,\|\, \mathtt{ltk} \,\|\, \mathtt{time})$

$\mathtt{m2} \leftarrow \text{MAC}(cookie, \mathtt{type} \,\|\, 0^3 \,\|\, \mathtt{sid}_i \,\|\, \mathtt{epk}_i \,\|\, \mathtt{ct1} \,\|\, \mathtt{ltk} \,\|\, \mathtt{time} \,\|\, \mathtt{m1})$

$\mathtt{InitHello} \leftarrow \mathtt{type} \,\|\, 0^3 \,\|\, \mathtt{sid}_i \,\|\, \mathtt{epk}_i \,\|\, \mathtt{ct1} \,\|\, \mathtt{ltk} \,\|\, \mathtt{time} \,\|\, \mathtt{m1} \,\|\, \mathtt{m2}$

$\xrightarrow{\mathtt{InitHello}}$

$e, r_r \leftarrow \{0,1\}^{\lambda} \times \{0,1\}^{\lambda}$

$(\mathtt{ct2}, \mathtt{shk2}) \leftarrow \text{CPAKEM.Enc}(\mathtt{epk}_i, e)$

$(\mathtt{ct3}, \mathtt{shk3}) \leftarrow \text{CCAKEM.Enc}(\mathtt{spk}_i, \text{KDF}_1(\sigma_r, r_r))$

$\mathtt{sid}_r \xleftarrow{\$} \{0,1\}^{32}$

$\mathtt{zero} \leftarrow \text{AEAD.Enc}(\kappa_9, 0, H_9, \emptyset)$

$\mathtt{m1} \leftarrow \text{MAC}(\text{H}(\mathtt{lbl}_3 \,\|\, \mathtt{spk}_i), \mathtt{type} \,\|\, 0^3 \,\|\, \mathtt{sid}_r \,\|\, \mathtt{sid}_i \,\|\, \mathtt{ct2} \,\|\, \mathtt{ct3} \,\|\, \mathtt{zero})$

$\mathtt{m2} \leftarrow \text{MAC}(cookie, \mathtt{type} \,\|\, 0^3 \,\|\, \mathtt{sid}_r \,\|\, \mathtt{sid}_i \,\|\, \mathtt{ct2} \,\|\, \mathtt{ct3} \,\|\, \mathtt{zero} \,\|\, \mathtt{m1})$

$\mathtt{RespHello} \leftarrow \mathtt{type} \,\|\, 0^3 \,\|\, \mathtt{sid}_r \,\|\, \mathtt{sid}_i \,\|\, \mathtt{ct2} \,\|\, \mathtt{ct3} \,\|\, \mathtt{zero} \,\|\, \mathtt{m1} \,\|\, \mathtt{m2}$

$\xleftarrow{\mathtt{RespHello}}$

Adding explicit key confirmation

Initiator | Responder

19: $\texttt{conf} \leftarrow \mathsf{AEAD.Enc}(\kappa_{10}, 0, H_{10}, \emptyset)$
20: $\texttt{m1} \leftarrow \mathsf{MAC}(\mathsf{H}(\texttt{lbl}_3 \parallel \texttt{spk}_r), \texttt{type} \parallel 0^3 \parallel \texttt{sid}_i \parallel \texttt{sid}_r \parallel \texttt{conf})$
21: $\texttt{m2} \leftarrow \mathsf{MAC}(\mathit{cookie}, \texttt{type} \parallel 0^3 \parallel \texttt{sid}_i \parallel \texttt{sid}_r \parallel \texttt{conf} \parallel \texttt{m1})$
22: $\texttt{InitConf} \leftarrow \texttt{type} \parallel 0^3 \parallel \texttt{sid}_i \parallel \texttt{sid}_r \parallel \texttt{conf} \parallel \texttt{m1} \parallel \texttt{m2}$

$\xrightarrow{\texttt{InitConf}}$

23: $tk_i \leftarrow \mathsf{KDF}_1(C_{10}, \emptyset)$
24: $tk_r \leftarrow \mathsf{KDF}_2(C_{10}, \emptyset)$

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

KEMTLS

Peter Schwabe, Douglas Stebila, and Thom Wiggers

The TLS 1.3 handshake

TLS 1.3

Client | **Server**

Server: static (sig): $\mathsf{pk}_S, \mathsf{sk}_S$

Client: $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{DH.Gen}$

Client → Server: pk_e

Server:

$(\mathsf{pk}'_e, \mathsf{sk}'_e) \leftarrow \mathsf{DH.Gen}$

$\mathsf{ss}_e \leftarrow \mathsf{DH.Shared}(\mathsf{sk}'_e, \mathsf{pk}_e)$

$K, K', K'', K''' \leftarrow \mathsf{KDF}(\mathsf{ss}_e)$

Server → Client: $\mathsf{pk}'_e, \mathsf{AEAD}_K(\mathsf{cert}[\mathsf{pk}_S] \| \mathsf{Sig}_{\mathsf{sk}_S}(\mathsf{transcript}) \| \text{key confirmation})$

Server → Client: $\mathsf{AEAD}_{K'}(\text{application data})$

Client:

