Post-Quantum Authentication with Lightweight Cryptographic Primitives

Henrique Faria and José Manuel Valença * [1]

[1]HasLAB, Departamento de Informática, Universidade do Minho

September 27, 2021

Abstract

We propose to adapt "low-algebra" digital signature schemes SPHINCS+ and PICNIC, present in the NIST-PQC contest, to the limitations of resource-bounded low-end devices. For this, we replaced the cryptographic primitives (hash functions and symmetric ciphers) of these schemes with lightweight alternatives presented in the NIST-LWC contest. With these specifically conceived primitives, we improve the performance of the signature schemes and still preserve the NIST's security levels.
Regarding SPHINCS+, besides replacing the hash function, we also take into consideration relaxing some parameters and introduce a new notion: security as life expectancy. Furthermore, we also introduce an attack to the SPHINCS+ scheme that takes advantage of the usage of FORS on this scheme and the way its leaves are calculated. Also, we give some solutions on how to avoid this attack. Additionally, a modification of PICNIC is introduced as PICNIC+WOTS where PICNIC is used to generate the secret keys for several WOTS+ signatures significantly reducing the size and signature time of each signature.

keywords: SPHINCS+; PICNIC; ASCON; SKINNY; WOTS+;

1 Introduction

Cryptographic authentication must be present in any information system. More so in safety-critical or mission-critical systems which rely upon a large loosely-structured network of semi-autonomous devices: e.g. sensors, controllers, communication units.

There are some major obstacles to such goal: usually, these devices are computationally limited only allowing the execution of the simplest cryptographic techniques. Moreover, recent progress in the development of quantum computers foresees the day when existing "number theoretic" public key cryptosystems no longer provide the security level required by such systems.

Thus critical systems must use authentication techniques that are immune to quantum attacks and, simultaneously, can run on resource-limited low-end devices.

*a82200@alunos.uminho.pt, jmvalenca@di.uminho.pt

The NIST-PQC[1] standardization process proposed several digital signature schemes (DSS's) capable of providing, in the context of quantum attacks, the required security levels. Of these, some classify as "low-algebra" and derive high levels of security, exclusively from the properties of simple cryptographic primitives, symmetric ciphers, and hash functions used in very long computation chains.

In another NIST contest (NIST-LWC[2]) "lightweight" cryptographic primitives have been developed. These are classical cryptographic algorithms, like hash functions and block ciphers capable of being used with a good security level in the least powerful devices.

These algorithms may not be suited to schemes which require high levels of algebraic complexity but there exists a good possibility of excelling in combination with "low-algebra" schemes. In this paper we will provide evidences on the performance of post-quantum "low-algebra" schemes (SPHINCS+ and PICNIC) merged with lightweight implementations of hash functions and block ciphers featured in the NIST-LWC initiative (ASCON and SKINNY).

Note that both ASCON and SKINNY appear in submissions selected to the 3rd round of the NIST-LWC standardization process: ASCON is part of the submission ASCON, whereas SKINNY is the underlying algorithm in the submission ROMULUS. According to the report[13], both algorithms have received comprehensive third-party security analysis. More than raw throughput, good security is the main requirement in using symmetric primitives in authentication schemes.

In this work we show that:

1. Using lightweight primitives (ASCON and SKINNY) is a viable solution to enhance "low-algebra" schemes' signature speed.
2. If we acknowledge SPHINCS+ security as the life expectancy of a low-end device we can relax its security parameters accordingly.
3. Once the SPHINCS+ scheme's hypertree height is diminished, there is a viable attack that can be avoided with a minor tweak.
4. PICNIC can be greatly enhanced both in signature size and speed by merging it with the WOTS+ scheme.

2 Results

Here we present how we merged PICNIC and SPHINCS+ with the lightweight ciphers and the results obtained.

2.1 SPHINCS+

The SPHINCS+ scheme is a hash based signature scheme that allows several trade-offs between signature size and signing speed. A full formal specification of SPHINCS+ is given in [3].

Regarding the proposed modifications, the SPHINCS+ scheme allows for the most simple implementation of both leightweight primitives as it only requires the use of a secure hash function independently of how it works underneath.

The table 1 show the average number of cycles it took to produce 100 signatures for the SPHINCS+ scheme using SHA3, ASCON, and SKINNY as the underlying hash function respectively.

Version	Average Cycles per Run
SPHINCS+ with SHA3	133,098,259
SPHINCS+ with ASCON-Hash	112,104,936
SPHINCS+ with SKINNY-Hash	6,435,300,567

Table 1: SPHINCS+ versions performance.

As can be seen in the above table, using ASCON allows for an average of 16% decrease on the number of cycles taken to produce 100 signatures. On the other hand, using SKINNY, it took 48 times more cycles to produce the same amount of signatures. Considering the above results, only the ASCON-Hash is viable, and we can further improve its speed if we look at its underlying behavior.

