Breaking Category Five SPHINCS$^+$ with SHA-256

Ray Perlner[1], John Kelsey[1,2], and David Cooper[1]

1. NIST, 2. COSIC/KU Leuven

Summary of Result

- SPHINCS$^+$ is a stateless hash-based signature selected for standardization by NIST
- We present a forgery attack that reduces classical security by 40 bits
  - For submitted parameter sets:
    - That target Category 5
    - While using SHA-256
- Our attack builds on a previous attack by Antonov on the DM-SPR property of SHA-256 (a security assumption for SPHINCS$^+$)
- The SPHINCS$^+$ team has proposed a tweak which defeats our attack by using SHA-512 instead of SHA-256 (where necessary)

Outline

- DM-SPR Property and Antonov's Attack
- Using Antonov's Attack to Forge $WOTS^+$ (This Paper)
- Optimizations (This Paper)
- The $SPHINCS^+$ Tweak
- Conclusion

Prefixes and *Distinct Function Multitarget Preimage Resistance* (DM-SPR)

- Many places in hypertree where a preimage can create a forgery:
  - Hashes in Merkle Trees
  - Hash Chains in $WOTS^+$
  - Hash trees in FORS
  - FORS public key hash
  - $WOTS^+$ public key hash (Our attack here)
- New targets are revealed with every honest signature
- To avoid a 1 out of t multi-target preimage attack:
  - Make sure hash input at each hypertree location has a distinct prefix
  - Formalized as a tweakable hash function with DM-SPR property

Antonov's Attack on SHA-256 DM-SPR [Antonov 2022]

- Collect t target hashes with different prefixes
- Find preimage with the same prefix for 1 of them
  - Use Herding to reach same state from all prefixes at the penultimate block
  - Use Multi-Target preimage search on compression function to find a block to append and reach a target
- Longest hash input in SPHINCS$^+$ is WOTS$^+$ public key hash
- That's still pretty short (34 blocks)
  - To balance cost of herding, multi-target preimage search, use some compression-function 3-collisions
    - Let t be $2^{10}3^{23} \approx 2^{46}$ instead of 2^{33}
    - 3-Collision search cost: $1.5 \cdot 3^{23} \cdot 2^{170.7} \approx 2^{208}$
    - Multi-Target preimage cost: $2^{256}/2^{46} \approx 2^{210}$

What's Left to Do?

- Antonov's attack lets us create a validly-signed WOTS$^+$ public key preimage
- But we need to know the corresponding private key to forge a SPHINCS$^+$ signature
  - This involves knowing preimages of parts of WOTS$^+$ public key
  - For validity, prefix must match hypertree location
  - But hypertree location depends which target we reached
  - No way to force correct prefix for all targets
- Or at least part of it…
  - As long as we can sign more than one possible digest with our WOTS$^+$ key
  - Can graft a forged Merkle-Tree root to the hypertree for less than 2^{256} work!

Our Attack: Outline

- Find a preimage of some $WOTS^+$ public key with enough private key info to sign **some** digests
- Brute-force search for a valid Merkle/FORS tree whose root has signable digest
- Sign the tree root with the attacked $WOTS^+$ key
- To forge a signature, try message randomization strings until the hypertree address is a descendent address of the tree root

WOTS+ Signature

- Write digest as base-w (16) number
- Append a base-w checksum
  - (960 – <sum of digits>)
- Sign each digit d_i of digest plus checksum by:
  - Hash $sk_{i,0}$ (with prefix) d_i times
  - Put the result in the signature
- Note: The signature of 0xF is just pk_i