$\mathsf{ss}_e \leftarrow \mathsf{DH.Shared}(\mathsf{sk}_e, \mathsf{pk}'_e)$

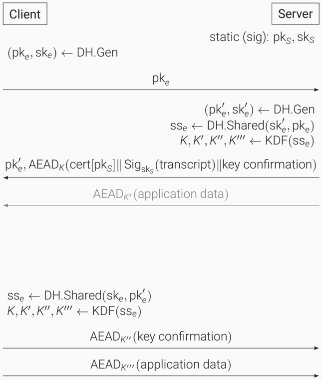
$K, K', K'', K''' \leftarrow \mathsf{KDF}(\mathsf{ss}_e)$

Client → Server: $\mathsf{AEAD}_{K''}(\text{key confirmation})$

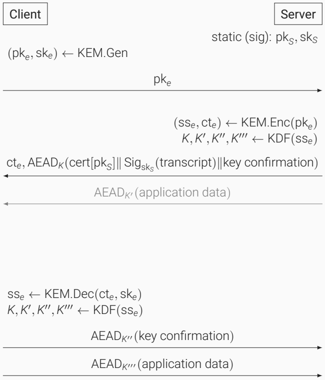
Client → Server: $\mathsf{AEAD}_{K'''}(\text{application data})$

The TLS 1.3 handshake

TLS 1.3



"Straight-forward" PQTLS



KEMTLS – the idea

- Remove signatures from handshake
- Obtain authentication using long-term KEM keys
- Inspiration from DH-based OPTLS by Krawczyk and Wee (2015)

KEMTLS – the idea

- Remove signatures from handshake
- Obtain authentication using long-term KEM keys
- Inspiration from DH-based OPTLS by Krawczyk and Wee (2015)

Motivation

- PQ KEMs are more efficient than PQ signatures
- Cannot build KEM and signature from the same “core TCB”

KEMTLS – the idea

- Remove signatures from handshake
- Obtain authentication using long-term KEM keys
- Inspiration from DH-based OPTLS by Krawczyk and Wee (2015)

Motivation

- PQ KEMs are more efficient than PQ signatures
- Cannot build KEM and signature from the same "core TCB"

Challenges

- Public keys are not known in advance
- (Typically only unilateral authentication)

The KEMTLS handshake

TLS 1.3

Client | Server

static (sig): $\mathsf{pk}_S, \mathsf{sk}_S$

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

pk_e

$(\mathsf{pk}'_e, \mathsf{sk}'_e) \leftarrow \mathsf{DH.Gen}$

$\mathsf{ss}_e \leftarrow \mathsf{DH.Shared}(\mathsf{sk}'_e, \mathsf{pk}_e)$

$K, K', K'', K''' \leftarrow \mathsf{KDF}(\mathsf{ss}_e)$

$\mathsf{pk}'_e, \mathsf{AEAD}_K(\mathsf{cert}[\mathsf{pk}_S] \| \mathsf{Sig}_{\mathsf{sk}_S}(\mathsf{transcript}) \| \mathsf{key\ confirmation})$

$\mathsf{AEAD}_{K'}(\mathsf{application\ data})$

$\mathsf{ss}_e \leftarrow \mathsf{DH.Shared}(\mathsf{sk}_e, \mathsf{pk}'_e)$

$K, K', K'', K''' \leftarrow \mathsf{KDF}(\mathsf{ss}_e)$

$\mathsf{AEAD}_{K''}(\mathsf{key\ confirmation})$

$\mathsf{AEAD}_{K'''}(\mathsf{application\ data})$

KEMTLS

Client | Server

static (KEM): $\mathsf{pk}_S, \mathsf{sk}_S$

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

pk_e

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

$K_1, K'_1 \leftarrow \mathsf{KDF}(\mathsf{ss}_e)$

$\mathsf{ct}_e, \mathsf{AEAD}_{K_1}(\mathsf{cert}[\mathsf{pk}_S])$

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

$K_1, K'_1 \leftarrow \mathsf{KDF}(\mathsf{ss}_e)$

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

$\mathsf{AEAD}_{K'_1}(\mathsf{ct}_S)$

$K_2, K'_2, K''_2, K'''_2 \leftarrow \mathsf{KDF}(\mathsf{ss}_e \| \mathsf{ss}_S)$

$\mathsf{AEAD}_{K_2}(\mathsf{key\ confirmation})$

$\mathsf{AEAD}_{K'_2}(\mathsf{application\ data})$

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

$K_2, K'_2, K''_2, K'''_2 \leftarrow \mathsf{KDF}(\mathsf{ss}_e \| \mathsf{ss}_S)$

$\mathsf{AEAD}_{K''_2}(\mathsf{key\ confirmation})$

$\mathsf{AEAD}_{K'''_2}(\mathsf{application\ data})$

Advantages/Disadvantages of KEMTLS

Advantages

- Faster handshake until first client payload
- Fewer (server) CPU cycles
- Possible to massively reduce bandwidth requirements
- Smaller TCB (no signing code!)
- No low-latency requirements for PQ signatures

