Optimization for SPHINCS$^+$ using Intel® Secure Hash Algorithm Extensions

Thomas Hanson[1] *, Qian Wang[2], Santosh Ghosh[2], Fernando Virdia[2], Anne Reinders[2], Manoj R. Sastry[2],

[1] University of Maryland, College Park
[2] Security and Privacy Research, Intel Labs, Intel Coorperation, Hillsboro, Oregon
thanson@umd.edu, qian4.wang@intel.com, santosh.ghosh@intel.com, fernando.virdia@intel.com, anne.reinders@intel.com, manoj.r.sastry@intel.com

Abstract. SPHINCS$^+$ was selected as a candidate digital signature scheme for standardization by the NIST Post-Quantum Cryptography Standardization Process. It offers security capabilities relying only on the security of cryptographic hash functions. However, it is less efficient than the lattice-based schemes. In this paper, we present an optimized software library for the SPHINCS$^+$ signature scheme, which combines the Intel® Secure Hash Algorithm Extensions (SHA-NI) and AVX2 vector instructions. We obtain significant speed-up of SPHINCS$^+$-128f-simple on both non-optimized (70%) and AVX2 reference implementations (8% -23%) offering 128-bit security.

Keywords: post-quantum cryptography, digital signatures, SPHINCS$^+$, SHA-NI, AVX2

1 Introduction

The security of the current most prevalent cryptographic signature schemes relies on hardness assumptions which are broken by quantum computers [1]. As a result, there is an ongoing effort to design and improve signature schemes which offer secure capabilities against both classical and quantum attacks. Recently, the National Institute of Standards and Technology (NIST) have announced their list of candidates digital signature schemes for standardization. Among the three selected candidates, CRYSTALS-Dilithium [2] and FALCON [3] are both based on hard problems on lattices, with SPHINCS$^+$ [4, 5] being the only scheme based on a different assumption. Indeed, SPHINCS$^+$ relies on the existence of a class of collision-resistant cryptographic hash functions. The security of cryptographic hash functions is generally better understood than the hardness of lattice assumptions, particularly those with algebraic structure such as Ring Learning With Errors (Ring-LWE) [6, 7]. However, despite offering an efficiency improvement over previous hash-based signature schemes, SPHINCS$^+$ is not as efficient as the lattice-based candidates. Indeed, according to the NIST Round 3 report [8] SPHINCS$^+$ is roughly a factor of 100 slower than CRYSTALS-Dilithium and FALCON. Thus, it is timely and crucial to find approaches to improve its efficiency.

SPHINCS$^+$ can be instantiated using different hash functions. For example, there are reference implementations for HARAKA, SHAKE and SHA2. Calls to the chosen hash function dominate the overall computation time during key generation, signing and signature verification. While on the one hand its design provides flexibility at the time of instantiating the scheme, on the other hand Intel has developed the SHA-NI Instruction Set Architecture (ISA) extension, which increases SHA-256 performance significantly. It is then natural to consider SHA-NI-accelerated implementations of SPHINCS$^+$ when instantiated using SHA-256.

From now on when writing SPHINCS$^+$ we are referring to its SPHINCS$^+$-128f-simple instantiation using the SHA-256 family of hash functions claiming 128-bit security.

Contributions. The contributions of this paper are twofold: First, we optimize the single-buffer implementation of SPHINCS$^+$ by utilizing the specific instructions in the SHA-NI instruction set, achieving a 70%+

*Thomas Hanson's research contributions were made during his internship at Intel Labs.

performance increase. Second, we further optimize the AVX2 implementation by the SPHINCS$^+$ team [9] by using SHA-NI where computing serial hashing operations, resulting in a 23% speed-up in verification time and a 8% speed-up in signing.

Related Work. Zhang *et al.* [10] have improved upon SPHINCS$^+$ by using different One-Time Signatures (OTS) or Few-Times Signatures (FTS) schemes, similarly to what was changed between SPHINCS [11] and SPHINCS$^+$. Sun *et al.* [12] demonstrate improvements by parallelizing parts of the SPHINCS$^+$ algorithms, however they use different parameters and hash functions to attain their results. one of the intriguing work is that of Yehia *et al.* [13], which uses an extra hash computation on message in order to pick the specific FTS scheme to use. This allows using smaller parameters for the FTS scheme, while maintaining the same security level.