The ASCON cipher output is obtained by drawing 8 bytes at a time from its internal state, meaning that if we require the hash function's output to have 8 bytes, we can improve the scheme performance-wise. This solution should only be considered regarding small non-critical IoT devices that do not need as much security as more important devices do. The average cycles it takes to produce 100 signatures using ASCON and a hash output of 8 bytes represent 61% of the cycles required to produce the same amount of signatures using SHA3 and a hash output of 8 bytes.

Version	Hash Output length (bits)	Average Cycles per Run
SPHINCS+ with SHA3	128	133,098,259
SPHINCS+ with ASCON-Hash	128	112,104,936
SPHINCS+ with SHA3	64	67,247,767
SPHINCS+ with ASCON-Hash	64	40,937,728

Table 2: SPHINCS+ performance.

From [3] we know that the number of layers in the hypertree and the Winternitz parameter do not affect the scheme's security. Additionally, lowering the height of the hypertree or the number of FORS trees also lowers the security of the scheme. However, we are aiming at low-end devices which usually do not need to sign a lot of messages over short periods of time. That means we can relax the hypertree height parameter to produce smaller signatures. As such, several parameters' combinations were tested and an average of the signature size and number of cycles to run each signature are presented in table 3. Each version was used to produce 100 signatures and they all use the best underlying hash function (ASCON-Hash).

Version index	h	d	k	w	Signature size (bytes)	Average Cycles per Run
Control version	66	22	33	16	17088	109,010,685
Version 1	66	22	33	256	11041	819,792,980
Version 2	44	22	33	16	16736	61,261,492
Version 3	33	11	33	16	10400	59,636,855
Version 4	66	11	33	256	7936	3,085,134,761
Version 5	55	11	33	256	7760	1,562,615,851
Version 6	44	11	33	256	7584	791,835,778
Version 7	33	11	33	256	7408	413,112,696

Table 3: SPHINCS+ with ASCON-Hash 128 bits versions' performance.

A quick analysis of the previous table shows that the best signing speed is obtained by having fewer layers and a low Winternitz parameter while the smallest signature size is obtained by having a lower hypertree, fewer layers, and a high Winternitz parameter. To attain NIST's first security level the hypertree must have a minimum height of 63 corresponding to SPHINCS+-128s[3]. However, for low-end devices, we introduce a new idea of security as life expectancy.

Security as life expectancy: The security of the SPHINCS+ signatures relies on each of the WOTS+ key-pairs single use. Therefore, for this scheme, life expectancy defines as the moment all WOTS+ key-pairs have been used once. To compute the life expectancy, one needs to know how many different WOTS+ key-pairs exist in a SPHINCS+ scheme, which is given by the number of XMSS trees in the last layer of the hypertree times the number of WOTS+ key-pairs in each XMSS: $2^{h-h'} * 2^{h'}$, where h is the total height of the hypertree and h' the height of each XMSS tree.

As an example of the life expectancy of a low-end device, assume we are using a version of the SPHINCS+ scheme with the following parameters: h = 33, d = 11. For this version, the number of WOTS+ key-pairs is 2^{33}. The best versions of this scheme presented on table 3 are version 3 regarding speed needing 0.02 seconds per signature and version 7 regarding memory requiring 0.1132 seconds per signature. Time-wise, if a device produced signatures non-stop, it would take approximately 5.45 and 30.83 years to exhaust all the possible WOTS+ key pairs, respectively. After this time, the device needs either to be replaced or to generate a new secret key. Therefore, we believe that relaxing the hypertree height is an acceptable modification.

2.2 PICNIC

PICNIC is a signature scheme based in a non-interactive zero knowledge proof of knowledge of a circuit's input. The proof of knowledge is an instance of the "multi-party computation (MPC) in-the head" approach of Ishtai[4]. The circuit is defined by a secure block cipher $\{F_x\}$, a pseudo-random generated plaintext u and a computed ciphertext y. The pair (u, y) is part of the signature scheme's public-key; the signer proves knowledge of a secret key x such that $y = F_x(u)$.

Moreover the only cryptographic building blocks of the scheme are symmetric primitives (block ciphers and hash functions) and its post-quantum security does not rely in any number theoretic assumptions. Thus the scheme is truly "low-algebra".

A full formal specification of PICNIC is given in [5, 8]. In all instances of PICNIC thus defined, the block cipher is instantiated with LowMC, a simple cipher designed with very few non-linear components. However, this cipher uses, for good diffusion properties, matrix multiplications of large boolean matrices, and a very large chunk of the signature verification algorithm is spent with these multiplications. Clearly, there is some gain in replacing LowMC with some alternative cipher more in the line of the NIST-LWC candidates.

In [6], one can find a formal specification of the Skinny cipher and in [7] ASCON-Hash specification. Both are present in the NIST-LWC contest, the former as an instance of the "substitution permutation network" (SPN) and the latter as an example of the "sponge" construction.

As in LowMC, both the non-linear s-boxes in SKINNY and ASCON have an algebraic degree of 2. With that in mind, these ciphers were modified to resemble the LowMC cipher S-box. Another tweak used on both ciphers was to reduce the number of rounds in each one, as keeping them would double the size of the scheme's signature.

