Determining SPHINCS+ Readiness for Standardization of SLH-DSA Signature

Jessica Ancillotti
Master's Thesis Defense
May 30, 2025

Co-Chair: Stanisław Radziszowski &
T.J. Borrelli
Observer: Richard Lange
Materials located at: https://people.rit.edu/jna1172/

Agenda

1. Thesis Overview
2. Key Concepts
3. Enter PQC
4. SPHINCS+ Overview
5. Objective's 1-4
6. Future Work
7. Conclusion

1 Thesis Overview

Thesis Overview

- ✓ Objective One: Comparative Analysis of SPHINCS+
- ✓ Objective Two: Hybrid Cryptographic Schemes
- ✓ Objective Three: SPHINCS+ in Blockchain Systems
- ✓ Objective Four: Feasibility of SPHINCS+

2

Key Concepts

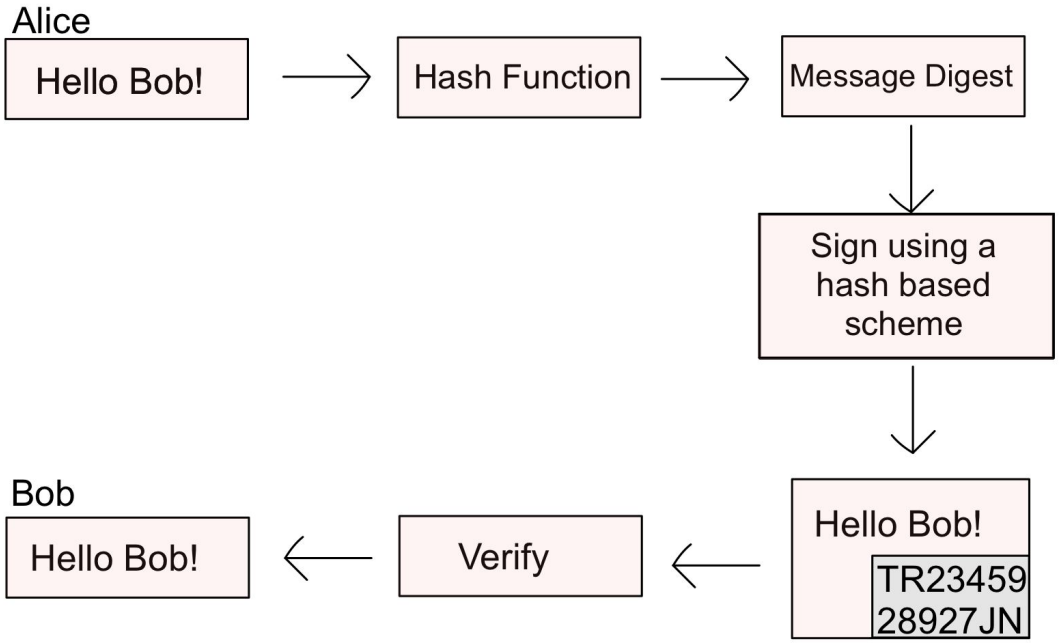
Digital Signatures

- Proves that a message comes from a specific sender and that it hasn’t been altered.
- Used in many identification and authentication protocols to provide authenticity, integrity, and non-repudiation of data.
- Several signature schemes that we use today include RSA, DSA, and ECDSA.

Hash-Based Signatures

- Generated from hash functions like SHA, SHAKE, etc.
- Based on the properties of one-way hash functions.
- Key generation, signing, and verification.
- Large signature sizes and the **secure hash functions** need to be resistant to collision attacks.

Hash-Based Digital Signature Example



Hash-Based vs Traditional Signatures

- The security of RSA and ECDSA rely on the hardness of the integer factorization and the elliptic curve discrete logarithm problem.
- Hash-based signatures rely on the security of hash functions, which are believed to be stronger against quantum attacks.

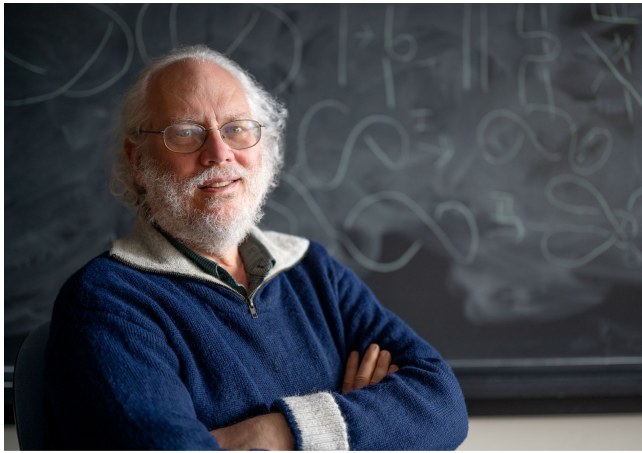
State vs Stateless

- Stateful Scheme
  - Require tracking internal state (to avoid OTS key reuse)
  - OTS key reuse can lead to signature forgery
  - Offer smaller key and signature sizes
- Stateless Scheme
  - Do not require maintaining signing state
  - Add redundancy and randomness to each signature
  - More secure against misuse or implementation errors
  - But larger signatures and longer signing times

Enter Post-Quantum Cryptography

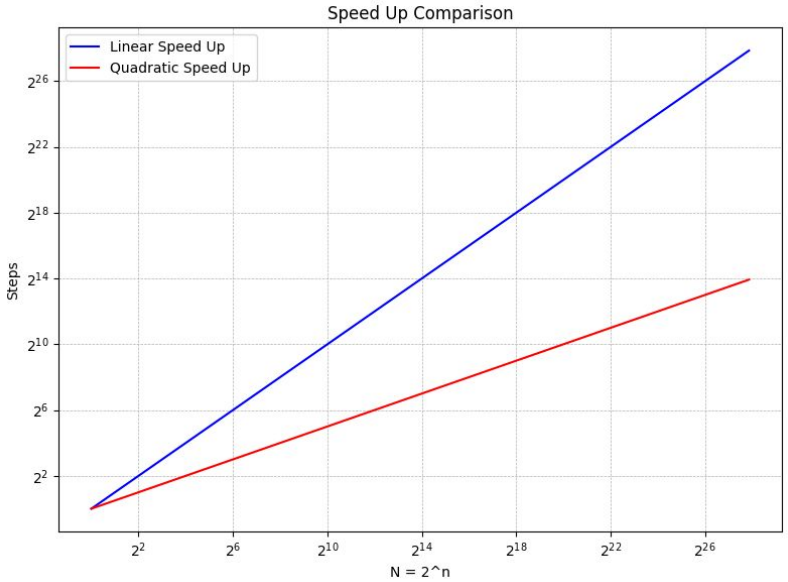
Shor's Algorithm

- **Shor's Algorithm** – used to solve integer factorization problem (and Discrete Log problem) in quantum polynomial time by utilizing quantum Fourier Transform and some classical techniques from number theory
- **Bad news for RSA and ECDSA**



Grover's Algorithms

- **Grover's Algorithm** – quantum algorithm that can be used as reverse database lookup to retrieve output in just √N queries on a database of size N
- **128 bit-hash function becomes as strong as a 64 bit one in QC**
- **But,** we can double the hash size using SHA-256 or SHA-512



Quantum-vulnerable Digital Signature Algorithms

Digital Signature Algorithm Family	Parameters	Transition
ECDSA [FIPS186]	112 bits of security strength	***Deprecated*** after 2030 ***Disallowed*** after 2035
	≥ 128 bits of security strength	***Disallowed*** after 2035
EdDSA [FIPS186]	≥ 128 bits of security strength	***Disallowed*** after 2035
RSA [FIPS186]	112 bits of security strength	***Deprecated*** after 2030 ***Disallowed*** after 2035
	≥ 128 bits of security strength	***Disallowed*** after 2035

NIST's not well done table

Quantum-vulnerable Digital Signature Algorithms

Digital Signature Family	Parameters	Transition
ECDSA [FIPS186]	112 bits of security strength	Deprecated after 2030 Disallowed after 2035
	$\geq$ 128 bits of security strength	Disallowed after 2035
EdDSA [FIPS186]	$\geq$ 128 bits of security strength	Disallowed after 2035
RSA [FIPS186]	112 bits of security strength	Deprecated after 2030 Disallowed after 2035
	$\geq$ 128 bits of security strength	Disallowed after 2035

Our proposed table

Security Categories

- Levels 1, 3, & 5 map to symmetric key strengths
- Level 2 and 4 based on SHA-256 and SHA-384 collision resistance
- Critique: Levels 2 and 4 are redundant

Security Category	Attack Type	Example
1	Key search on a block cipher with a 128-bit key	AES-128
2	Collision search on a 256-bit hash function	SHA-256
3	Key search on a block cipher with a 192-bit key	AES-192
4	Collision search on a 384-bit hash function	SHA3-384
5	Key search on a block cipher with a 256-bit key	AES-256

SPHINCS+ Overview

Introduced in 2015 by Bernstein, Hopwood, Hulsing, Lange, Niederhagen, Papachristodoulou, Schneider, Schwabe, and Wilcox-O'Hearn

SPHINCS

- First introduced in 2015
- SPHINCS uses Hash to Obtain Random Subset with Trees (HORST) an improvement from HORS to sign messages
- SPHINCS selection of the starting node is pseudorandomly.
- Allow for 2^{50}signatures per key pair

