SPHINCS$^+$

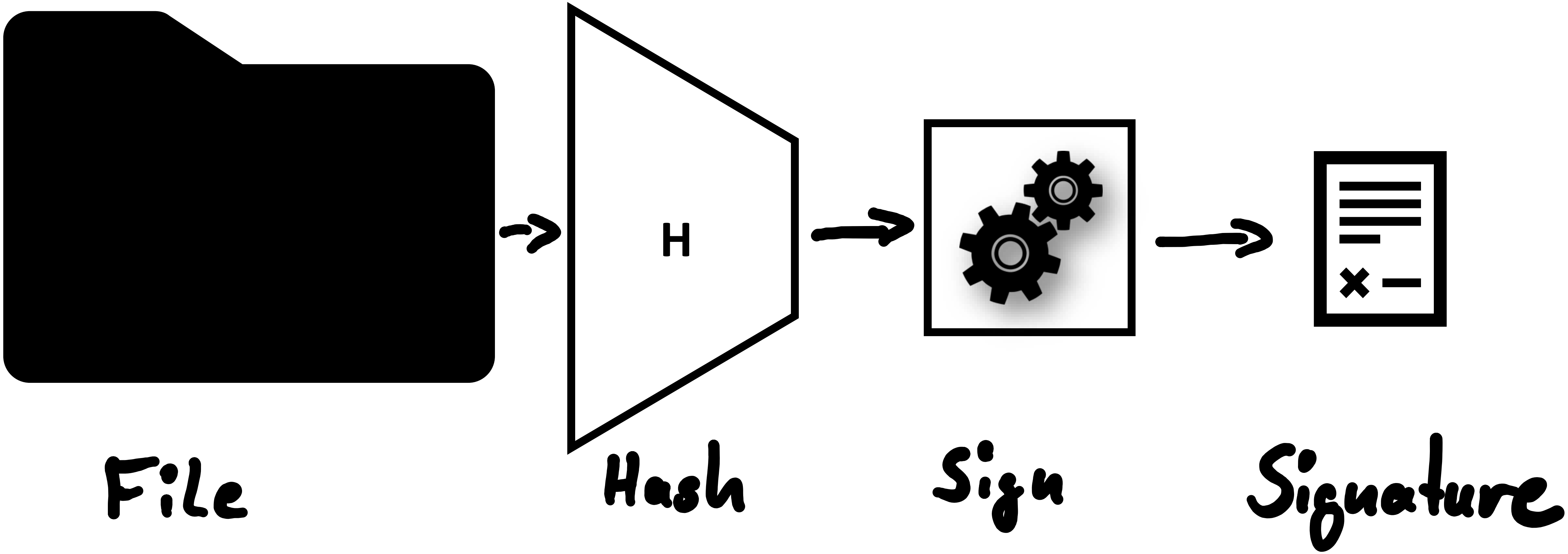
Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens,
Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer,
Stefan-Lukas Gazdag, **Andreas Hülsing**, Panos Kampanakis,
Stefan Kölbl, Mike Kudinov, Tanja Lange, Martin M. Lauridsen,
Florian Mendel, Ruben Niederhagen, Christian Rechberger,
Joost Rijneveld, Peter Schwabe, Bas Westerbaan

Hash-based signatures
(Merkle ´89)

Boring crypto:

