Key-Update Mechanism for SDLSP

Florian Weber

joined work with Andreas Hülsing and Tanja Lange

Symmetric Key-Updates

- SDLSP only supports symmetrically secured key-updates at the moment.

Symmetric Key-Updates

- SDLSP only supports symmetrically secured key-updates at the moment.

Advantages

- Post Quantum.
- Forward-secure.

Symmetric Key-Updates

- SDLSP only supports symmetrically secured key-updates at the moment.

Advantages

- Post Quantum.
- Forward-secure.

Disadvantages

- No recovery from corruption.
- Does not scale well.
- No way to integrate with PKI.

Symmetric Key-Updates

- SDLSP only supports symmetrically secured key-updates at the moment.

Advantages

- Post Quantum.
- Forward-secure.

Disadvantages

- No recovery from corruption.
- Does not scale well.
- No way to integrate with PKI.

$\Rightarrow$ Introduce a post-quantum, forward-secure *asymmetric* key-update mechanism.

- In essence this is a form of an Authenticated Key Exchange (AKE).

Authenticated Key Exchange

- Two parties, each may have a long-term key-pair for authentication.
- Compute a shared secret that only those two parties know.
- Authenticated = Parties can be certain, that the peer is who they think it is.

Hybrid Security

Many post-quantum schemes are new, what if they are insecure?

- Use a fallback (usually: an elliptic curve algorithm)
- Fallback does not necessarily have to be pre-quantum!
- Not all schemes really require fallbacks (hash-based signatures)

Updating long-term keys

- Requiring a key-update implies that there is no acceptable way to indefinitely store keys in a secure manner.
- If long-term keys cannot be updated than they are implicitly assumed to stay secure indefinitely.

$\Rightarrow$ long-term secrets should probably be updatable.

Possible Approaches

Considered KEMs

- NIST's choice: Kyber
- More conservative: Frodo
- Worth a look for special use-cases: Classic McEliece
- Pre-Quantum (Fallback): X25519

- Still in NIST competition: BIKE, HQC (sizes like Frodo)
- Lost NIST competition: Saber, NTRU, NTRU prime (sizes like Kyber)
- Broken: SIKE

Table 1: KEM-sizes (in bytes)

Scheme	SK	PK	CT
Kyber-512	1632	800	768
Kyber-768	2400	1184	1088
Kyber-1024	3168	1568	1568
FrodoKEM-640	19888	9616	9720
FrodoKEM-976	31296	15632	15744
FrodoKEM-1344	43088	21520	21632
mceliece348864	6492	261120	96
mceliece460896	13608	524160	156
mceliece6688128	13932	1044992	208
mceliece6960119	13948	1047319	194
mceliece8192128	14120	1357824	208
X25519	32	32	32

Considered Signatures

- NIST's first Choice: Dilithium
- NIST's second Choice: Falcon
- Pre-Quantum (Fallback): ECDSA

- Not competitive: SPHINCS+, Picnic (huge signatures)
- Broken: Rainbow
- Weakened: GeMSS

Table 2: Signature-Sizes (in bytes)

Scheme	SK	PK	Sig
Dilithium2	2544	1312	2420
Dilithium3	4016	1952	3293
Dilithium5	4880	2592	4595
Falcon-512	1281	897	666
Falcon-1024	2305	1793	1280
ECDSA(P256)	32	32	64

Stateful Signatures

- SPHINCS+ is not competitive
- Its components might be...
- Assumptions are more conservative than those of pre-quantum signatures!
  - Level 1 might be enough?
- Some versions *already* standardized.
- But: Stateful!
  - Footgun, but possibly managable in our setting.
  - Hybrid sigantures might take the edge of?

Table 3: Sizes of Stateful Signatures (in bytes)

Scheme	SK	PK	Sig
XMSS-SHA2_10_256	32	64	2500
XMSS-SHA2_16_256	32	64	2692
XMSS-SHA2_20_256	32	64	2820
LMS 15	32	56	1616
WOTS+(32,16)	32	32	2144
WOTS+(32,4)	32	32	4256
WOTS+(64,16)	64	64	8384
WOTS+(64,4)	64	64	16704
LMOTS_SHA256_N32_W1	32	56	8516
LMOTS_SHA256_N32_W2	32	56	4292
LMOTS_SHA256_N32_W4	32	56	2180
LMOTS_SHA256_N32_W8	32	56	1124

Signatures + KEM (massively simplified!)