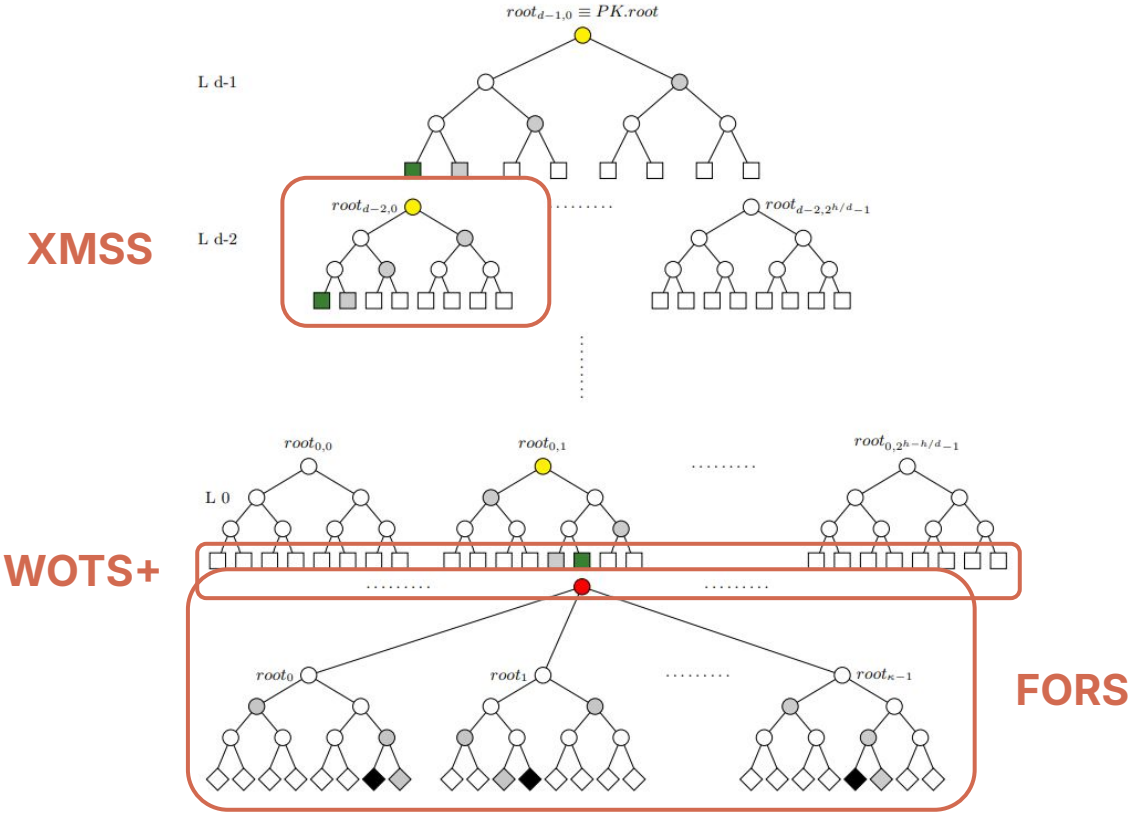
SPHINCS+

- Stateless hash-based signature framework
- First signature scheme to propose parameters to resist quantum cryptanalysis
- SHA-256 & SHAKE hashing algorithms for 128, 192, and 256 bits
- Two modes: Fast (Simple) and Small (Robust)
- Allow for 2^{64} signatures per key pair

SPHINCS+ Key Components + Contributions

- ✓ Verifiable index selection
- ✓ One-Time Signature Scheme (WOTS+)
- ✓ Forest of Random Subsets (FORS), replacing HORST
- ✓ Hypertree
  - Extended Merkle Signature Scheme (XMSS)

SPHINCS+ Example



https://eprint.iacr.org/2021/1495.pdf

SPHINCS+ vs SLH-DSA

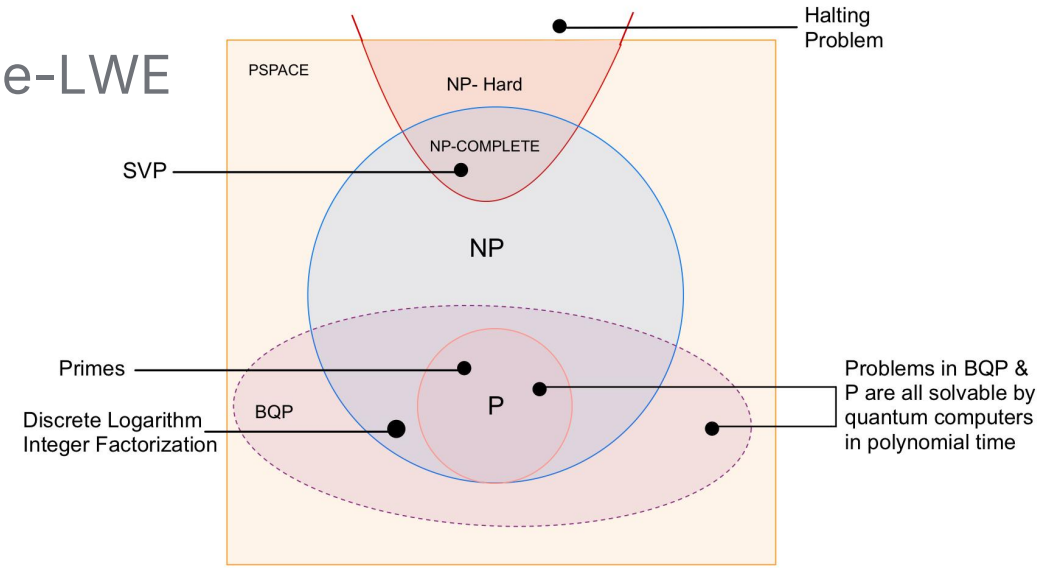
- ✓ Updated Hash Functions:
  - Replaces SHA-256 with SHA-512 in higher security categories to meet full Category 5 strength.
  - Removes Haraka as a hash function option.
- ✓ Standardization Fixes:
  - Clarifies raw messages vs message digest.
  - Aligns FORS index extraction with the reference implementation.

Objective One: Comparison & Benchmark

SPHINCS+ in Comparison

Three NIST Selected PQC Signatures

- ML-DSA - Lattice, Module-LWE
- FALCON - Lattice, NTRU
- SPHINCS+, Hash-Based



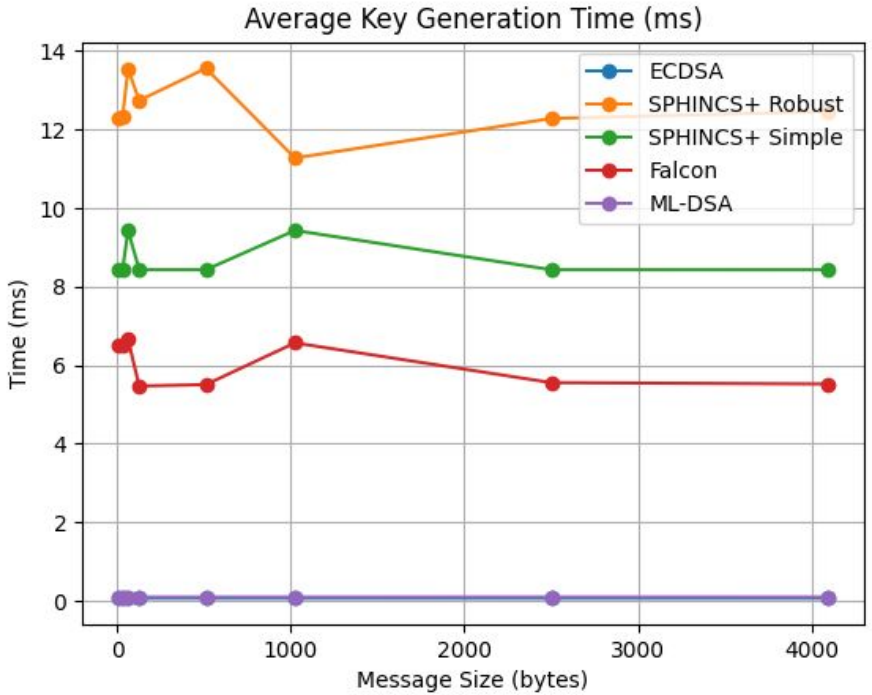
Size Comparison

Method	Public Key Size	Private Key Size	Signature Size	Security Level
ML-DSA-44	1,312	2,560	2,420	2
ML-DSA-65	1,952	4,032	3,309	3
ML-DSA-87	2,592	4,896	4,627	5
Falcon-512	897	1,281	690	1
Falcon-1024	1,793	2,305	1,330	5
SPHINCS+ SHA256-128f Simple	32	64	17,088	1
SPHINCS+ SHA256-128s Robust	32	64	7,856	1
SPHINCS+ SHA256-192f Simple	48	96	35,664	3
SPHINCS+ SHA256-192s Robust	48	96	16,224	3
SPHINCS+ SHA256-256f Simple	64	128	49,856	5
SPHINCS+ SHA256-256s Robust	64	128	29,792	5
RSA-2048	256	256	256	< 1
ECC 256-bit	64	32	64	< 1