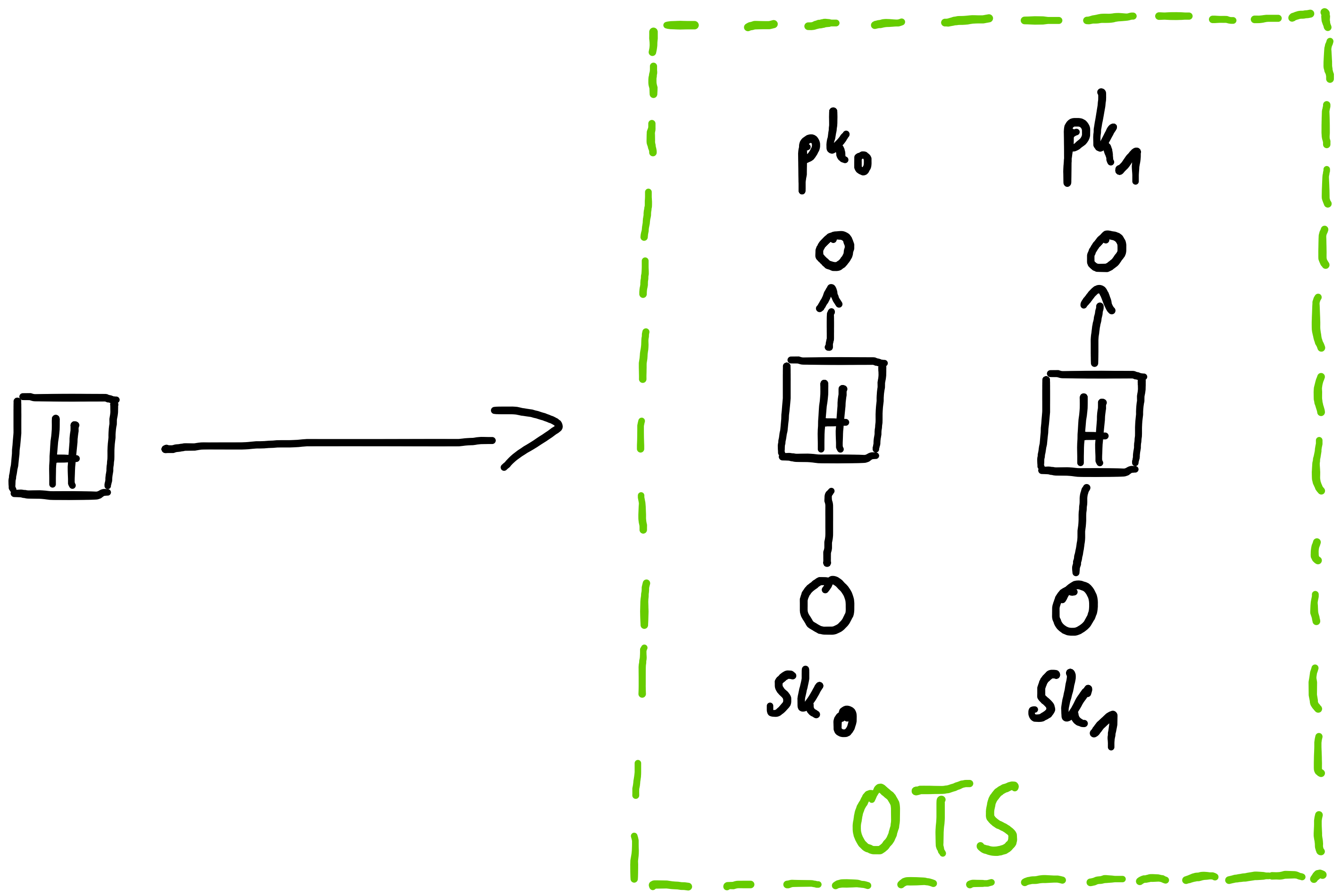
- Dates back to beginning of public key cryptography
- No fancy new mathematical assumption:
  Only requires a secure hash function
  („minimal security assumptions“)
- Stateful schemes are first PQ-signatures standardized
  (LMS & XMSS)

Signatures & Hash Functions



One-time signatures (Lamport'76)

(1-bit)



SPHINCS (Eurocrypt 2015)

Joint work with Daniel J. Bernstein, Daira Hopwood, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn