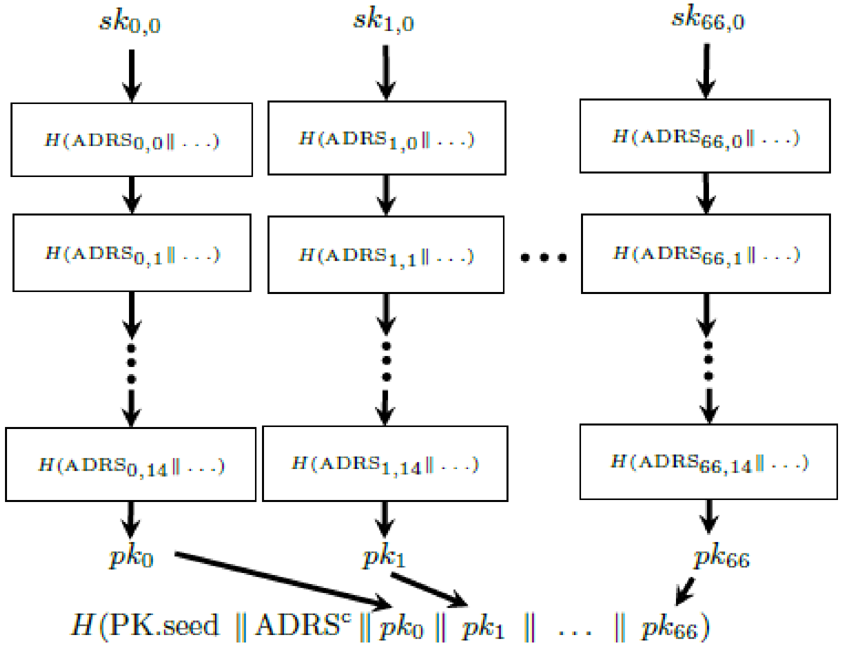


Fig. 2. A WOTS+ public key.

Finding a Merkle/FORS Root We Can Sign

- Aim to sign a digest like:

  xxxxxxxx xxxxxxxx xxxxxxxF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF

- Modify Antonov's multi-target preimage search to find a WOTS⁺ public key that can sign this
  - Treat the part that signs xxxx… as prefix – so we know $sk_{i,0}$ for this part
  - Use the last block of the prefix and the part that signs FFFF… for herding and multitarget preimage search
  - Target the SHA-256 state immediately before the first block that signs checksum
  - The part that signs the checksum will come from the target honest signature
- Can forge a signature on any Merkle/FORS root of the above form as long as checksum works out

Making Sure the Checksum Works Out

- For a digest like:

  xxxxxxxx xxxxxxxx xxxxxxxF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF

  - Checksum is $960 - 41 \cdot 15 - \sum x$
- We can increment, but not decrement, digits of honest checksum
  - Increment a digit by hashing (with prefix) sk_{i,d_i}
- Can choose targets with unusually small checksums
- Need $\sum x$ to be small enough with high enough probability

Optimization: Batched Multicollision Search

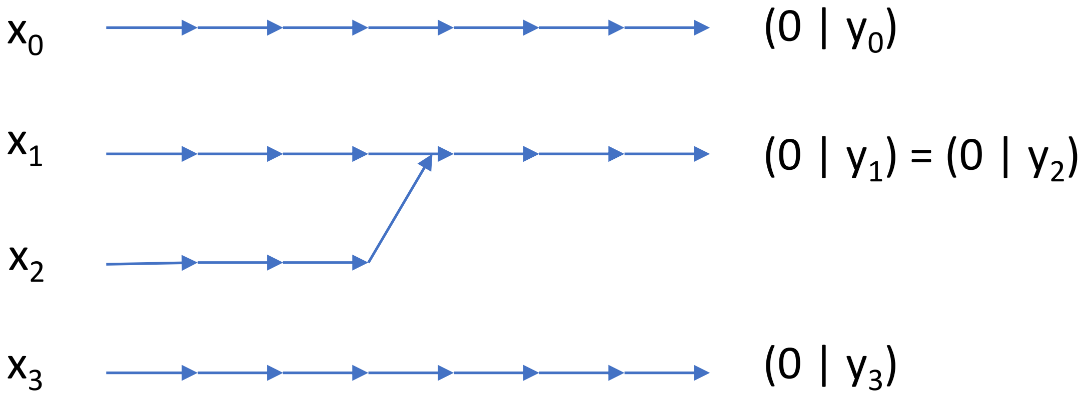
- Best parameterization of our attack involves finding lots of 4-way collisions with distinct prefixes
- It is cheaper to search for lots of collisions at once
  - Finding a single 4-way collision costs $\sim 2^{192}$
  - Finding t 4-way collisions costs $\sim 2^{192} t^{1/4}$
  - (Ignoring prefixes and memory access costs)
- To get good memory access costs, use parallel collision search techniques
- To avoid wasting time colliding already-used prefixes
  - Compute collisions in smaller batches of size αt
- More detail in paper

Memory Considerations for our Attack

- Sometimes memory intensive attacks or attacks that don't parallelize well are more expensive in practice than simple time-complexity suggests
- To show these considerations do not undermine our attack:
  - We analyze highly parallel implementations of all steps of our attack
  - in a computation model, where reading or writing a bit to a memory of size M costs $2^{-5}\sqrt{M}$ bit operations (probably somewhat pessimistic)
- This only changed the cost of our attack by about 2 bits of security

Memory/Parallelization-Friendly Collision Search [V-OW 1994]

- Each of M threads picks, and remembers, a random starting point x_i.
- Each thread iteratively hashes x_i until it reaches a "distinguished point" with $n/2 - \log_2(M)$ leading zeroes.
- The threads then collectively sort their outputs to find colliding distinguished points, y_i, y_j.
- Two threads iteratively recompute hashes of x_i, x_j to find the hash collision.
- Time = $\sim 2^{n/2}$/ M; Space = ~M.