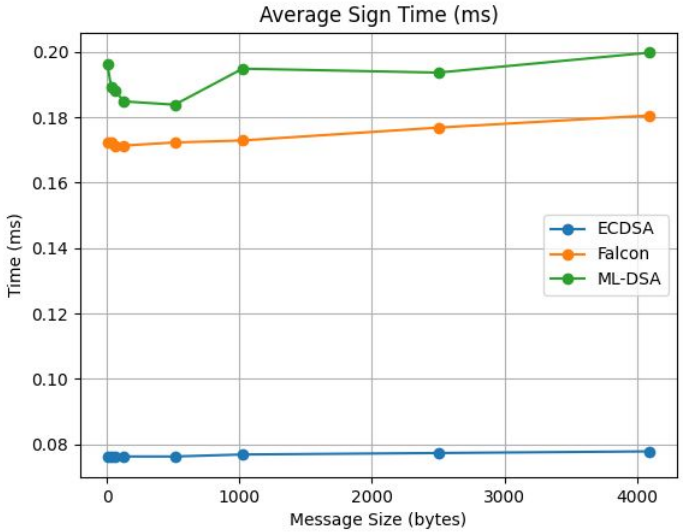
Benchmarking Tests

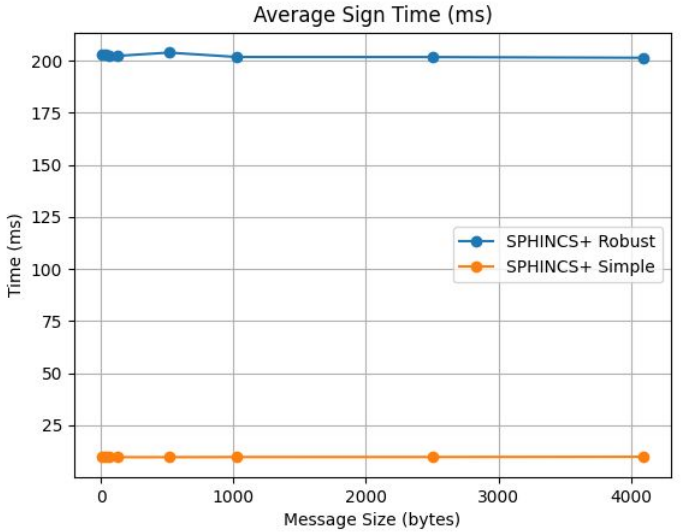
- Performed key generation, signing and verification time across all NIST selected PQC digital signatures and a quantum vulnerable scheme, ECDSA
- 1000 iterations were performed over varying message sizes
  - 4, 32, 64, 128, 512, 1024, 2500, 4096 (Bytes)

Key Generation Time

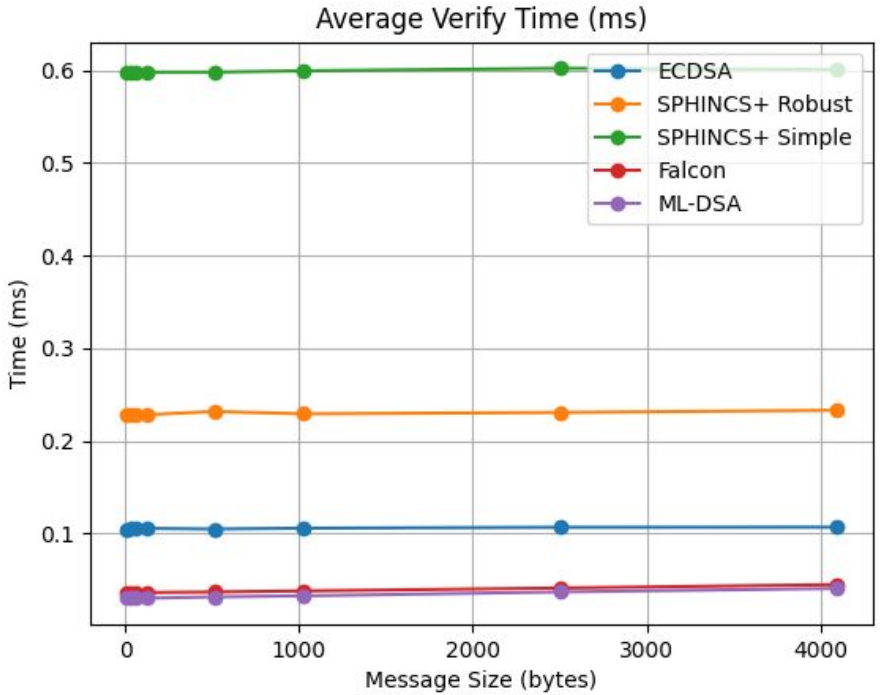


Signing Time





Verification Time



Objective One Conclusion

- All three NIST-selected schemes meet post-quantum security goals but differ in trade-offs.
  - FALCON: Compact signatures, fast verification, strong security proofs.
  - ML-DSA: Slightly larger sizes, avoids floating-point arithmetic, easy to implement, fast performance.
  - SPHINCS+: Relies only on hash functions, maximally conservative, stateless, no algebraic assumptions.

Objective One Conclusion

- Downside: Very large signatures, slow signing, especially in Robust variant.
- Simple variant improves performance but still much slower than the others.
- If lattice schemes are broken, SPHINCS+ is the only standardized non-algebraic fallback.
    - Calls for continued research into alternative non-lattice digital signatures.

Objective 2: Development and Evaluation of SPHINCS+ Hybrid Scheme

Challenges with Pure-PQ Designs

- Replacing Classical Cryptography with PQ Algorithms is infeasible for systems like Vehicle-to-Vehicle communication.
- On going side channel attacks for PQC Algorithms.
- NIST has stated, “..utilization of hybrid algorithms (combinations of classical quantum-vulnerable and quantum-resistant public-key algorithms)” [2].

Hybrid Setup

Hybrid Key Generation:

$$\begin{aligned} sk_1 &\leftarrow \mathrm{KeyGen}_{\mathrm{classical}}(), \quad pk_1 = \mathrm{derive}(sk_1), \\ sk_2 &\leftarrow \mathrm{KeyGen}_{\mathrm{pqc}}(), \quad\quad pk_2 = \mathrm{derive}(sk_2) \end{aligned}$$

Hybrid Signing:

$$\begin{aligned} \sigma_1 &= \mathrm{Sign}_{\mathrm{classical}}(sk_1, m), \\ \sigma_2 &= \mathrm{Sign}_{\mathrm{pqc}}(sk_2, m) \end{aligned} \quad \Rightarrow \quad \sigma = (\sigma_1, \sigma_2)$$

Hybrid Verification:

$$\begin{aligned} \mathrm{Verify}_{\mathrm{classical}}(pk_1, m, \sigma_1) &\rightarrow \mathrm{accept/reject}, \\ \mathrm{Verify}_{\mathrm{pqc}}(pk_2, m, \sigma_2) &\rightarrow \mathrm{accept/reject} \end{aligned}$$

$$\mathrm{Valid} = \mathrm{Verify}_{\mathrm{classical}} \wedge \mathrm{Verify}_{\mathrm{pqc}}$$

Hybrid Security

$$\text{Valid} = \text{Verify}_{\text{classical}} \wedge \text{Verify}_{\text{pqc}}$$

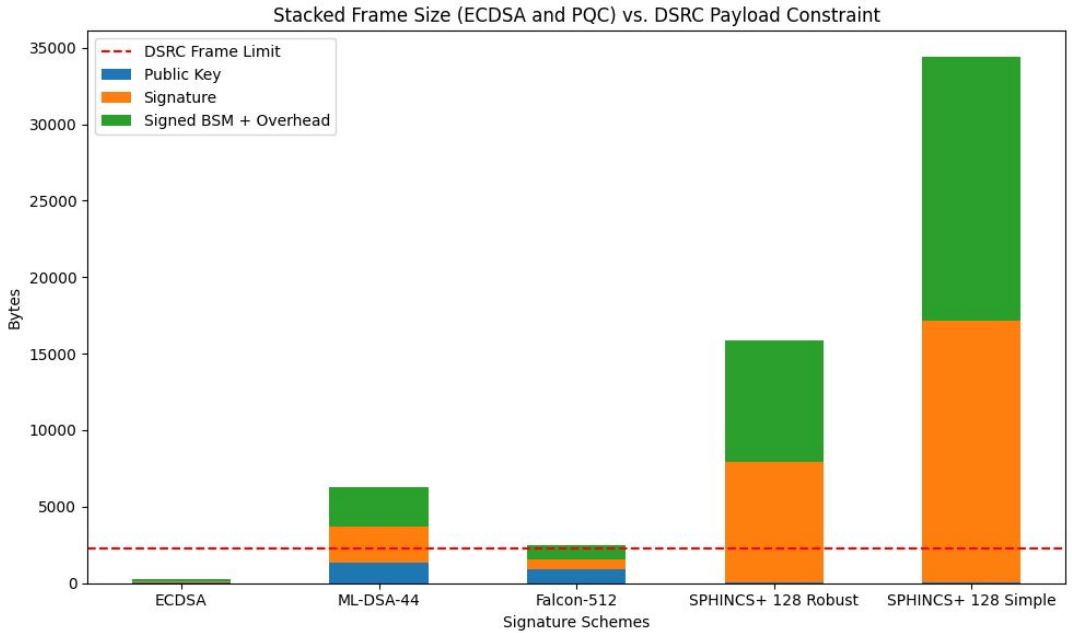
- Hybrid schemes require both classical and post-quantum signatures to verify, ensuring layered security.
- Failure of one breaks all: If either component is broken, the entire hybrid signature becomes invalid.
- Transition-ready design: Once a component is deprecated, systems must migrate to a fully post-quantum scheme to maintain trust.