SKINNY: The SKINNY cipher has a non-linear s-box based on the 8-bit state transformation:

$$(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0)$$

$$\downarrow$$

$$(x_7, x_6, x_5, x_4 \oplus \overline{(x_7 \vee x_6)}, x_3, x_2, x_1, x_0 \oplus \overline{(x_3 \vee x_2)})$$

Figure 1: SKINNY 8-bit transformation.

SKINNY's 8-bit state transformation was modified to work on each 8-bit state as if it was two 4-bit states and, for each, the first three bits remained unchanged while the fourth-bit updates as the result of applying an AND gate to the first and second bits. The following image shows the new transformation.

$$(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0)$$

$$\downarrow$$

$$(x_7, x_6, x_5, (x_7 \wedge x_6), x_3, x_2, x_1, (x_3 \wedge x_2))$$

Figure 2: Modified SKINNY 8-bit transformation.

As previously mentioned, another problem identified was the size of each signature on the PICNIC-SKINNY implementation. This size reflects the number of rounds the SKINNY cipher uses on its state encryption, resulting in view transcripts with double the original version's size. To attain the same signature size, the number of rounds in the SKINNY cipher was cut from 56 to 24. A thorough analysis of the security of SKINNY's modified S-box and its rounds can be found in A.

ASCON: The ASCON cipher, just like SKINNY, has a non-linear s-box which needs to be addressed. This s-box is applied to all the five words in the cipher's internal state. Each word is represented by the w_i's in the scheme below.

$$(w_0, w_1, w_2, w_3, w_4)$$

$$\downarrow$$

$$(w_0 \oplus (w_1 \wedge w_2), w_1 \oplus (w_2 \wedge w_3), w_2 \oplus (w_3 \wedge w_4), w_3 \oplus (w_4 \wedge w_0), w_4 \oplus (w_0 \wedge w_1))$$

Figure 3: ASCON 5-words transformation.

The AND gate on each word was replaced by a series of S-boxes that applied, to each word's bytes, the same operations presented for the SKINNY modified S-box.

Once again, the transcript generated was too big when compared to the original cipher. The solution found was to reduce the rounds on the ASCON cipher. So, on the initial state permutations, rounds were reduced from 12 to 4, and on the permutations occurring after every 8 bytes of plaintext absorbed, rounds were reduced from 6 to 3.

The resulting PICNIC schemes were run 100 times and the values presented are the average signature size and average signature speed.

Version	Number of Runs	Signature size (bytes)	Real Time
Original PICNIC-FS	100	[32500,33500]	9.56
PICNIC with SKINNY	100	[37000,38000]	2.51
PICNIC with ASCON	100	[38000,38600]	2.37

* The times presented were measured in seconds.

Table 4: PICNIC versions performance.

As presented in the table above, both versions are significantly better than the original taking less than a third of the time to sign a message. Additionally, it's shown how both Sponge models and SPN(substitution Permutation Networks) designed to be used by low-end devices are good alternatives when it comes to reducing performance-wise costs.

Nevertheless, PICNIC still uses lots of memory. Not only that, both modifications slightly increased PICNIC's signature due to the transcript size in each round. To solve this issue we implemented the WOTS+ scheme over PICNIC presented in section 4.

3 SPHINCS+ Attack

The security margin for SPHINCS+ is the ratio, in relation to the height of the original hypertree, of minimum height for which we can not forge a signature in an acceptable time.

To estimate the security margin we run our attack against several hypertrees of differente heights. Much like previous attacks, this attack was executed on simple domestic personal computer with low processing capabilities. Furthermore the computer was being used for fay-to-day tasks and other programs. Even so it was able to produce a valid signature for a hypertree of height 28 in approximately 120 hours.

That establishes a security margin which is no better than 58%; meaning that is unsafe to use this scheme with a height inferior to 28, since a simple domestic computer can forge signatures in acceptable time.

In SPHINCS+'s scheme, the leaves indexes used in the FORS signature, as well as the index of the XMSS tree and the WOTS+ key pair are the result of hashing the message we intend to sign with a random value R.

As the FORS secret keys are computed using the indexes of the XMSS and WOTS+ key pair, for a given XMSS and WOTS+ key pair the FORS public key

is always the same independently of the message being signed. Furthermore, as each WOTS+ key pair has the same AUTH path in a XMSS, and each XMSS, independently of the WOTS+ key pair used in it, is signed with the same WOTS+ key pair in the layer above, the hypertree signature for a choosen WOTS+ key pair is also exactly the same.

The only difference between signatures using the same WOTS+ key pair is the combination of the FORS trees' signatures. However, such combinations can overlap over one or multiple trees, and that's the peculiarity that an EUF-CMA attack takes advantage of.

In this attack, an adversary chooses a message he wants to forge and a random value R and uses them as input to the known hash function. From the hash's output, he retrieves the XMSS and WOTS+ key pair indexes to target, and, from the remaining digest, the indexes of each FORS trees he needs for the forgery.