2 Preliminaries

2.1 SPHINCS$^+$

SPHINCS$^+$ was selected for standardized digital signature schemes in order to have a stateless post-quantum signature scheme whose security does not rely on the hardness of lattice problems. NIST also asked for public feedback on a version of SPHINCS$^+$ with a lower number of maximum signatures [8].

SPHINCS$^+$ is a hash-based signature scheme which uses a structure based on several layers of Merkle trees; in total d layers of trees are calculated. On all but the lowest layer, the leaves of the tree are used to sign the roots of the next layer of trees. The leaves on the lowest layer are used to sign the message itself. Due to its design, the entire multi-layer tree structure does not need to be calculated when signing. Instead, for each signature, a single path down to the lowest layer is chosen pseudo-randomly (based on the randomness of the secret key and a hash of the message), and only the trees along the path to that leaf need to be calculated. All of the keys for the component signature schemes are chosen deterministically with a carefully constructed pseudo-random function, so they do not have to be retained and can be recomputed as needed.

SPHINCS$^+$ uses the Winternitz type one-time signature scheme, WOTS$^+$, to sign the intermediate Merkle trees' roots. WOTS$^+$ uses a chaining function to sign messages: l values are chosen based on the WOTS$^+$ secret key, and each is hashed repeatedly w times. The end of the hash chains produced are hashed together to create the public key. The message being signed is converted into l integers between 0 and $w-1$, which correspond to positions in the hash chains. These positions in the hash chains form the WOTS$^+$ signature. The verifier can use the message to complete each hash chain, by performing $w - m_i$ more hashes, recreating the public key. In SPHINCS$^+$, the public key is compressed using a call to a tweakable hash function, rather than using l-trees as in previous definitions of SPHINCS.

Importantly, the verification procedure recreates the WOTS$^+$ public key, which is then compared to the actual public key. This property allows the WOTS$^+$ private key to be generated during the SPHINCS$^+$ signing procedure, and used to sign an intermediate tree, since the verifier can then calculate the public key themselves from the signature. A incorrect WOTS$^+$ signature will cause a failure during verification . The structure of the signature for each tree is very similar to that of the XMSS signature scheme [14, 15, 16].

At the leaves of the lowest layer's tree, a few-time signature scheme, FORS, is used. This is the important difference from XMSS, which allows SPHINCS$^+$ to be stateless, where XMSS needs to keep state in order to track which leaves have been used to sign message so that it doesn't use a WOTS$^+$ key twice. Using a few-time signature scheme allows there to be a few "leaf collisions", where the random path to the leaf is the same for different signed messages, without catastrophically losing security.

3 Improving Performance using Intel's Instruction Set Architecture Extensions

3.1 Evaluating SHA-256

On Intel's computing platforms, there are three specific ISA extensions that could be used to accelerate the SHA-256 algorithm: SHA-NI, AVX2 and AVX512.

SHA-NI is part of the Streaming SIMD Extensions (SSE) instruction set [17], and introduces instructions specifically designed to greatly decrease latency in single-buffer evaluation of SHA-256. Depending on the implementation, SHA-NI can result in a 50%-75% performance increase. A dual-buffer implementation of SHA-NI is also available.

The Advanced Vector Extensions AVX2 and AVX512 have been widely used to accelerate cryptographic algorithms. Although they do not include specific instructions for SHA-256, they provide large registers which can allow multi-buffered SHA-256 calls. AVX2 can compute 8 multi-buffered SHA-256 calls in parallel and AVX512 can compute 16 multi-buffered SHA-256 calls in parallel. However, neither can be used to increase the single-buffer performance of SHA-256. In Section 5.1 we will discuss how we can take advantage of SHA-NI alongside AVX2 to overall improve performance of SPHINCS$^+$.

In Table 1 we report SHA-256 hash call performance in cycles when using the three extensions described above, as well as using no extensions, when run on an Intel® Core i7™-1185G7 CPU. We note that because the $\times 1$ and $\times 16$ tests use implementations from different libraries, they are not directly comparable. However, the numbers allow us to measure relative speeds of the instruction sets within each test. Also note that the $\times 16$ experiments use dual-buffer SHA-NI.

Table 1: Number of cycles spent for $\times 1$ hash calls and $\times 16$ hash calls of SHA-256.