Real-Time Requirements for V2V Communication [6]

- ✓ Vehicles exchange Basic Safety Messages (BSMs) regularly
- ✓ Each BSM must be verified cryptographically
  - In practice: up to 100 vehicles in range
- ✓ To prevent delays:
  - Must verify ≥ 1 signature per vehicle per 100 ms
- ✓ Digital signatures in V2V use elliptic curves (ECDSA)

Pure-PQ in V2V Reality [6]

- Dedicated Short-Range Communication (DSRC) → 2,304-byte limit
- Secure Protocol Data Unit (SPDU) contains the certificate and signed BSM



PQ Approach [6]

Scheme	Avg. Verify	v_{max}	Acceptable?
ECDSA	0.000157	636942	Yes
ML-DSA-44	0.030355	3294	Yes
FALCON-512	0.050803	1968	Yes
SPHINCS+ Simple	0.534464	187	Yes*
SPHINCS+ Robust	0.226387	431	Yes*

Scheme	Avg. Verify	v_{max}	Acceptable?
ECDSA	0.001	67521	Yes
ML-DSA-44	0.189	529	Yes
FALCON-512	0.446	224	Yes
SPHINCS+ Simple	-	-	-
SPHINCS+ Robust	5.436	18	No

Objective Two: Conclusion

- Hybrid schemes can offer layered post-quantum security.
- Drawback: Signature size becomes much larger therefore timing gets slower.
- SPHINCS+ alone exceeds IEEE 1609.2 2,304-byte V2V frame limit.
- Fails verification time benchmarks in hardware-constrained V2X environments.
- Urgent need for a compact, non-lattice-based fallback.

Objective 3:
Applicability of SPHINCS+ in Blockchain

Blockchain Overview

- Blockchain records transactions across a peer-to-peer network.
- Each block links to the previous one via cryptographic hashes.
- Bitcoin uses Proof of Work (PoW) and digital signatures to prevent double-spending and enable secure transactions.
  - Miners solve computational puzzles to validate blocks and secure the network.

Bitcoin Transaction Processes Simplified

- Bitcoin uses ECDSA for transaction signing
- Signs a double SHA-256 hash of the serialized transaction
  - The sender uses their private key to sign the hash. This proves ownership of the referenced unspent transaction outputs (UTXO).
- The recipient or miners use the public key to verify that the signature matches the hash and the UTXO being spent.

QC Threat to Bitcoin

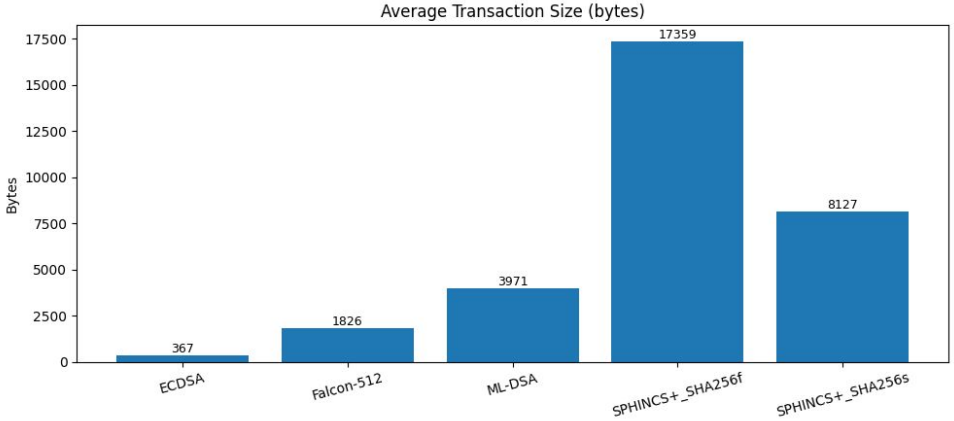
- Bitcoin’s reliance on ECDSA makes it vulnerable to quantum attacks.
  - Pay-to-Public-Key exposes the full public key on-chain at creation.
  - Pay-to-Public-Key Hash keeps the public key hidden until the UTXO is spent.
- Proposed Framework:
  - Uses quantum-resistant signatures instead of ECDSA.
  - Users sign transactions with a PQC private key.
  - Miners verify PQC signatures before adding blocks to the chain.

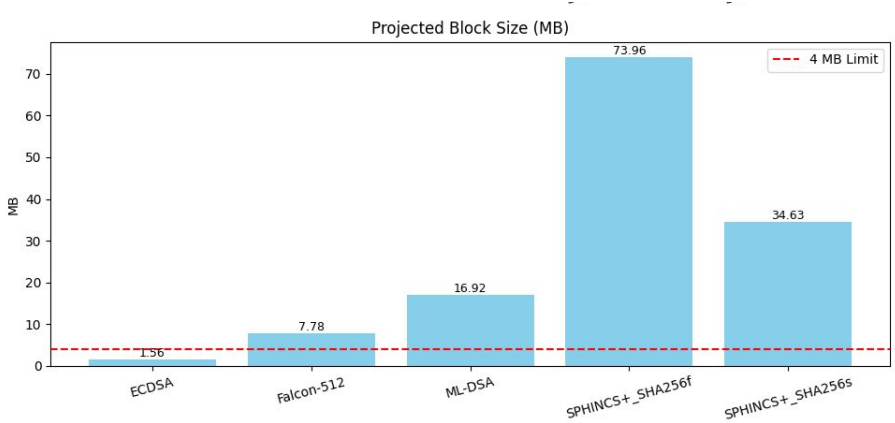
Size Impact

- Used Bitcoin.org datasets from May 6, 2025 for average block size and transaction count.
- Simulated replacing ECDSA with PQC signatures to assess transaction and block size impact.
- Calculated new transaction sizes, projected block sizes, and estimated transaction per block.

Size Impact to transactions and block

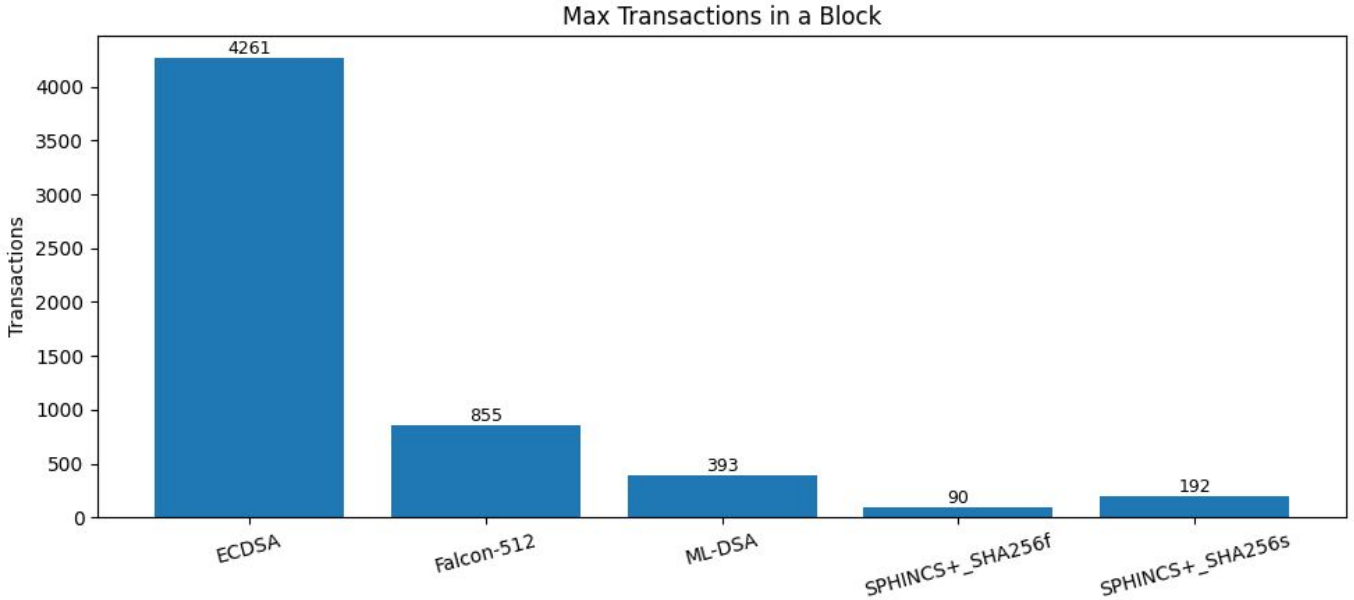
Smaller bars are better in the case of transaction size and block size





Size Impact to # of Transactions

Larger bars are better in the case of max transactions



Bitcoin Transaction Fees

- At a fee rate of 6 satoshis per byte, the estimated fees for various signature schemes are:
  - ECDSA (367 bytes): 2,202 satoshis (0.000022 BTC) ($2.28)
  - FALCON-512 (1,826 bytes): 10,956 satoshis (0.00011 BTC) ($11.37)
  - ML-DSA-44 (3,971 bytes): 23,826 satoshis (0.00024 BTC) ($24.72)
  - SPHINCS+ SHA256s (8,127 bytes): 48,762 satoshis (0.00049 BTC) ($50.59)
  - SPHINCS+ SHA256f (17,359 bytes): 104,154 satoshis (0.001 BTC) ($108.05)

Mock Setup Design Overview

- ✓ Setup
  1. Generate key pairs
  2. Derive addresses via RIPEMD-160(SHA-256(pubkey))
  3. Hash transaction using double SHA-256
  4. Sign and verify transactions
- ✓ Metrics Collected:
  - Key generation and serialization time
  - Signing and verification time

Bitcoin Transaction Signing

Step	ECDSA	ML-DSA	FALCON	SPHINCS+s	SPHINCS+f
Key Pair	0.079	0.102	5.775	14.638	0.427
Signing	0.108	0.262	0.169	202.362	9.644
TX Serialization	0.012	0.097	0.057	0.259	0.529
Verification	0.151	0.087	0.028	0.211	0.581

Step	SPHINCS+s Hybrid	SPHINCS+f Hybrid	FALCON Hybrid	ML-DSA Hybrid
Key Pair	17.347	0.647	7.422	0.178
Signing	205.352	10.663	0.261	0.331
TX Serialization	0.264	0.634	0.061	0.107
Verification	0.359	0.725	0.166	0.225

Objective Three Conclusion

- ✓ Post-Quantum signatures are essential for blockchain security
- ✓ SPHINCS+ is not practical for Bitcoin
  - Fees can exceed $50 per transaction, making micro-transactions infeasible.
  - Violates the 4 MB SegWit limit, over 95% drop in transactions per block.
- ✓ Still an urgent need for a compact, non-lattice-based post-quantum scheme.