Afterwards, he generates more messages which are sent to the signing oracle. He then selects those which are signed with the same XMSS and WOTS+ key pair indexes. For each of the selected signatures, he then checks if any index of a message digest overlaps the required FORS indexes, and if so, he can withdraw from the oracle's response the FORS tree signature. Note that, from the signature output by the oracle, he only needs to retrieve the part corresponding to the overlapping tree/trees. After repeating this process enough times, he will have the forgery complete.

Assume we want to attack a SPHINCS+ scheme with the following FORS trees, each with four leaves.

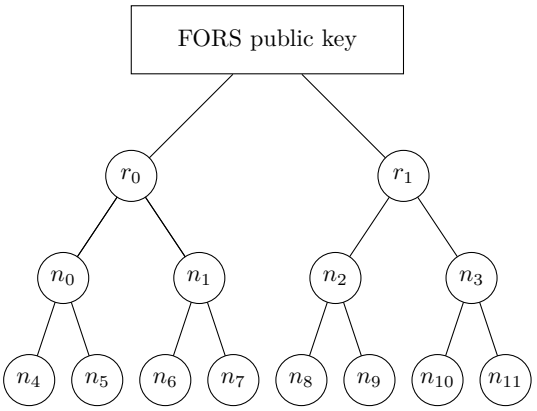

Figure 4: FORS tree example

Assume that we have generated three messages that use the same WOTS+ key pair, meaning they share the same hypertree signature H over the same FORS public key. The first message m0 is the one we intend to forge, and the second and third were sent to the oracle to be signed.

The table below shows, for each message, the computed leaves of each FORS's tree and the nodes contained in each FORS tree's signature. The signatures of m0 will be represented with the symbol ? as the table only shows the results of querying the oracle with the messages m1 and m2.

Message	Hash output	Signature first tree	Signature second tree
m_0	$[0,0]$	?	?
m_1	$[0,3]$	$(n_4,[n_5,n_1])$	$(n_{11},[n_{10},n_2])$
m_2	$[2,0]$	$(n_6,[n_7,n_0])$	$(n_8,[n_9,n_3])$

* For each signature the first node is the secret key and the nodes in brackets are the authentication path.

Table 5: Message being forged (m_0) and messages signed by the oracle (m_1 and m_2).

As already mentioned, the messages above share the same FORS trees, meaning that if two of them share the same leaf of a specific tree they share the same signature for that same FORS tree. That allows us to obtain m0's signature by combining m1's first tree signature and m2's second tree signature updating the table as follows. The rest of the SPHINCS+ signature (Hypertree signature) is the same for the three messages, so we only need to copy either m1's or m2's hypertree signature to have a successful forgery.

Message	Hash output	Signature first tree	Signature second tree
m_0	$[0,0]$	$(n_4,[n_5,n_1])$	$(n_8,[n_9,n_3])$
m_1	$[0,3]$	$(n_4,[n_5,n_1])$	$(n_{11},[n_{10},n_2])$
m_2	$[2,0]$	$(n_6,[n_7,n_0])$	$(n_8,[n_9,n_3])$

* For each signature the first node is the secret key and the nodes in brackets are the authentication path.

Table 6: Successfull forgery of message m_0.

A first version of this attack was written in C and used on SPHINCS+ benchmark implementation. The results are summed in the table below.

Number of Runs	N (Security parameter)	W	h	d	k	a	Average Cycles per run
100	16	16	12	3	33	6	7,281,725,614

* The time was measured in seconds.

Table 7: Successfull attack on SPHINCS+.

During this attack, while attempting to forge a signature, several collisions for other leafs were found. This lead us to believe that, if we try to forge several signatures at the same time the overhead time-wise would be negletible as it would be the time spent checking if the collision found was for one of the forgeries we are attempting.

As such, this attack was also tested for multiple forgeries at the same time. The following table compares the time it took to forge a signature on the first version of the algorithm with the time took to forge h' forgeries in the second for hypertrees of height 12 and 18.

Number of forgeries	Number of Runs	N (Security parameter)	W	h	d	k	a	Average Cycles per run
1	100	16	16	12	3	33	6	7,281,725,614
16	100	16	16	12	3	33	6	21,060,881,184
1	100	16	16	18	3	33	6	1,075,532,284,162
64	100	16	16	18	3	33	6	1,584,206,932,122

Table 8: Single forgery vs Multiple forgeries.

As can be seen in the previous table, the time difference in both attacks is neglectable given the number of messages forged. The time overhead corresponds to the time it takes to verify if the signature output by the oracle corresponds to one of the WOTS+ key-pairs we are targeting and to retrieve the required FORS tree signatures. Furthermore, this time overhead grows linearly with the number of forgeries we are forging for a hypertree with a given height.

We now introduce the notion of security margin for SPHINCS+ as the minimum height of the hypertree for which we can not forge a signature in an acceptable time. To define a base security margin, we run an attack against several hypertrees of different heights. Much like the previous attacks, this attack was run on a personal computer with low processing capabilities and shared resorces. As stated before we were able to produce a valid signature, for a hypertree of height 28, in 120 hours of processing time. This suggests security margin no larger then 50%.