Algorithm	1 Hash	16 Hashes	Cycle increase factor
Base	901	4759	5.28
SHA-NI	(single-buffer) 270	(dual-buffer) 2081	7.71
AVX2	—	2192	—
AVX512	—	878	—

3.2 Accelerating SPHINCS$^+$

Since SPHINCS$^+$ is a hash-based signature scheme, we start by exploring the number of hash calls in SPHINCS$^+$,[1] which we report in Table 2. The signing algorithm in SPHINCS$^+$ is relatively slow, using one and two orders of magnitudes more hash calls than signature verification and key generation. As a result, our first goal is to reduce the signing time, especially for scenarios where many messages must be signed. We would also like to reduce the verification time as much as possible, as sometimes verification is conducted on resource limited devices.

Table 2: Number of blocks of SHA-256 computed in SPHINCS$^+$ -128f-simple

Function	SHA-256
Key Generation	4847
Sign	112937
Verify	13117

4 Combining SHA-NI and AVX2

The current implementation of SHA-256 in the implementations of SPHINCS$^+$ provided by the SPHINCS$^+$ Team [9] uses AVX2. However, it does not take advantage of SHA-NI, which takes advantage of instructions available on recent Intel CPUs to speed up the SHA-256 algorithm. We can incorporate SHA-NI instructions into the AVX2 implementation SPHINCS$^+$ from [9] to decrease the scheme's latency. The instructions necessary for SHA-NI were introduced in 2013 [17].

We developed an optimized software library for SPHINCS$^+$ which decreases the latency of the hash calls using Intel ISA extensions, and in turn decreases the SPHINCS$^+$ latency overall. We use a combination of SHA-NI (from SSE) and AVX2 to provide a performance increase over the reference implementation that just

[1] We remind the reader that when writing SPHINCS$^+$ we are referring to its instantiation using the SHA2 family of hash functions claiming 128-bit security.

uses AVX2. The library invokes calls to an AVX2 implementation of SHA-256 for the hash calls which can be parallelized in SPHINCS$^+$, and calls to a SHA-NI implementation of SHA-256 for the serial hash calls that cannot be parallelized. By optimizing each of these steps, we reduce the overall latency for SPHINCS$^+$. To speed-up the single-buffer hash calls, we used an implementation of SHA-256 using SHA-NI from the Intel® Integrated Performance Primitives (IPP) Cryptography library [18].

5 Experiments

5.1 Experimental Setup

We ran our experiments on an Intel® Core i7™-1185G7 CPU core, which includes the SSE, AVX2 and AVX512 instruction sets, and it runs at a maximum frequency of 3.0 GHz. The parameter set used is SPHINCS$^+$-128f-simple, which is the fastest implementation of SPHINCS$^+$ for target NIST security level 1. The reference of implementation and AVX2 implementation is constructed from the sphincsplus-master code base [9]. For the single buffer SHA-NI, we are importing the code used in IPPCP library [18] and the dual-buffer SHA-NI from the IPSEC-MB library [19].

5.2 Performance Results

Figure 1 shows a comparison between the "reference 1" implementation (no ISA acceleration) and single-buffer SHA-NI. The SHA-NI implementation gives a $60-70\%$ speedup for SPHINCS$^+$ algorithms

Figure 1: Performance of SHA-NI compared to the reference SPHINCS$^+$ implementation without ISA

Figure 2 shows a comparison between the "reference 2" implementation (using AVX2) and our combined SHA-NI and AVX2 implementation. The combined implementation does not offer a performance increase for Key Generation, but offers an 8.7% speedup in Signing and a 23% speedup in Verification.

From Figure 2, we find out there is a much larger performance speed-up in verification compared to signing. The cause for the discrepancy in speed-up is because the portion of parallelizable hash calls are different. From Table 3, we have shown the comparison between the number of SHA-256 calls that are multi-buffered (and can use AVX2), and those that are single-buffered (and can be accelerated with SHA-NI). This result shows that almost all of the calls to SHA-256 in the Key Generation and Signing algorithms are parallelizable, whereas a much larger portion of the SHA-256 calls in Verification are not parallelizable which we can exploit with SHA-NI.