Objective 4: Feasibility of SPHINCS+

TLS, C-V2X

C-V2X Network

- Cellular Vehicle-to-Everything imposes stricter timing and size constraints than DSRC-based V2V.
- A maximum 437-byte payload.
- No current PQC scheme is suitable for C-V2X without major redesign.
- SPHINCS+ is not practicable in **all** areas of vehicle communication.

Transport Layer Security (TLS) Handshake

https://www.thesslstore.com/blog/tls-1-3-handshake-tls-1-2/

TSL

Algorithm	Avg. Time	Certificate Size
RSA-3072	9.2187	1.4 KB
ML-DSA-44	5.829	5.3 KB
FALCON-512	5.753	2.4 KB
SPHINCS+ Simple	38.892	22.9 KB
RSA-3072 & SPHINCS+ Simple	46.019	24.0 KB
RSA-3072 & ML-DSA-44	10.856	6.0 KB
RSA-3072 & FALCON	11.857	3.4 KB

Objective 4: Conclusion

- ✓ SPHINCS+ is infeasible for C-V2X due to high latency and large signature sizes, which risk missing strict communication windows.
- ✓ TLS 1.3 provides a more forgiving environment.
  - SPHINCS+ still introduces notable handshake delay.

9 Future Work

Future Work

- ✓ Evaluate PQC in C-V2X Context
  - Address stricter bandwidth and timing constraints than DSRC.
  - Investigate signature compression or fragmentation for feasibility.
- ✓ Integrate PQC into Bitcoin Core
  - Modify transaction structure for large PQC signatures.
  - Focus on minimal-disruption schemes like FALCON.

Future Work

- ✓ Mitigate ECDSA Legacy Risks
  - Identify and flag vulnerable addresses for quantum-safe migration.
  - Propose key rotation protocols to phase out ECDSA over time.
- ✓ Hardware Acceleration for SPHINCS+
  - Leverage GPU optimizations for non-latency-sensitive applications.
  - Assess real-world deployment and further performance tuning.

7

Conclusion

SPHINCS+ Conclusion

Conclusion

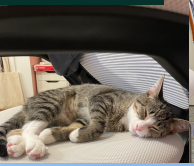
- ✓ Does not reach the performance compared to lattice-based schemes like FALCON or ML-DSA
  - SPHINCS+ signature size is relatively **large**, which results in a **slower** signing.
  - Level 1 security already very slow, increasing to Level 3 & 5 will only increase this slowness.

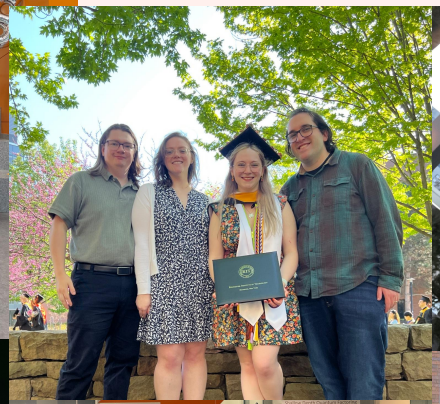
Conclusion

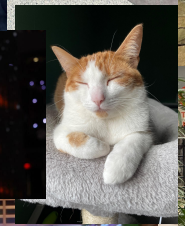
- ✓ NIST deemed SPHINCS+ to be a **backup plan** if lattices are eventually targeted.
  - But a backup plan should still be feasible for all systems.
- ✓ NIST has reopened the Digital Signature Proposals
  - Urges they not be based on lattices
  - Interested in signature schemes that have short signatures and fast verification

Waygate Technologies

Acknowledgments










Thank You!

Any Questions?

Resources

- **[1]** National Institute of Standards and Technology, "Stateless Hash-Based Digital Signature Standard," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST FIPS 205 ipd, Aug. 2023. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.ipd.pdf
- **[2]** Barker, W., Polk, W., & Souppaya, M. (2021). Getting Ready for Post-Quantum Cryptography: Exploring Challenges Associated with Adopting and Using Post-Quantum Cryptographic Algorithms. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.CSWP.04282021
- **[3]** D. J. Bernstein, A. H ülsing, S. K ölbl, T. Lange, R. Niederhagen, J. Rijneveld, P. Schwabe, et al., "SPHINCS+," in Submission to the NIST Post-Quantum Project, v.3.1, Jun. 2022, pp. 2129–2146. [Online]. Available: https://sphincs.org/data/sphincs+-r3.1-specification.pdf
- **[4]** D. J. Bernstein, A. H ülsing, S. K ölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The SPHINCS + Signature Framework," in Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. London United Kingdom: ACM, Nov. 2019, pp. 2129–2146. [Online]. Available: https://dl.acm.org/doi/10.1145/3319535.3363229
- **[5]** D. Stinson and M. Paterson, Cryptography: Theory and Practice, ser. Textbooks in Mathematics. CRC Press, 2018. [Online]. Available: https://books.google.com/books?id=nHxqDwAAQBAJ
- **[6]** Geoff Twardokus, Nina Bindel, Hanif Rahbari, and Sarah McCarthy. When Cryptography Needs a Hand: Practical Post-Quantum Authentication for V2V Communications. Presented at the Network and Distributed System Security (NDSS) Symposium 2024, San Diego, CA, USA, February 26 – March 1, 2024. https://doi.org/10.14722/ndss.2024.24267
- **[7]** VegeBun, "How do hash-based post-quantum digital signatures work? (Part 1)," Dec. 2022. [Online]. Available: https://research.dorahacks.io/2022/12/16/hash-based-post-quantum-signatures-1/
- **[8]** VegeBun, "How do hash-based post-quantum digital signatures work? (Part 2)," Dec. 2022. [Online]. Available: https://research.dorahacks.io/2022/12/16/hash-based-post-quantum-signatures-2/
- **[9]** Yehia, M. Hash-based signatures revisited: A dynamic fors with adaptive chosen message security. International Conference on Cryptology 2020. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7334985/

8 Extra Material

For questions

Digital Signature On Ramp Round 2 Candidates

Code-Based	**Lattice-Based**	**MPC-in-the-Head**	**Multivariate**
CROSS	Hawk	Mirath (MIRA/MiRiTH)	UOV
LESS		MQOM	MAYO
		PERK	QR-UOV
Symmetric-Based	**Isogeny-Based**	RYDE	SNOVA
FAEST	SQIsign	SDitH	

Computer Specs

- ✓ Objective 2:
  - x86-64 CPU w/ 6 Cores
  - 32 GB of RAM
- ✓ Objective 1, 3, 4
  - ARM CPU w/ 8 Core
  - 16 GB of RAM

Hybrid Size Comparison

Method	Public Key	Private Key	Signature	Type of Scheme	Security
ML-DSA-44	1,376	2,592	2,484	Lattice	2 (128-bit)
ML-DSA-65	2,016	4,064	3,373	Lattice	3 (192-bit)
ML-DSA-87	2,656	4,928	4,691	Lattice	5 (256-bit)
FALCON-512	961	1,313	754	Lattice	1 (128-bit)
FALCON 1024	1,857	2,337	1,394	Lattice	5 (256-bit)
SLH-DSA-SHA2-128f Simple	96	96	17,152	Hash-based	1 (128-bit)
SLH-DSA-SHA2-128s Robust	96	96	7,920	Hash-based	1 (128-bit)
SLH-DSA-SHA2-192f Simple	112	128	35,728	Hash-based	3 (192-bit)
SLH-DSA-SHA2-192s Robust	112	128	16,288	Hash-based	3 (192-bit)
SLH-DSA-SHA2-256f Simple	128	160	49,920	Hash-based	5 (256-bit)
SLH-DSA-SHA2-256s Robust	128	160	29,856	Hash-based	5 (256-bit)

Hybrid Time Comparison

Scheme	Key Gen	Sign	Verify
ECDSA + SPHINCS+ Robust	16.256	205.919	0.371659
ECDSA + SPHINCS+ Simple	0.607468	10.79311	0.734677
ECDSA + FALCON-512	6.35757	0.285816	0.18719
ECDSA + ML-DSA-44	0.17314	0.354733	0.235762

TSL w/ HTTP

Algorithm	Avg. Time	Certificate Size
RSA-3072	9.166	1.4 KB
ML-DSA-44	5.551	5.3 KB
FALCON-512	6.152	2.4 KB
SPHINCS+ Simple	36.205	22.9 KB
RSA-3072 & SPHINCS+ Simple	47.892	24.0 KB
RSA-3072 & ML-DSA-44	10.296	6.0 KB
RSA-3072 & FALCON	11.714	3.4 KB

Qubits and QC

- Physical Qubit - Actual hardware quibt
- Logical Qubit - A “virtual” qubit made by combining physical qubits to form a stable error corrected qubit
- A quantum computer capable of breaking modern cryptographic schemes must be general-purpose, large-scale, and fault-tolerant.