3.1 Possible solutions

The previous attack takes advantage of SPHINCS+ by finding XMSS and WOTS+ key pair indexes collisions over several messages. A simple solution would be to increase the height of the hypertree. However, that would hardly be the best option as it would increase SPHINCS+'s signature size and the time it takes to process a signature making the scheme less apealing.

The best possible solution we came up with to prevent such an attack, is to make the FORS public key independent from the hypertree indexes. Our solution is to first, guarantee that all FORS's tree signatures are different and only then compute the indexes either from the FORS public key value or from a FORS tree's signature.

One way to do this is to take advantage of the security notion of EUF-CMA SPHINCS+ is claimed to have. According to this security notion, the scheme is secure as long as the same message isn't used more than once. So, we can compute each FORS tree secret values by using the message to be signed and the secret seed. That means, as all messages are different, all the FORS tree's signatures and the FORS public keys will also be different. We will still use the digest from hashing the message with the value R to choose the leaf used in each FORS tree. Afterward, we can generate the hypertree indexes in two ways.

We can hash the FORS public key to obtain the indexes to be used in the hypertree, or we can have a FORS signature with one extra tree that does not contribute to retrieve the FORS public key but to derive the hypertree required indexes from its root node. Either way, this makes the previous attack invalid as it forces an adversary to generate a FORS signature or a FORS public key to retrieve the indexes. This is somewhat hard to do as an adversary can not easily generate a valid FORS signature from the message not knowing the secret

seed. Furthermore, using the message to generate the FORS signature means that even if the FORS public key is the same for two different messages (we found a collision), its FORS signature is most likely different and can't easily be used in a forgery.

To prove this point, let's assume an adversary generates random secret values for a fake FORS scheme and obtains a digest from a chosen message and R value. With this, he can create what seems to be a valid FORS signature. Now, regarding our two solutions, he can either hash the FORS public key to obtain the indexes required for the hypertree or generate another FORS tree and derive the indexes from its root node.

The first solution requires him to find a FORS public key that not only generates a pair of XMSS and WOTS+ key pair indexes that had already been used in a valid signature. But also to have a collision for every hash chain length in the respective WOTS+ key pair signature. The second solution forces an adversary to forge not only a FORS tree, whose root node generates a collision of the same indexes for an XMSS and WOTS+ key pair. But also to have a collision for every hash chain length in the respective WOTS+ key pair signature.

In the worst-case, we can assume that, in either case, the adversary can find a collision for the same indexes of the XMSS and WOTS+ key pair, which corresponds to successfully forging a FORS signature and having it generate the correct indexes for the XMSS and WOTS+ key-pairs. Unless the FORS public key is the same for a previously signed message by the oracle for those same indexes, the adversary still has to forge a WOTS+ signature which we know is EUF-CMA secure. With this in mind, we can assume our solution makes this scheme also EUF-CMA.

4 PICNIC+WOTS

PICNIC's signature is considerable due to the number of rounds needed to guarantee the scheme's security. Each round increases the signature's length by approximately 174 bytes per round. To reduce the number of rounds the WOTS+ scheme was implemented over PICNIC. This modification computes a WOTS+ secret value for each of the PICNIC's rounds using the computed commits in that round and PICNIC's secret key. Additionally, if it exists, the previous round's secret value is added as an input under the same WOTS+ signature. Furthermore, in the PICNIC-UR version, the UR commits computed are also used to generate the WOTS+ secret key in each round. Also, we propose that the challenge should be calculated using the WOTS+ public keys. That allows for the verifier to only store a WOTS+ hash value per round of the signature, increasing its efficiency and memory usage.

Unlike PICNIC, this version does not allow for parallel processing of each round, instead it allows for parallel processing of sets of rounds. Each set of rounds equals the WOTS+ number of chains as each chain's secret values depends on the previous round secret value.

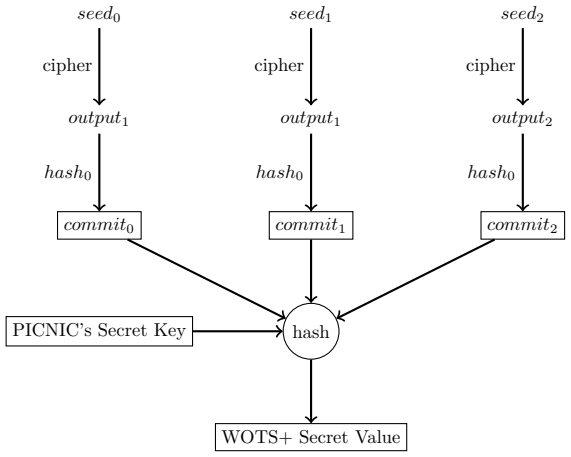

Figure 5: PICNIC's first WOTS+ secret value computation.