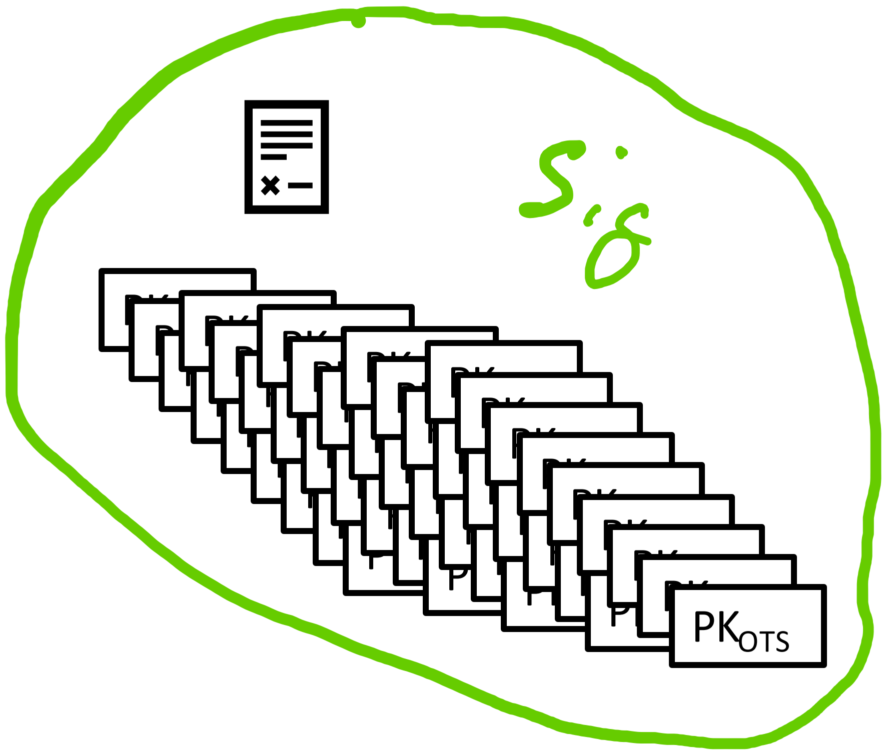
SPHINCS(+) Design Criteria

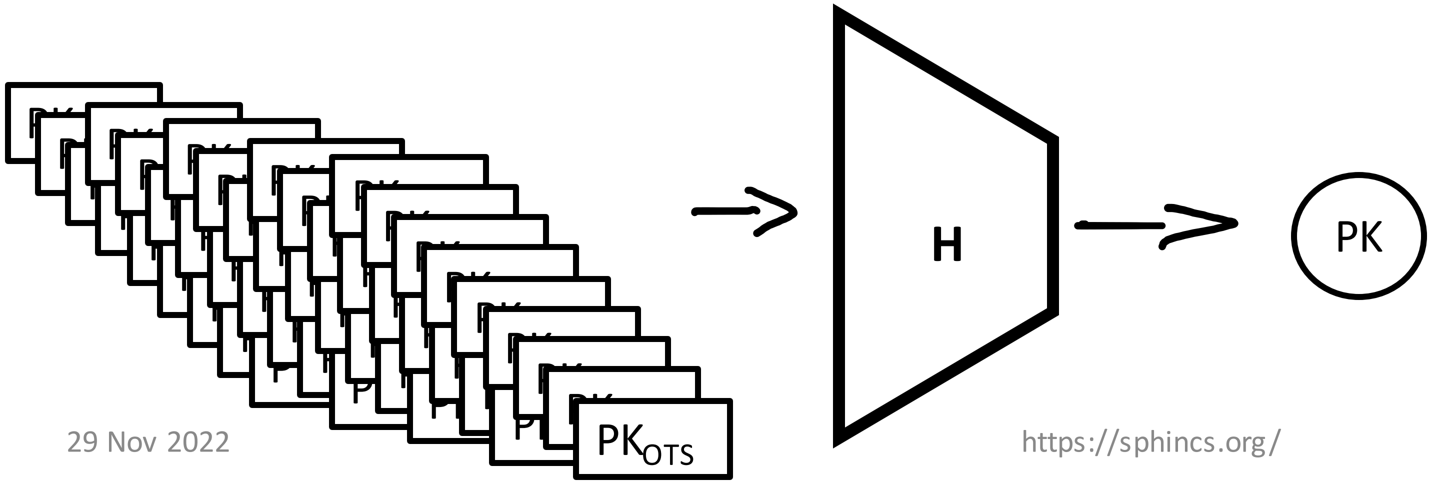
- Stateless
- Practical performance
- Conservative security
  - Collision resilience
  - n-bit hash == n-bit classical security
    (n/2-bit quantum security)