Hash Function Properties

- ✓ Collision Resistant
- ✓ Preimage Resistant
- ✓ Second Preimage Resistant
- ✓ Efficient
- ✓ Arbitrary Message Size
- ✓ Fixed Output Size

SPHINCS+ Parameter Sets

	n	h	d	h'	a	k	lg_w	m	sec level	pk bytes	sig bytes
SLH-DSA-SHA2-128s SLH-DSA-SHAKE-128s	16	63	7	9	12	14	4	30	1	32	7 856
SLH-DSA-SHA2-128f SLH-DSA-SHAKE-128f	16	66	22	3	6	33	4	34	1	32	17 088
SLH-DSA-SHA2-192s SLH-DSA-SHAKE-192s	24	63	7	9	14	17	4	39	3	48	16 224
SLH-DSA-SHA2-192f SLH-DSA-SHAKE-192f	24	66	22	3	8	33	4	42	3	48	35 664
SLH-DSA-SHA2-256s SLH-DSA-SHAKE-256s	32	64	8	8	14	22	4	47	5	64	29 792
SLH-DSA-SHA2-256f SLH-DSA-SHAKE-256f	32	68	17	4	9	35	4	49	5	64	49 856

[1] https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.ipd.pdf

SPHINCS+ Parameter Sets

Table 1: Overview of the number of function calls we require for each operation. We omit the single calls to $\mathbf{H_{msg}}, \mathbf{PRF_{msg}}$, and $\mathbf{T}_k$ for signing and single calls to $\mathbf{H_{msg}}$ and $\mathbf{T}_k$ for verification as they are negligible when estimating speed.

	F	**H**	**PRF**	$T_{\texttt{len}}$
Key Generation	$2^{h/d}w\texttt{len}$	$2^{h/d}-1$	$2^{h/d}\texttt{len}$	$2^{h/d}$
Signing	$kt+d(2^{h/d})w\texttt{len}$	$k(t-1)+d(2^{h/d}-1)$	$kt+d(2^{h/d})\texttt{len}$	$d2^{h/d}$
Verification	$k+dw\texttt{len}$	$k\log t+h$	–	d

[1] https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.ipd.pdf

Tweakable Hash Function

- Tweakable hash functions (Th) ensure domain separation in SPHINCS+ by incorporating structured tweaks (ADRS) and public seed values.
- Each hash call is uniquely bound to its role (WOTS+, FORS, Merkle) to prevent collisions and enable modular security.

Bitmask

$$M^{\oplus} = M \oplus \text{MGF1-SHA-256}(\mathbf{PK.seed}||\mathbf{ADRS}^c, l).$$

$$\mathbf{F}(\mathbf{PK.seed}, \mathbf{ADRS}, M_1) = \text{SHA-256}(\mathbf{PK.seed}||\text{toByte}(0, 64-n)||\mathbf{ADRS}^c||M_1^{\oplus}).$$

$$\mathbf{F}(\mathbf{PK.seed}, \mathbf{ADRS}, M_1) = \text{SHA-256}(\mathbf{PK.seed}||\text{toByte}(0, 64-n)||\mathbf{ADRS}^c||M_1)$$

SHA-512 Categories

- H_{msg}, which is used to generate the digest of the message being signed.
- PRF_{msg}, which is the pseudo-random function, PRF, that generates the randomizer, R, for the randomized hashing when the message is being signed.
- T_l, which is the hash function that maps the $ln - byte$ message to an $n - byte$ message.
- H, which takes a $2n - byte$ message and behaves similar to T_l.

Background - OTS

- A signature scheme is a one-time signature scheme if it is secure when **ONLY ONE** message is signed (can be verified any number of times)
- The first scheme of this type that we'll discuss is the Lamport Signature Scheme first published in 1979

Background – Lamport OTS Example

Example: 7879 is prime and 3 is a primitive element in $\mathbb{Z}_{7879}$

$$f(x) = 3^x \pmod{7879}$$

$$\begin{aligned} y_{1,0} &= 5831 & z_{1,0} &= 2009 \\ y_{1,1} &= 735 & z_{1,1} &= 3810 \\ y_{2,0} &= 803 & z_{2,0} &= 4672 \\ y_{2,1} &= 2467 & z_{2,1} &= 4721 \\ y_{3,0} &= 4285 & z_{3,0} &= 268 \\ y_{3,1} &= 6449 & z_{3,1} &= 5731 \end{aligned}$$

Message to sign: $x = (1, 1, 0)$
Signature: $(y_{1,1}, y_{2,1}, y_{3,0})$ = $(735, 2467, 4285)$
Verify the signature:

$$\begin{aligned} 3^{735} \pmod{7879} &= 3810 \\ 3^{2467} \pmod{7879} &= 4721 \\ 3^{4285} \pmod{7879} &= 268 \end{aligned}$$

[4] https://books.google.com/books?id=nHxqDwAAQBAJ

Background – WOTS/WOTS+

- The Winternitz Signature Scheme provides a significant reduction in key size with the use of a hash function
- Winternitz Parameter w
- WOTS vs WOTS+
  - Introduces Randomization

Background – WOTS Example

Example: Suppose that k=9, l=3, w=3. Three hash chains:

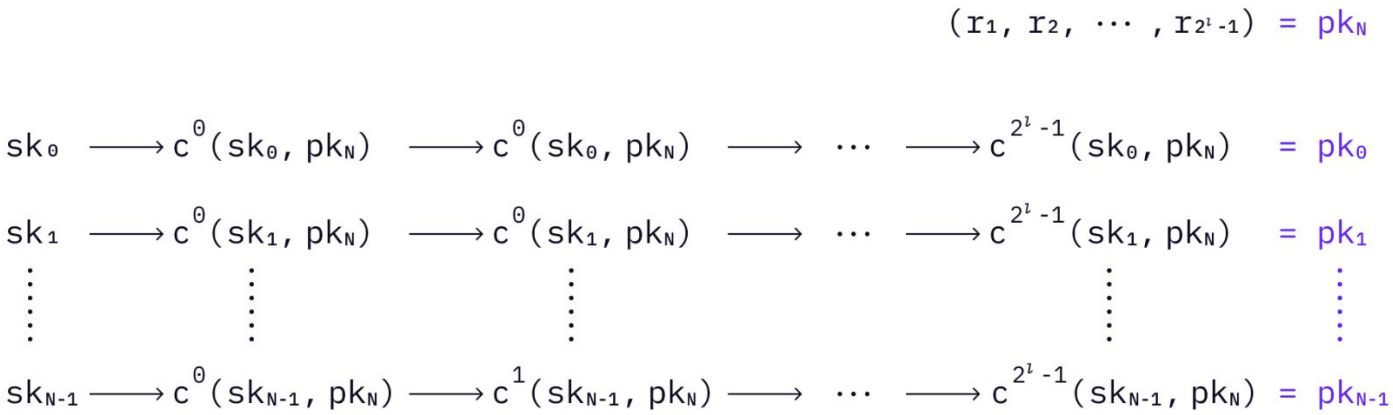
$$y_1^0 \to y_1^1 \to y_1^2 \to \boxed{y_1^3} \to y_1^4 \to y_1^5 \to y_1^6 \to y_1^7 \to z_1$$
$$y_2^0 \to y_2^1 \to y_2^2 \to y_2^3 \to y_2^4 \to \boxed{y_2^5} \to y_2^6 \to y_2^7 \to z_2$$
$$y_3^0 \to \boxed{y_3^1} \to y_3^2 \to y_3^3 \to y_3^4 \to y_3^5 \to y_3^6 \to y_3^7 \to z_3$$

Message to sign: 011101001
Signature: $x_1 = 011 = 3, x_2 = 101 = 5, x_3 = 001 = 1$
Released values: $a_1 = y_1^3, a_2 = y_2^5, a_3 = y_3^1$
Verify the signature:

$$f^5(a_1) = z_1$$
$$f^3(a_2) = z_2$$
$$f^7(a_3) = z_3$$

[4] https://books.google.com/books?id=nHxqDwAAQBAJ

WOTS+ Chain



https://research.dorahacks.io/2022/10/26/hash-based-post-quantum-signatures-1/

WOTS+ Chain

- Just like in the above example, WOTS+ also has a chaining function
- Tweakable hash function in red

Algorithm 4 chain(X, i, s, **PK**.seed, **ADRS**)

Chaining function used in WOTS$^+$.