Mission Control | **Satellite**

$pk_e, sk_e := EKEM.gen()$

$pk'_{mc}, sk'_{mc} := Sig.gen()[opt1]$

$\xrightarrow{pke,\ Enc(psk, pk'_{mc}) \quad sig(sk_{mc}, ...)}$

$ce, ke = EKEM.enc(pke)$

$\xleftarrow{ce,\ Enc((psk, ke), pks') \quad sig(sks, ...)}$

$pks', sks' = Sig.gen()[opt2]$

$\xrightarrow{Enc((psk, ke), "")}$

- Requires replay-protection! For example a counter or key-confirmation (shown).
- 1 Roundtrip/ 1.5 Roundtrips
- If the signature-scheme is stateful: long-term key-update non-optional.

Triple-KEM (massively simplified!)

Mission Control | **Satellite**

$pk_e, sk_e := EKEM.gen()$

$pk'_{mc}, sk'_{mc} := Sig.gen()[opt1]$

$c_s, k_s = SKEM.enc(pk_s)$

$\xrightarrow{Enc(psk, c_s), pk_e, \quad Enc((psk, k_s), pk'_{mc})}$

$c_e, k_e = EKEM.enc(pk_e)$

$c_{mc}, k_{mc} = MCKEM.enc(pk_{mc})$

$pk'_s, sk'_s = SKEM.gen()[opt2]$

$\xleftarrow{c_e, Enc((psk, k_s, k_e), c_{mc}) \quad Enc((psk, k_s, ke, k_{mc}), pk'_s)}$

$\xrightarrow{Enc((psk, k_s, k_e, k_{mc}), "")}$

- Usually more efficient (KEMs instead of signatures).
- Essentially invulnerable to replay-attacks.
- Option to mix KEMs.

Results

Sign+KEM

Table 4: SK = Sign+KEM, SKU = with long-term-key-Update, SC = Signature-Chain

Scheme	Packet 1	Packet 2	Packet 3
SK(Kyber512+X25519+Dilithium+ECDSA)	3348	3300	(16)
SKU(Kyber512+X25519+Dilithium+ECDSA)	4692	4644	(16)
SK(Kyber512+X25519+Falcon+ECDSA)	1594	1546	(16)
SKU(Kyber512+X25519+Falcon+ECDSA)	2523	2475	(16)
SK(Kyber512+X25519+XMSS-SHA2_10_256)	3364	3316	(16)
SKU(Kyber512+X25519+XMSS-SHA2_10_256)	3428	3380	(16)
SC(Kyber512+X25519,WOTS+(32,16))	3024	2992	(16)
SC(Kyber768+X25519,WOTS+(32,16))	2408	3312	(16)
SC(Kyber1024+X25519,WOTS+(32,16))	3792	3792	(16)
SC(Kyber1024+X25519,WOTS+(64,16))	10032	10032	(16)

Triple-KEM

Table 5: TK = Triple-KEM, TKU = with long-term-key-Update

Scheme	Packet 1	Packet 2	Packet 3
TK(Kyber512,McEliece348864)	928	880	16
TKU(Kyber512,McEliece348864)	262048	262016	16
TK(Kyber512+X25519)	1664	1616	16
TKU(Kyber512+X25519)	2496	2464	16
TK(Kyber768+X25519)	2368	2256	16
TKU(Kyber768+X25519)	3584	3488	16
TK(Kyber1024+X25519)	3232	3216	16
TKU(Kyber1024+X25519)	4832	4832	16

Our Pre-Selection

Table 6: SK = Sign+KEM, TK = Triple-KEM, SC = Signature-Chain

Scheme	Packet 1	Packet 2	Packet 3
SK(Kyber512+X25519+Dilithium+ECDSA)	3348	3300	(16)
SK(Kyber512+X25519+Falcon+ECDSA)	1594	1546	(16)
SK(Kyber512+X25519+XMSS-SHA2_10_256)	3364	3316	(16)
SC(Kyber512+X25519,WOTS+(32,16))	3024	2992	(16)
TK(Kyber512+X25519)	1664	1616	16

- First Recommendation: TK(Kyber)
- Second Recommendation: SK(Kyber, Falcon)

Discussion

One-Time Keys

- Our protocols assume that ephemeral keys are not reused.
- Real world implementations sometimes decide to “optimize” this.
- Can we assume that yours won’t, or do the protocols need to be hardened?