How to go stateless (from an OTS)

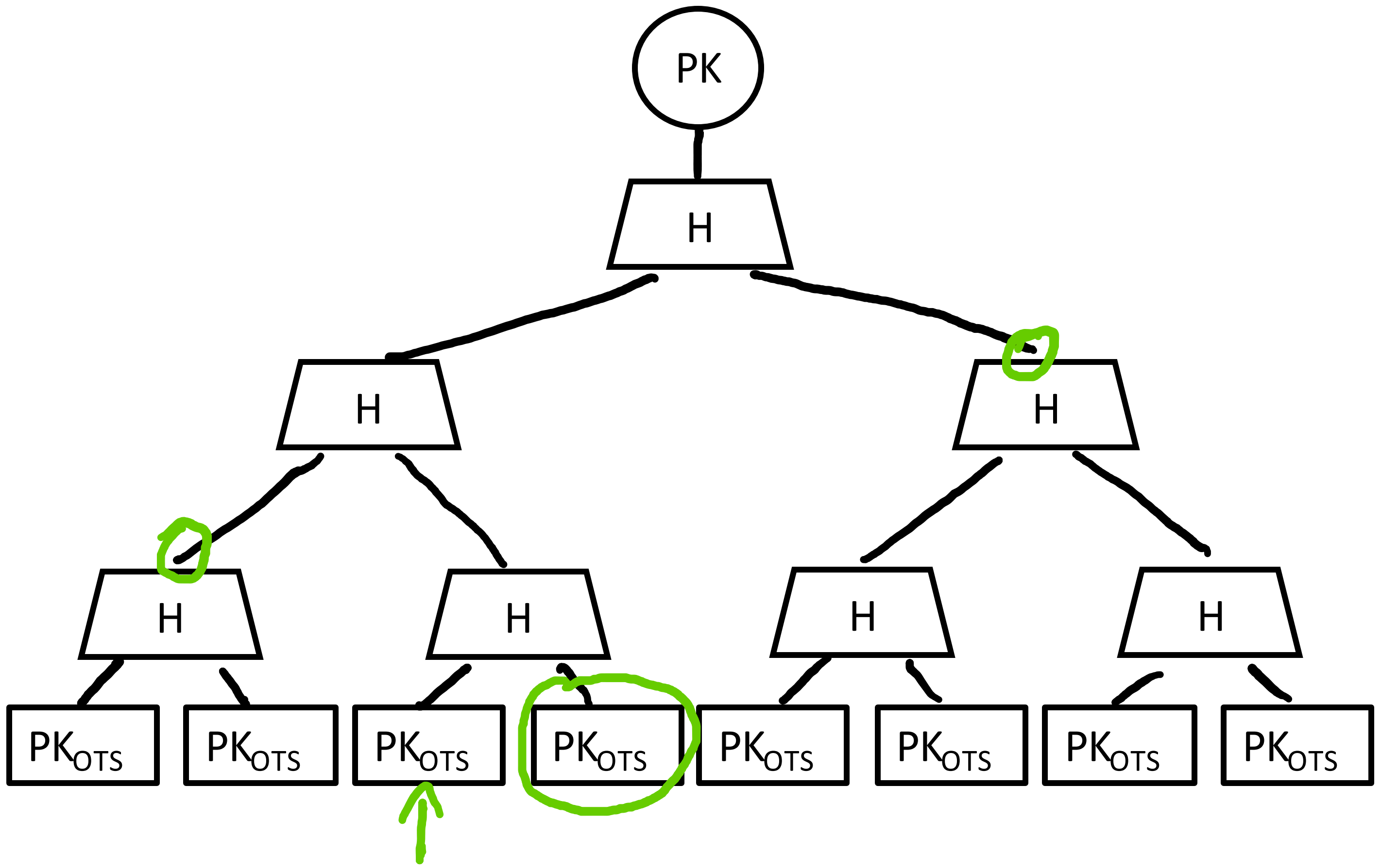
Security parameter k

1. Generate 2^{2k} OTS key pairs
2. Authenticate all OTS public keys