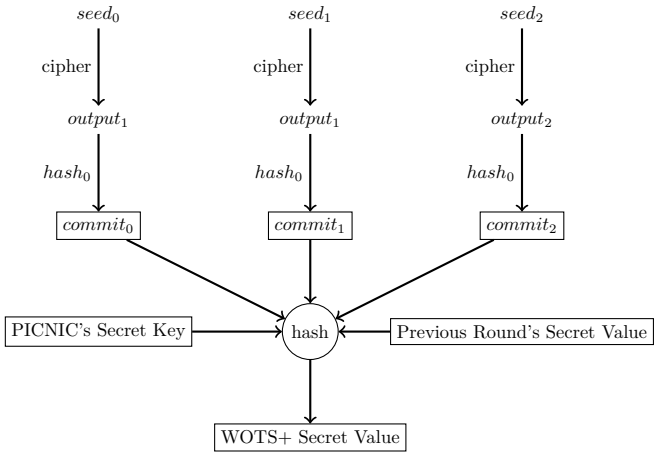

Figure 6: PICNIC's remaining WOTS+ secret values computation.

Several versions of these PICNIC modifications using different ciphers and hashes were devised and implemented. All the modifications aim at diminishing the number of PICNIC's rounds both for classical and post-quantum security. The original versions of PICNIC-FS and PICNIC-UR for both classical and post- quantum settings are summarized below.

Version	Number of rounds	Average Signature Size	Number of Runs	Real Time
PICNIC-FS Classical	219	32860	100	8.85
PICNIC-UR Classical	219	53961	100	9.22
PICNIC-FS Post-Quantum	438	65702	100	16.46
PICNIC-UR Post-Quantum	438	107890	100	18.84

* The times presented were measured in seconds.

Table 9: Original PICNIC versions' performance.

PICNIC's MPC clasical security is attained through the expression $(\frac{3}{2})^r$ from [8], where r is the number of rounds needed to achieve the desired security. The WOTS+ classical security can be obtained with the expression

$2^{n-log(lw)-log(2w+1)}$ from [9], where l in the number of chains in the WOTS+ signature and w is the length of each chain. By merging both scheme's security claimes we have the expression $(\frac{3}{2})^r + \frac{r}{l} * 2^{n-log(lw)-log(2w+1)} \geq 2^{128}$, were $\frac{r}{l}$ is the number of WOTS+ signatures used in the PICNIC scheme. As the main objective is to reduce PICNIC's signature lenght, the chosen parameters for the classical security were $w = 16$, $l = 5$ and $n = 17$ leading to $r = 55$. Note that, as each WOTS+ signature has 5 chains, PICNIC's rounds must be a multiple of 5. In the following tables W represents the chain length of a WOTS+ signatures, N represents the hash output lenght in a WOTS+ signature, and L represents he number of chains in a WOTS+ signature.

These settings applied to PICNIC-FS, resulted in signatures with a third of the original signature and a four to nineteen times faster algorithm, depending on the version. Regarding PICNIC-UR, the signatures are six to seven times smaller and up to eighteen times faster.

Cipher	Hash	Number of rounds	N	W	L	Average Signature Size	Number of Runs	Real Time
LowMC	SHAKE256	55	17	16	5	7450	100	2.26
LowMC	ASCON	55	17	16	5	7447	100	2.29
LowMC	SKINNY	55	17	16	5	7443	100	6.93
ASCON	ASCON	55	17	16	5	8824	100	0.54
SKINNY	SKINNY	55	17	16	5	8598	100	3.40

* The times presented were measured in seconds.

Table 10: Modified classical PICNIC-FS versions' performance without chains.

Cipher	Hash	Number of rounds	N	W	L	Average Signature Size	Number of Runs	Real Time
LowMC	SHAKE256	55	17	16	5	7449	100	2.23
LowMC	ASCON	55	17	16	5	7455	100	2.67
LowMC	SKINNY	55	17	16	5	7453	100	7.00
ASCON	ASCON	55	17	16	5	8828	100	0.91
SKINNY	SKINNY	55	17	16	5	8609	100	3.37

* The times presented were measured in seconds.

Table 11: Modified classical PICNIC-FS versions' performance with chains.

Cipher	Hash	Number of rounds	N	W	L	Average Signature Size	Number of Runs	Real Time
LowMC	SHAKE256	55	17	16	5	7456	100	2.99
LowMC	ASCON	55	17	16	5	7452	100	2.60
LowMC	SKINNY	55	17	16	5	7447	100	7.83
ASCON	ASCON	55	17	16	5	8836	100	0.96
SKINNY	SKINNY	55	17	16	5	8611	100	3.91

* The times presented were measured in seconds.

Table 12: Modified classical PICNIC-UR versions' performance without chains.

Cipher	Hash	Number of rounds	N	W	L	Average Signature Size	Number of Runs	Real Time
LowMC	SHAKE256	55	17	16	5	7451	100	3.03
LowMC	ASCON	55	17	16	5	7457	100	2.73
LowMC	SKINNY	55	17	16	5	7454	100	7.67
ASCON	ASCON	55	17	16	5	8825	100	0.94
SKINNY	SKINNY	55	17	16	5	8604	100	3.97

* The times presented were measured in seconds.