Input: Input string X, start index i, number of steps s, public seed **PK**.seed, address **ADRS**.
Output: Value of **F** iterated s times on X.

```
1: if (i+s) ≥ w then
2:     return NULL
3: end if
4:
5: tmp ← X
6:
7: for j from i to i+s−1 do
8:     ADRS.setHashAddress(j)
9:     tmp ← F(PK.seed, ADRS, tmp)
10: end for
11: return tmp
```

WOTS+ Public Key

- Input: Secret Seed, public seed, and address
- Outputs: Compressed WOTS+ public key

Algorithm 5 wots_PKgen(**SK**.seed, **PK**.seed, **ADRS**)

Generate a WOTS$^+$ public key.

Input: Secret seed **SK**.seed, public seed **PK**.seed, address **ADRS**.
Output: WOTS$^+$ public key pk.

```
skADRS ← ADRS                          ▷ Copy address to create key generation key address
skADRS.setTypeAndClear(WOTS_PRF)
skADRS.setKeyPairAddress(ADRS.getKeyPairAddress())
for i from 0 to len − 1 do
    skADRS.setChainAddress(i)
    sk ← PRF(PK.seed, SK.seed, skADRS)          ▷ Compute secret value for chain i
    ADRS.setChainAddress(i)
    tmp[i] ← chain(sk, 0, w − 1, PK.seed, ADRS)  ▷ Compute public value for chain i
end for
wotspkADRS ← ADRS                      ▷ Copy address to create WOTS+public key address
wotspkADRS.setTypeAndClear(WOTS_PK)
wotspkADRS.setKeyPairAddress(ADRS.getKeyPairAddress())
pk ← T_len(PK.seed, wotspkADRS, tmp)                          ▷ Compress public key
return pk
```

WOTS+ Signing

- WOTS+ Signature is core for the XMSS
- These will sign the FORS signature and the root nodes of each binary tree used
- Input: Message, secret seed, public seed and address
- Output: WOTS+ signature

Algorithm 6 wots_sign(M, **SK**.seed, **PK**.seed, **ADRS**)

Generate a WOTS$^+$ signature on an n-byte message.

Input: Message M, secret seed **SK**.seed, public seed **PK**.seed, address **ADRS**.
Output: WOTS$^+$ signature sig.

$csum \leftarrow 0$

$msg \leftarrow \text{base\_2}^b(M, lg_w, len_1)$ ▷ Convert message to base w

for i **from** 0 **to** $len_1 - 1$ **do** ▷ Compute checksum
  $csum \leftarrow csum + w - 1 - msg[i]$
end for

$csum \leftarrow csum \ll ((8 - ((len_2 \cdot lg_w) \bmod 8)) \bmod 8)$ ▷ For $lg_w = 4$ left shift by 4

$msg \leftarrow msg \parallel \text{base\_2}^b\left(\text{toByte}\left(csum, \left\lceil \frac{len_2 \cdot lg_w}{8} \right\rceil\right), lg_w, len_2\right)$ ▷ Convert $csum$ to base w

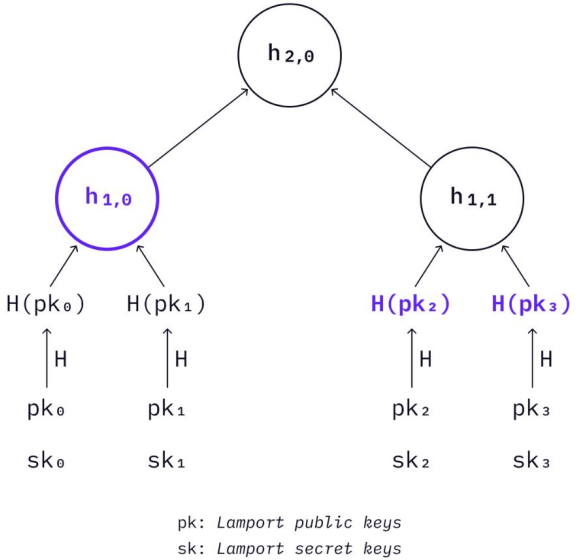
skADRS $\leftarrow$ **ADRS**
skADRS.setTypeAndClear(WOTS_PRF)
skADRS.setKeyPairAddress(ADRS.getKeyPairAddress())
for i **from** 0 **to** $len - 1$ **do**
  skADRS.setChainAddress(i)
  $sk \leftarrow$ **PRF**(**PK**.seed, **SK**.seed, skADRS) ▷ Compute secret value for chain i
  ADRS.setChainAddress(i)
  $sig[i] \leftarrow \text{chain}(sk, 0, msg[i], \textbf{PK}.\text{seed}, \textbf{ADRS})$ ▷ Compute signature value for chain i
end for
return sig

Background – Merkle Trees (MSS)

- Merkle extends this idea of using **one-way hash functions** like Lamport OTS and WOTS to construct a tree of digital signatures
- Use a binary tree (Merkle tree) to create combinations of various public keys of one-time signature schemes
  - **Larger amount of signatures** can be used without increasing the public key size

Background – Merkle Trees (MSS)

- A merkle tree of height h has 2^h leaves (digests of Lamport key pairs)
- When signing a message, a **pair of Lamport keys** are chosen that have not been used before
- **Stateful** signature scheme!



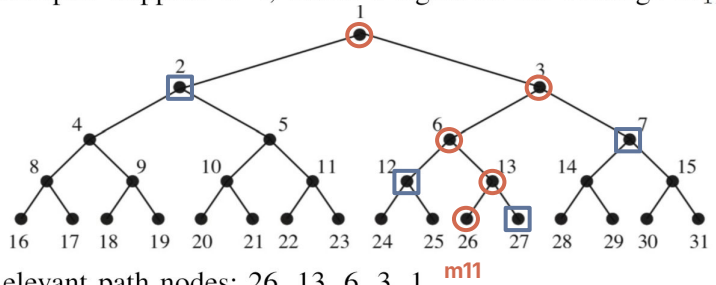
In this simple Merkle tree, a message signed by sk_2 is $\sigma = (2, auth_2, \sigma_2, pk_2)$, where $auth_2 = (h(pk_3), h_{1,0})$, and σ_2 is half of sk_2

[5] https://research.dorahacks.io/2022/10/26/hash-based-post-quantum-signatures-1/

Background – MSS Example

- You can save only the root, reducing the space from O(2^h) to O(1)
- Signature must include the **authentication path** for verification
- Verify by hashing your way up the tree to see if it equals the public key

Example: Suppose d=4, create a signature for message m_{11}:



Relevant path nodes: 26, 13, 6, 3, 1
Sibling nodes: V(27), V(12), V(7), V(2)

Authenticate key K_{11}:

1) Compute $V(26) = h(K_{11})$
2) Compute $V(13) = h(V(26)||V(27))$
3) Compute $V(6) = h(V(12)||V(13))$
4) Compute $V(3) = h(V(6)||V(7))$
5) Compute $V(1) = h(V(2)||V(3))$
6) Verify that $V(1) = K$

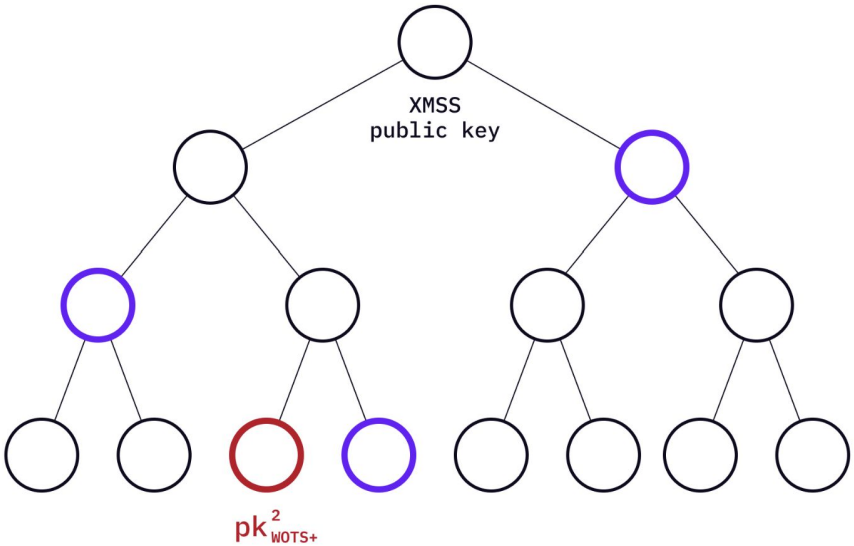
Signature: $K_{11}, s_{11}, V(27), V(12), V(7), V(2)$

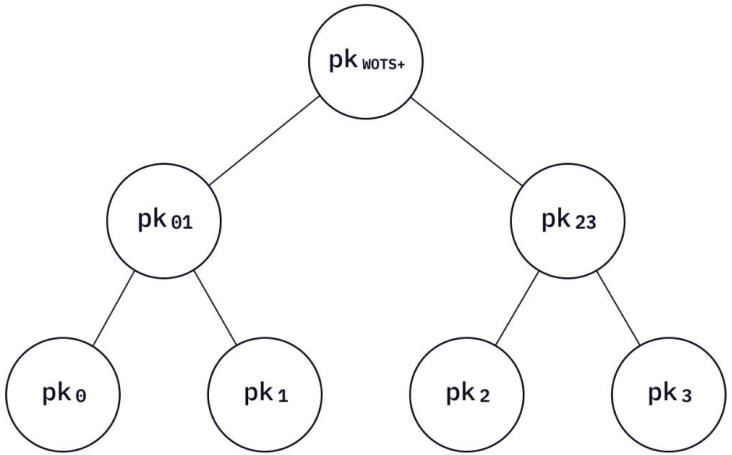
[4] https://books.google.com/books?id=nHxqDwAAQBAJ