3. Sign message with random OTS
4. Sig is OTS sig + authentication information





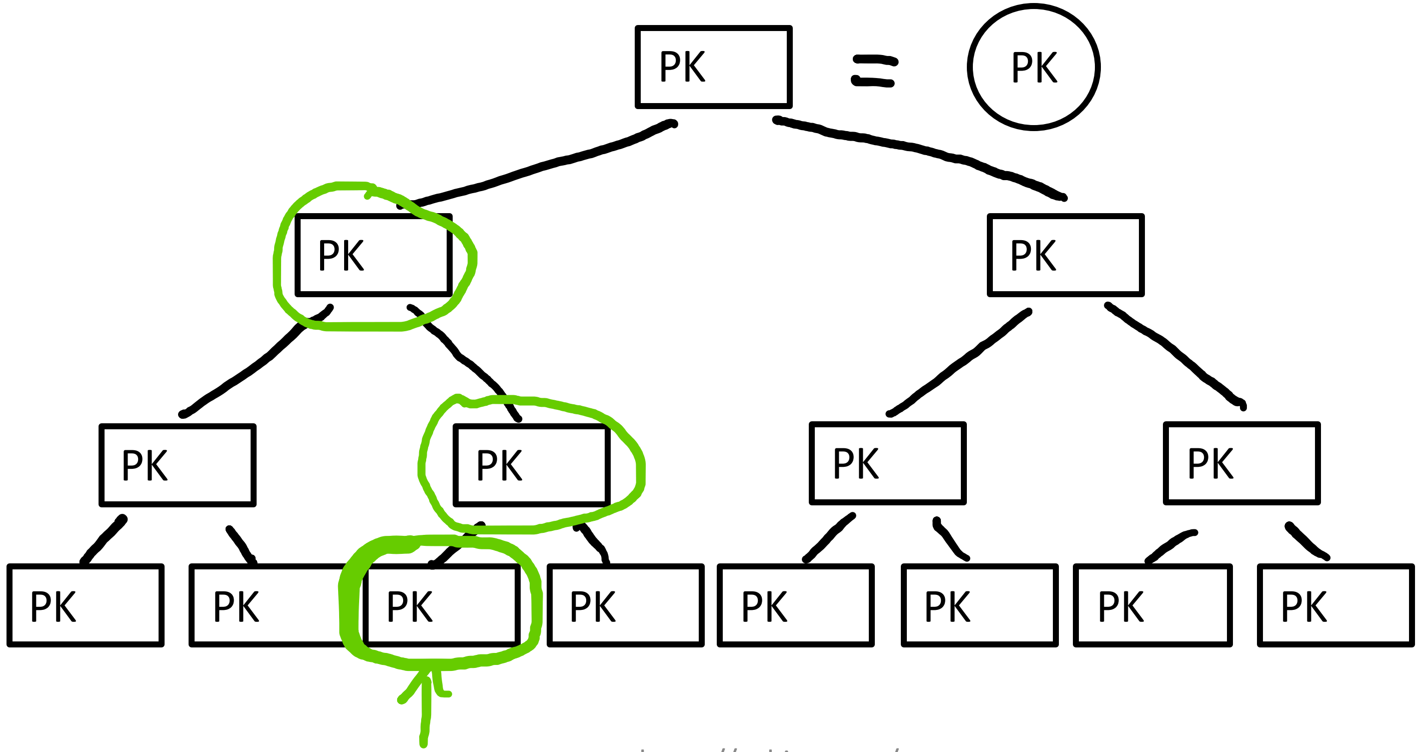
Merkle Tree [Merkle'79]