Table 13: Modified classical PICNIC-UR versions' performance with chains.

Our scheme's post-quantum security can be obtained through the expression $\sqrt{(\frac{3}{2})^r} + \frac{r}{l} * 2^{\frac{n}{2}-log(lw)-log(2w+1)} \geq 2^{128}$. As in the previous security expression,

both scheme's security claimes were drawn from [8] and [9]. To attain the minimum level of post-quantum security the parameters used were $w = 16$, $l = 5$ and $n = 34$ which also led to $r = 55$.

With these parameters, PICNIC-FS signatures are seven to eight times smaller than the original scheme's signature and 2 to 35 times faster to compute depending on the version. In the PICNIC-UR versions, the schemes produce signatures eleven to thirteen times smaller and are two to thirty-seven times faster. The following tables summarize the obtained results.

Cipher	Hash	Number of rounds	N	W	L	Average Signature Size	Number of Runs	Real Time
LowMC	ASCON	55	34	16	5	8395	100	2.75
LowMC	SKINNY	55	34	16	5	8380	100	9.31
ASCON	ASCON	55	34	16	5	9767	100	0.89
SKINNY	SKINNY	55	34	16	5	9545	100	4.70

* The times presented were measured in seconds.

Table 14: Modified post-quantum PICNIC-FS versions' performance without chains.

Cipher	Hash	Number of rounds	N	W	L	Average Signature Size	Number of Runs	Real Time
LowMC	ASCON	55	34	16	5	8387	100	2.55
LowMC	SKINNY	55	34	16	5	8390	100	9.18
ASCON	ASCON	55	34	16	5	8872	100	0.90
SKINNY	SKINNY	55	34	16	5	9542	100	4.77

* The times presented were measured in seconds.

Table 15: Modified post-quantum PICNIC-FS versions' performance with chains.

Cipher	Hash	Number of rounds	N	W	L	Average Signature Size	Number of Runs	Real Time
LowMC	ASCON	55	34	16	5	8398	100	2.24
LowMC	SKINNY	55	34	16	5	8385	100	8.92
ASCON	ASCON	55	34	16	5	9762	100	0.59
SKINNY	SKINNY	55	34	16	5	9546	100	4.61

* The times presented were measured in seconds.

Table 16: Modified post-quantum PICNIC-UR versions' performance without chains.

Cipher	Hash	Number of rounds	N	W	L	Average Signature Size	Number of Runs	Real Time
LowMC	ASCON	55	34	16	5	8393	100	2.23
LowMC	SKINNY	55	34	16	5	8396	100	8.99
ASCON	ASCON	55	34	16	5	8886	100	0.57
SKINNY	SKINNY	55	34	16	5	9543	100	4.67

* The times presented were measured in seconds.

Table 17: Modified post-quantum PICNIC-UR versions' performance with chains.

The previous tables show a great improvement in PICNIC's signature size and processing speed, especially when using the created ASCON cipher with ASCON-HASH. Several other parameter combinations are still being tested for their speed and security. Furthermore, if either ASCON or SKINNY ciphers are proven secure, these modifications provide an immediate improvement, both in performance and memory.

Note that, the use of LowMC with the ASCON-Hash is alredy a secure and much faster and shorter version of PICNIC.

5 Conclusion

The eminent possibility of usable quantum computers threatens the security level of current "number theoretic" public-key cryptosystems. To prepare for this eventuality, the cryptographic community took action and started a contest aiming to find post-quantum cryptosystems to prepare for this eventuality. Although new alternatives to replace these pre-quantum cryptosystems are under evaluation, the least powerful devices can't use them either because they lack memory space or acceptable processing speed. Among all the candidates, some are considered "high-algebra" and some "low-algebra". The "low-algebra" candidates use simple primitives that require low processing capacity but use them so many times that it takes too long to sign a message. Both the PICNIC and the SPHINCS+ schemes, can be adapted to "low-end" devices by using one of the lightweight primitives ASCON or SKINNY, as these were tailored to guarantee security and efficiency on such devices. Nevertheless, the most challenging part of the adaptation is to balance the assurance of security with low memory usage and good performance during a signature. As can be seen in the 2 section, speed and memory consumption are intertwined as decreasing one often increases the other. In SPHINCS+, using the ASCON-Hash primitive allows it to sign faster without increasing the memory used, thus contributing hugely to decrease the time elapsed during a signature. This time advantage also allows us to sacrifice some speed to save memory usage while still outperforming the original version of SPHINCS+ time-wise. Several combinations of parameters were taken into consideration when trying to attain a good trade-off between time and memory consumption. From all the possible combinations, the most promising ones are:

- $h = 33$,$d = 11$,$k = 33$, $w = 16$.

  This version takes 0.02 seconds to produce a signature with 3704 bytes.