Adapting V-OW to k-collisions [JL 2009]

- Similar to V-OW, but distinguished points need $\frac{k-1}{k}n - \log_2(M)$ leading zeroes
- M needs to be at least as large as the expected number of 2-collisions: $2^{\frac{k-2}{k}n}$

Important Memory Costs

- When using V-OW like techniques to find collisions with different prefixes
  - need to define a function on a 256-bit input that will pick each of t prefixes $\frac{1}{t}$ of the time
  - In order to deal with different prefixes either
    a. Recompute state including hash-chains every time
    b. Store precomputed values (about $512t$ bits)
  - Option b turns out to be cheaper, even with memory costs for reasonable values t
- Sorting distinguished points costs $O(M^{1.5})$
  - Dominant cost when looking for 4-way or bigger collisions

Attack Complexity

Table 1. Summary of Our Results on SPHINCS$^+$ Category Five Parameters

Parameter Set	Cost				Reference
	Herd	Link	Signable	Total	
SPHINCS$^+$-256f	$2^{214.8}$	$2^{216.4}$	$2^{215.7}$	$\approx 2^{217.4}$	Section 4.3
SPHINCS$^+$-256s	$2^{214.8}$	$2^{216.4}$	$2^{215.7}$	$\approx 2^{217.4}$	Section 4.3

SPHINCS$^+$ Tweak [Hülsing 2022]

- In response to Antonov's attack on DM-SPR the SPHINCS $^+$ team issued a tweak to the SPHINCS$^+$ specification
  - Replaced SHA-256 with SHA-512, for hashing multi-block inputs in Category 3 and 5 parameters
  - Still some use of SHA-256, but doesn't seem exploitable

Conclusion

- Our attack shows that some submitted parameter sets of SPHINCS$^+$ are not as strong as claimed
- The problem is not the security proof for the SPHINCS$^+$ construction, but how its tweakable hash functions are instantiated
- Lesson: need to be very careful trying to get more than 128 bits of security from SHA-256

- On the upside:
  - SPHINCS$^+$'s proposed tweak seems to address these issues
  - SHA-256 on fixed-length inputs pretty reliably gets 128 bits of security, so it's unlikely this sort of oversight leads to a practical break

Thank You!

References

[Antonov 2022] Antonov, S.: *Round 3 official comment: SPHINCS+* (2022), https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/FVItvyRea28/m/mGaRi5iZBwAJ

[Hülsing 2022] Hülsing, A.: *Round 3 official comment: SPHINCS+* (2022), https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/Ca4zQeyObOY

[Joux 2004] A. Joux, *Multicollisions in iterated hash functions. Application to cascaded constructions*, in ed. by M.K. Franklin. CRYPTO'04. Lecture Notes in Computer Science, vol. 3152 (Springer, 2004), pp. 306–316, https://www.iacr.org/archive/crypto2004/31520306/multicollisions.pdf

[Dean 1999] R.D. Dean, *Formal Aspects of Mobile Code Security*. Ph.D. thesis, Princeton University (January 1999)

[KS 2004] J. Kelsey, B. Schneier, *Second preimages on n-bit hash functions for much less than 2^n work*, in ed. by R. Cramer, Advances in Cryptology—EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22–26, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3494 (Springer, 2005), pp. 474–490, https://eprint.iacr.org/2004/304.pdf

[KK 2005] J. Kelsey, T. Kohno, *Herding hash functions and the nostradamus attack*, in ed. by S. Vaudenay. *EUROCRYPT*. Lecture Notes in Computer Science, vol. 4004 (Springer, 2006), pp. 183–200, https://eprint.iacr.org/2005/281

[V-OW 1994] van Oorschot, P.C., Wiener, M.J. Parallel Collision Search with Cryptanalytic Applications. *J. Cryptology* **12**, 1–28 (1999) https://people.scs.carleton.ca/~paulv/papers/JoC97.pdf

[JL 2009] Joux, A., Lucks, S. (2009). Improved Generic Algorithms for 3-Collisions. In: Matsui, M. (eds) Advances in Cryptology – ASIACRYPT 2009. https://eprint.iacr.org/2009/305