Certification Tree [Merkle'87]

(for 2-time signature)

— = Certification (Signature on PK)



Stateless hash-based signatures

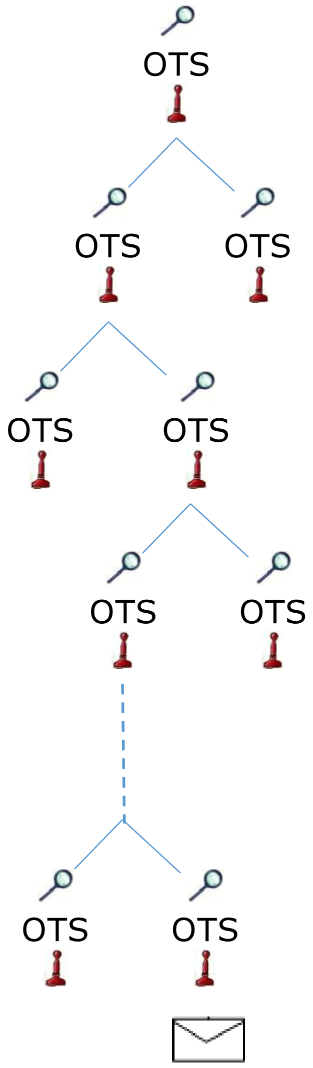
[NY89,Gol87,Gol04]

Goldreich's approach [Gol04]:

Security parameter k = 128

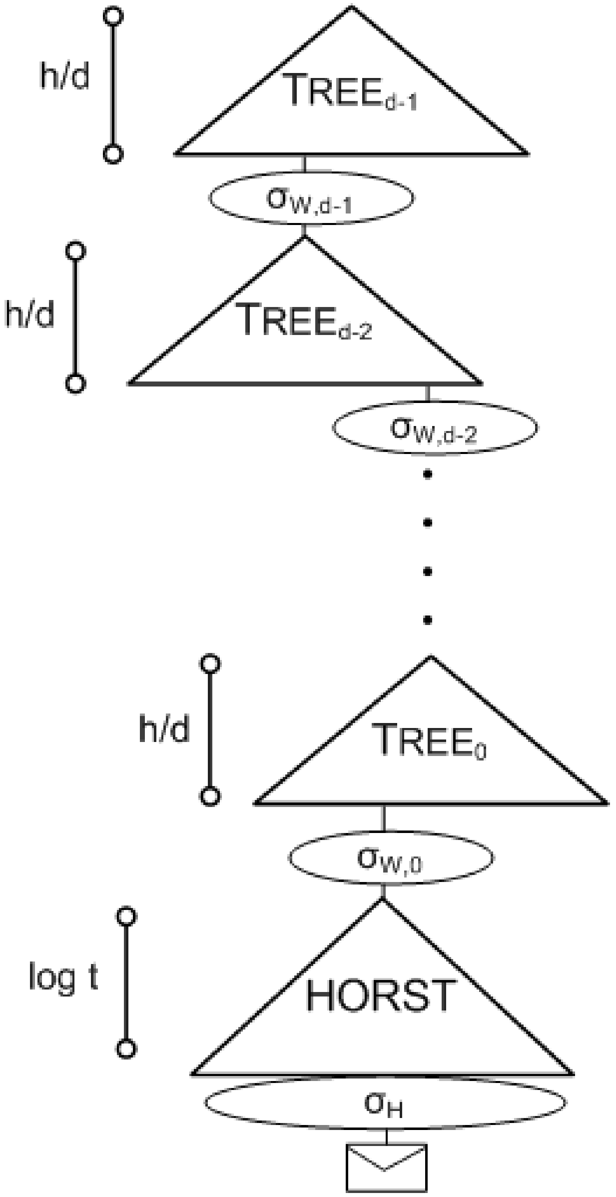
- Use binary certification tree with OTS
- Key pairs are generated pseudorandomly
- Requires huge tree to avoid collisions (height 256)