Background – XMSS

- Uses a Merkle tree to manage **WOTS+ keys** similar to how MSS uses a Merkle tree to manage Lamport OTS keys
- The root is the public key and each leaf node is a WOTS+ public key, comprised of a key set
- Contains a “sub-tree” (L-tree) where each node is XOR'd with a bitmask value to incorporate collision resistance

Background – XMSS

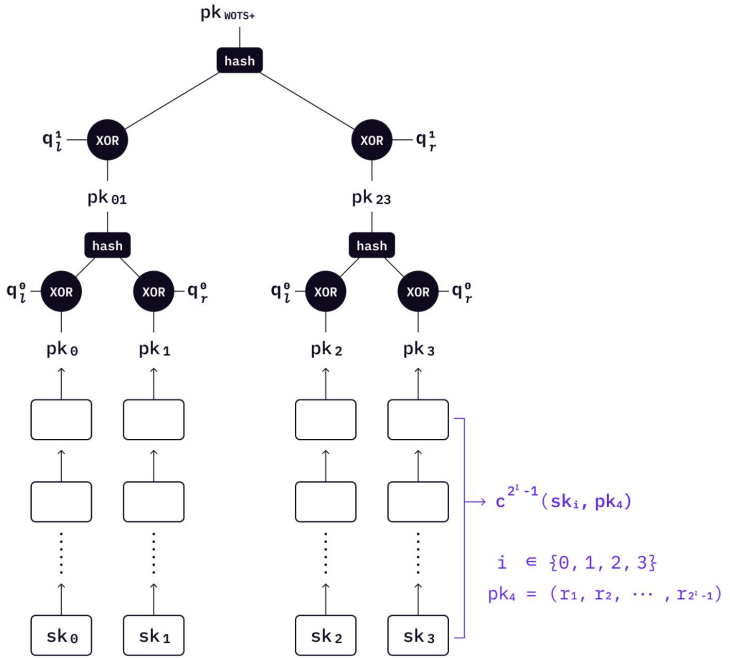




[5] https://research.dorahacks.io/2022/10/26/hash-based-post-quantum-signatures-1/

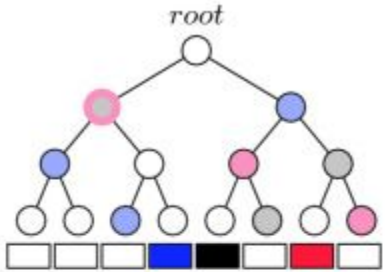
Background - XMSS

- To sign a message, a **WOTS+ public key** that has <u>not</u> been used is chosen (still stateful!)
- $\sigma = (\sigma, auth, i, leaf)$
- Verify by completing the hash chains and hashing up the tree to see if it equals the public key

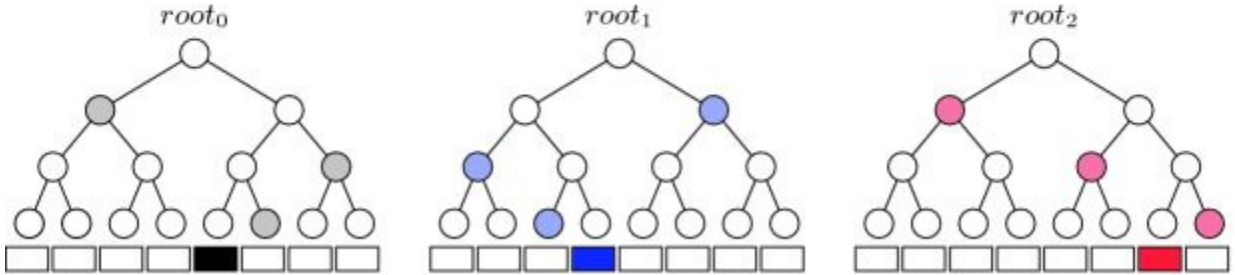


[5] https://research.dorahacks.io/2022/10/26/hash-based-post-quantum-signatures-1/

HORST vs FORS



(a) HORS signature within a binary tree construction



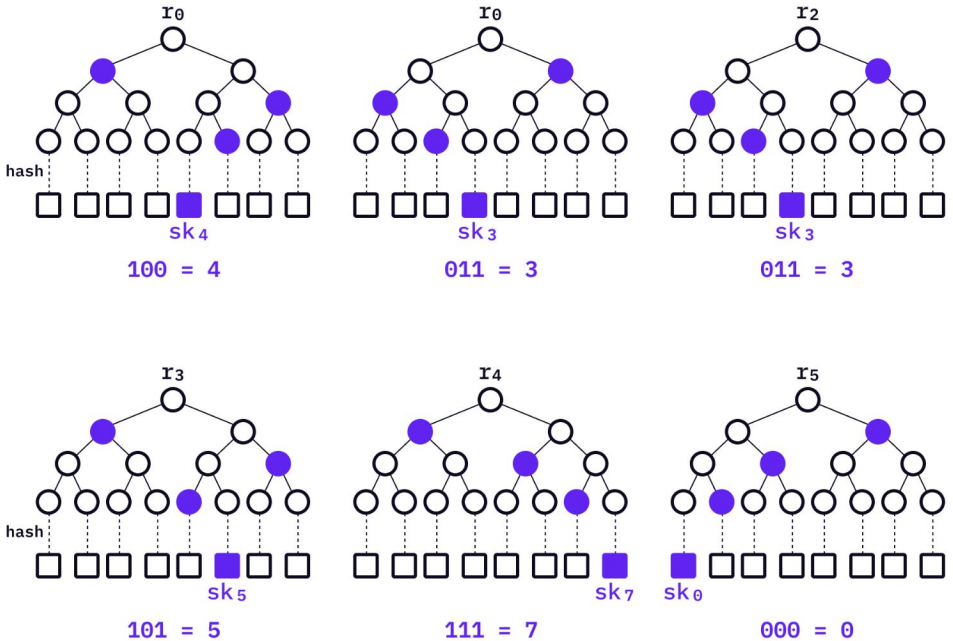
(b) FORS signature within κ binary trees construction

[7] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7334985/

Background - FORS

- ✓ Forest of Random Subsets (FORS)
- ✓ Few-time signature scheme
- ✓ Improvement of HORS (SPHINCS)
  - Hash to Obtain Random Subset
  - Main difference between them is that each tree in FORS has its own root

Background – FORS

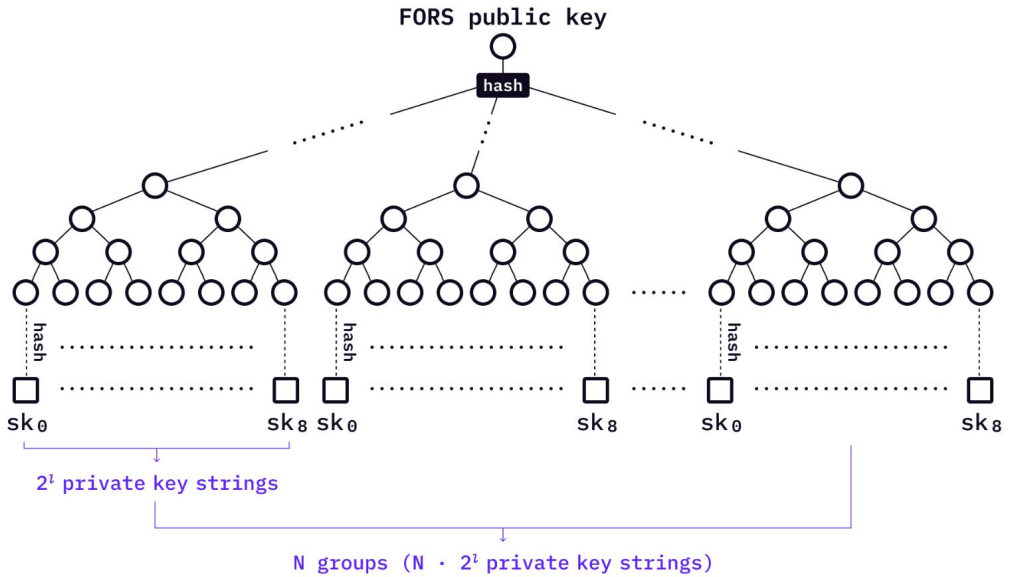


$$\sigma = (sk_4^1, auth_4^1, sk_3^2, auth_3^2, sk_3^3, auth_3^3, sk_5^4, auth_5^4, sk_7^5, auth_7^5, sk_0^6, auth_0^6)$$

[6] https://research.dorahacks.io/2022/12/16/hash-based-post-quantum-signatures-2/

Background - FORS

- Each root is hashed together to create the FORS public key
- Verify - reconstruct each **root node** using the auth path and hash all the roots to compare with the FORS public key



$$\sigma = (sk_4^1, auth_4^1, sk_3^2, auth_3^2, sk_3^3, auth_3^3, sk_5^4, auth_5^4, sk_7^5, auth_7^5, sk_0^6, auth_0^6)$$

[6] https://research.dorahacks.io/2022/12/16/hash-based-post-quantum-signatures-2/

Background - The Hypertree

- To make unlikely that a random selection of a leaf node results in the same node being selected, a SPHINCS tree needs to be quite large
- Rather than increasing the height of an XMSS tree, the authors of SPHINCS+ decide to create a **hypertree** of XMSS trees