Much of this difference is because of the adjacency paths that are part of the signature. During signing, multiple WOTS$^+$ or FORS keys must be generated, and they must be hashed together to create each tree. The signer provides the intermediate hash values along the path to the leaf with the signature to the verifier, who can then compute them. The described procedure for the signer benefits greatly from parallelism, but the verification procedure cannot be parallelized.

Figure 2: Performance of combined implementation compared to the reference AVX2 SPHINCS$^+$ implementation

Table 3: Number of cycles spent using single-buffer (×1) and multi-buffer (×8) SHA-256 in SPHINCS$^+$ -128f-simple

Function	×1	percentage	×8	percentage
Key Generation	3642	0.5%	679508	99.5%
Sign	57376	0.2%	33718342	99.8%
Verify	1134404	25.5%	3321470	74.5%

6 Conclusion

The main contribution of this paper is an optimized SPHINCS$^+$ library which combines AVX2 and SHA-NI, using AVX2 where hash calls can be parallelized to take advantage of its low multi-buffer latency, and SHA-NI where for non-parallelizable hash calls to take advantage of its low single-buffer latency. We conducted our experiments on an Intel® Core i7™-1185G7 CPU core, which includes the SSE, AVX2 and AVX512 instruction sets. The combined implementation offers an 8.7% speedup in Signing and a 23% speedup in Verification. In the future, we plan to extend the combined library of SHA-NI and AVX512 to other variants of SPHINCS$^+$ instantiation using the SHA2 family of hash functions.

References

[1] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *J. Comput.*, 1997.

[2] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):238–268, Feb. 2018.

[3] Pierr-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, Zhenfei Zhang, and et al. Falcon: Fast-fourier lattice-based compact signatures over ntru, Jan 2020.

[4] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS$^+$ signature framework. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 2129–2146, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M Lauridsen, Florian Mendel,

Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS+. Technical report, National Institute of Standards and Technology, 2022. Available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[6] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 617–635. Springer, 2009.

[7] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):1–35, 2013.

[8] NIST IR 8413. Status report on the third round of the nist post-quantum cryptography standardization process, July 2022.

[9] SPHINCS$^+$ Team. SPHINCS$^+$, 2022. Available at https://github.com/sphincs/sphincsplus, commit 79ea9b7.

[10] Kaiyi Zhang, Hongrui Cui, and Yu Yu. Sphincs-α: A compact stateless hash-based signature scheme. Cryptology ePrint Archive, Paper 2022/059, 2022. https://eprint.iacr.org/2022/059.

[11] Daniel J Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. Sphincs: practical stateless hash-based signatures. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 368–397. Springer, 2015.

[12] Shuzhou Sun, Rui Zhang, and Hui Ma. Efficient parallelism of post-quantum signature scheme sphincs. *IEEE Transactions on Parallel and Distributed Systems*, 31(11):2542–2555, 2020.

[13] Mahmoud Yehia, Riham AlTawy, and T. Aaron Gulliver. Verifiable obtained random subsets for improving sphincs+. In Joonsang Baek and Sushmita Ruj, editors, *Information Security and Privacy*, pages 694–714, Cham, 2021. Springer International Publishing.

[14] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. Xmss-a practical forward secure signature scheme based on minimal security assumptions. In *International Workshop on Post-Quantum Cryptography*, pages 117–129. Springer, 2011.

[15] A. Huelsing, D. Butin, S. Gazdag, J. Rijneveld, and A. Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, RFC Editor, May 2018. Available at https://www.rfc-editor.org/rfc/rfc8391.txt.

[16] David A. Cooper, Daniel C. Apon, Quynh H. Dang, Michael S. Davidson, Morris J. Dworkin, and Carl A. Miller. Recommendation for stateful hash-based signature schemes. Technical report, October 2020.

[17] Sean Gulley, Vinodh Gopal, Kirk Yap, Wajdi Feghali, Jim Guilford, and Gil Wolrich. Intel (r) SHA extensions. Technical report, Intel Corporation, 2013. Available at https://www.intel.com/content/dam/develop/external/us/en/documents/intel-sha-extensions-white-paper.pdf.

[18] Intel. Intel integrated performance primitives cryptography, 2022. Available at https://github.com/intel/ipp-crypto.

[19] Intel. Intel crypto multi-buffer library, 2022. Available at https://github.com/intel/ipp-crypto/blob/develop/sources/ippcp/crypto_mb/Readme.md.