Ok speed but HUGE signatures



SPHINCS [BHH+15]

- Select index (pseudo-)randomly
- Mix both methods:
  Use a certification tree of Merkle trees
- Use a few-time signature key-pair on leaves to sign messages
    - Few index collisions allowed
    - Allows to reduce tree height (± 64)



SPHINCS+ vs SPHINCS

- Allow for 2^{64} instead of 2^{50} signatures per key pair
- Add multi-target attack mitigation (Tweakable hash functions)
- “Simple” and “Robust” parameters
- New few-time signature scheme FORS
- Verifiable index selection
- Optional non-deterministic signatures

Sizes

	Sec	public key size	secret key size	signature size
SPHINCS$^+$-128s	I	32	64	7 856
SPHINCS$^+$-128f	I	32	64	17 088
SPHINCS$^+$-192s	III	48	96	16 224
SPHINCS$^+$-192f	III	48	96	35 664
SPHINCS$^+$-256s	V	64	128	29 792
SPHINCS$^+$-256f	V	64	128	49 856

Table 8: Key and signature sizes in bytes

Speed

(on single core of 3Ghz CPU)

	Sign	Verify	|sig|
SPHINCS+ -SHA2-128s-simple	~ 214 ms	~ 0.28 ms	7856 byte
SPHINCS+ -SHA2-128f-simple	~ 11 ms	~ 0.72 ms	17088 byte
SPHINCS+ -SHA2-192s-simple	~ 415 ms	~0.48 ms	16224 byte
SPHINCS+ -SHA2-192f-simple	~ 18 ms	~ 1.17 ms	35664 byte