Advantages/Disadvantages of KEMTLS

Advantages

- Faster handshake until first client payload
- Fewer (server) CPU cycles
- Possible to massively reduce bandwidth requirements
- Smaller TCB (no signing code!)
- No low-latency requirements for PQ signatures

Disadvantages

- No payload in first server message
- Delayed explicit authentication
- Delayed authentication of cipher suite

Handshake performance

Average time in ms for handshake establishment (fast network)

		Handshake time (31.1 ms latency, 1000 Mbps bandwidth)					
		Excl. int. CA cert.			Incl. int. CA cert.		
		Client sent req.	Client recv. resp.	Server HS done	Client sent req.	Client recv. resp.	Server HS done
TLS 1.3	ERRR	66.4	**97.7**	35.5	66.5	**97.7**	35.5
	SFXr	80.1	**111.3**	49.2	80.4	**111.5**	49.4
	KDDD	63.8	**95.1**	32.9	64.1	**95.4**	33.2
	NFFF	64.8	**96.0**	33.8	65.1	**96.4**	34.2
KEMTLS	SSXr	84.5	**124.6**	62.5	84.3	**124.4**	62.3
	KKDD	63.3	**94.8**	32.6	63.7	**95.2**	32.9
	NNFF	63.4	**95.0**	32.7	63.7	**95.3**	33.0

Label syntax: ABCD: A = ephemeral key exchange, B = leaf certificate, C = intermediate CA certificate, D = root certificate.

Label values: Dilithium, ECDH X25519, Falcon, rainbow, Kyber, NTRU, RSA-2048, SIKE, XMSS_s^{MT}; all level-1 schemes.

Handshake performance

Average computation time in ms for asymmetric crypto

		Excl. int. CA cert.		Incl. int. CA cert.	
		Client	Server	Client	Server
TLS 1.3	**ERRR**	0.134	0.629	0.150	0.629
	SFXr	11.860	4.410	12.051	4.410
	KDDD	0.059	0.072	0.081	0.072
	NFFF	0.138	0.241	0.180	0.241
KEMTLS	**SSXr**	15.998	7.173	16.188	7.173
	KKDD	0.048	0.017	0.070	0.017
	NNFF	0.107	0.021	0.149	0.021

Label syntax: ABCD: A = ephemeral key exchange, B = leaf certificate, C = intermediate CA certificate, D = root certificate.

Label values: Dilithium, ECDH X25519, Falcon, rainbow, Kyber, NTRU, RSA-2048, SIKE, $XMSS_s^{MT}$; all level-1 schemes.

Handshake performance

Transmitted bytes for asymmetric cryptographic objects

		Excl. int. CA cert.	Incl. int. CA cert.
TLS 1.3	ERRR	848	1376
	SFXr	2999	3097
	KDDD	7720	11452
	NFFF	3675	5262
KEMTLS	SSXr	1845	1943
	KKDD	5556	9288
	NNFF	3486	5073

Label syntax: ABCD: A = ephemeral key exchange, B = leaf certificate, C = intermediate CA certificate, D = root certificate.

Label values: Dilithium, ECDH X25519, Falcon, rainbow, Kyber, NTRU, RSA-2048, SIKE, $XMSS_s^{MT}$; all level-1 schemes.

KEMTLS with pre-distributed keys

What if the client already knows the server's long-term key?

KEMTLS with pre-distributed keys

What if the client already knows the server's long-term key?

Examples

- IoT devices that only commicate to one server
- Caching of public keys in the browser
- Pre-distribution of keys via, e.g., DNS
- Apps that communicate only to few servers

KEMTLS with pre-distributed keys

What if the client already knows the server’s long-term key?

Examples

- IoT devices that only commicate to one server
- Caching of public keys in the browser
- Pre-distribution of keys via, e.g., DNS
- Apps that communicate only to few servers

Summary

- All the advantages of KEMTLS without the disadvantages
- Different PQ-KEMs become best choice (McEliece. . .)

KEMTLS – ongoing work

- “Real-world” experiment in collaboration with Cloudflare:

  Celi, Faz-Hernández, Sullivan, Tamvada, Valenta, Wiggers, Westerbaan, and Wood: *Implementing and Measuring KEMTLS.* `https://eprint.iacr.org/2021/1019`

- Internet draft:

  Celi, Schwabe, Stebila, Sullivan, and Wiggers: *KEM-based Authentication for TLS 1.3.* `https://datatracker.ietf.org/doc/html/draft-celi-wiggers-tls-authkem-00`

- Formal verification using Tamarin (Hoyland and Wiggers; very much WIP)

Resources online

- PQ-WireGuard paper: https://eprint.iacr.org/2020/379
- PQ-WireGuard software:
  https://cryptojedi.org/crypto/#pqwireguard

Resources online

- PQ-WireGuard paper: https://eprint.iacr.org/2020/379
- PQ-WireGuard software: https://cryptojedi.org/crypto/#pqwireguard
- KEMTLS paper: https://eprint.iacr.org/2020/534
- KEMTLS with predistributed keys: https://eprint.iacr.org/2021/779
- KEMTLS software: https://github.com/thomwiggers/kemtls-experiment/