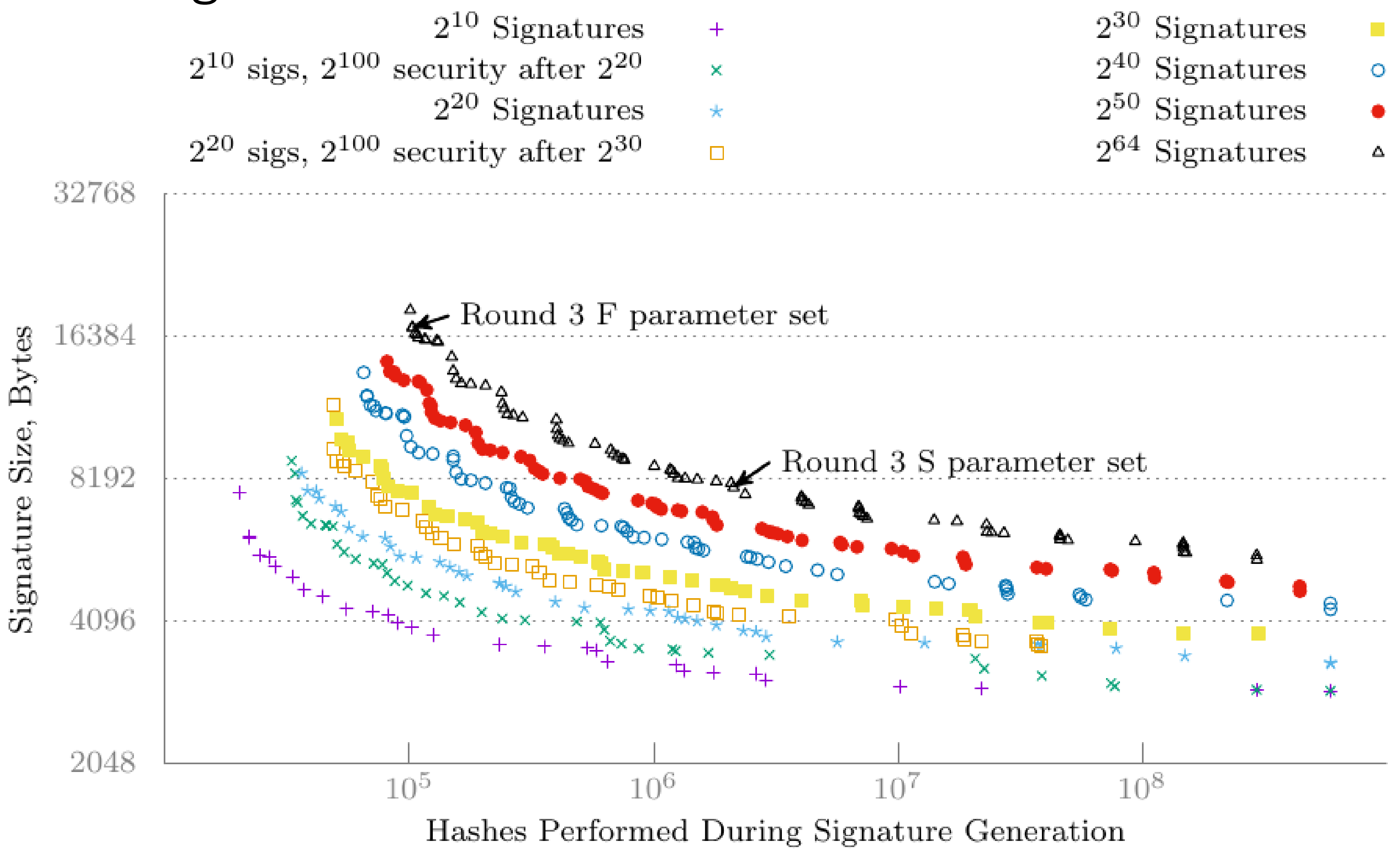
News

- Fixed tight security proof.
  [Hülsing, Kudinov. "Recovering the tight security proof of SPHINCS+."https://eprint.iacr.org/2022/346.pdf]
  - Main technical part of proof already verified in EasyCrypt
- Implementation on Arm Cortex-{A55,A510,A78,A710,X1,X2}.
  [Becker, Kannwischer, "Hybrid scalar/vector implementations of Keccak and SPHINCS+ on AArch64". https://kannwischer.eu/papers/2022_armv8keccak.pdf]
- Talk next session: Qian Wang. Optimization for SPHINCS+ using Intel® Secure Hash Algorithm Extensions
- …

We want feedback!

Lower q_{sign}?

- "NIST asks for public feedback on a version of SPHINCS+ with a lower number of maximum signatures."



Lower q_{sign}?

- Note to come to ePrint soon.
- **Factor > 2 size reduction** (for 2^{20} sigs)!
- Results for NIST level I security -> Interest in higher levels?
- What applications would benefit?
- What would be the number of expected signatures?
- Does the reduced size / better speed
  make a <u>fundamental</u> difference?

SPHINCS+C?

- No Spoilers! Watch next talk ☺
- What applications would benefit?
- What would be the number of expected signatures?
- Does the reduced size / better speed
  make a fundamental difference?
- How relevant is constant signing time
  (not be confused with constant time implementation)
- **In short: Would the change be justified?**

Conclusion

- The most conservative selected signature scheme.
- No size & speed records, but for many applications...
  (e.g. code-signing, email & document signatures, etc)
    - … size is negligible compared to data, and
    - … runtime is not that critical
    - … (long-term) security is of utmost importance
- Possible synergies with stateful hash-based signatures
- PLEASE give us your feedback

Thank you!
Questions?