- $h = 33$,$d = 11$,$k = 33$, $w = 256$.

  This version takes 0.1132 seconds to produce a signature with 3000 bytes.

If a low-end device has the primitives hardwired, the above time can be even further decreased for both implementations, making this scheme an even better suited candidate to ensure the security of low- end devices' communications. Furthermore, in this paper, a notion of security based on a device's life expectancy is presented. The viability of SPHINCS+ signatures relies on the single usage of each WOTS+ key pair. As soon as all the WOTS+ key pairs have been used, the scheme becomes insecure. This notion plays well with the duration of low-end devices as most are meant to be used for a short time before being replaced. Notice also that the minimum time before replacing a device that implements each version of the algorithm is 0.7 and 3.85 years respectively. But this only needs to happen if the device issues signatures nonstop throughout those years. It is also important to notice that the above results are safe as long as the attack previously pointed out is mitigated.

The SPHINCS+'s presented results not only show a way of fastening the original SPHINCS+'s signatures using lightweight cryptography (to be applied to "normal" devices) as also a way of allowing the least powerful devices to implement that same scheme using minimum memory and time with a decent life expectancy and security.

As for PICNIC, both lightweight primitives were adapted to fit the scheme and attained good results, greatly improving the scheme's performance. The biggest problem with PICNIC is its signature size, which renders it unsuited for devices with memory restraints. However, the PICNIC modification with the WOTS+ scheme improved the previous results time and memory-wise.

As it stands, if we consider using fifty-five rounds on PICNIC, and apply a WOTS+ signature over each set of 5 rounds with a 16 length chain and a security parameter of 17, we attain the first level of classical security for PICNIC. If the security parameter is 34, the scheme attains the first level of post-quantum security instead, while maintaining the number of rounds. Furthermore, using a WOTS+ signature every five rounds, allows for parallel computation of sets of five rounds. The speed the scheme can attain if computed in parallel is yet to be tested, but it is expected to improve even further than the modification improvements seen so far.

The improvements regarding PICNIC, have provided us with means to not only reduce its signature length but also to improve its speed by shortening the number of rounds as a result of applying WOTS+ over PICNIC. The current secure and best combination uses LowMC as the cipher and ASCON-Hash as the hash.

Both schemes presented show potential to be applied to "low-end" devices to grant them post-quantum security. Each scheme having its own advantages. Although the PICNIC scheme is faster than SPHINCS+, it has a bigger signature size. That makes PICNIC ideal for "low-end" devices prioritizing speed over memory and SPHINCS+ better suited for devices that prioritize memory efficiency over speed. However, using the modifications, still under evaluation, given to make PICNIC's signature shorter, PICNIC might have signatures with similar size to the ones of SPHINCS+ while maintaining or even further increasing its processing speed advantage.

References

[1] *https://csrc.nist.gov/Projects/post-quantum-cryptography*

[2] *https://csrc.nist.gov/Projects/lightweight-cryptography*

[3] Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Bas Westerbaan. *SPHINCS+: Submission to the NIST post-quantum project, v.3.* October 1, 2020.

[4] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. *Zero-Knowledge from Secure Multiparty Computation.* SIAM Journal on Computing 39, 3 (2009), 1121–1152.

[5] Greg Zaverucha. *The Picnic Signature Algorithm Specification Version 3.0.* September 30, 2020.

[6] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. *SKINNY-AEAD and SKINNY-Hash v1.0*.

[7] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. *Ascon v1.2*. March 29, 2019.

[8] Melissa Chase, David Derler, Steven Goldfeder, Daniel Kales, Jonathan Katz, Vladimir Kolesnikov, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Xiao Wang, Greg Zaverucha. *The Picnic Signature Scheme Design Document*. September 30, 2020.

[9] Mikhail A. Kudinov, Evgeniy O. Kiktenko, and Aleksey K. Fedorov. *Security analysis of the WOTS+ signature scheme: Updating security bounds*. February 18, 2020.

[10] Lorenzo Grassi, Daniel Kales, Christian Rechberger, and Markus Schofnegger. *Survey of key-recovery attacks on LowMC in a single plaintext/ciphertext scenario*. September 1, 2020.

[11] Subhadeep Banik et al. *Cryptanalysis of LowMC instances using single plaintext/ciphertext pair*. In: IACR Transactions on Symmetric Cryptology 2020.4 (Dec. 2020), pp. 130–146. doi: 10.46586/tosc.v2020.i4.130-146.

[12] Claude Carlet. *Vectorial Boolean functions for cryptography*. In: Boolean Models and Methods in Mathematics, Computer Science, and Engineering. Vol. 134. 2010, p. 398.

[13] M. Turan, K. McKay, D. Chang, C. Calik, L. BassHam, J. Kang, and J. Kelsey. *NISTIR 8369, Status Report on the Second Round of the NIST Lighweight Cryptography Standardization Process*. Computer Security Division. Information Technology Laboratory. 21 July 2021.

Appendices

A Security vs. signature size in PICNIC

The security of an MPC-in-the-head zero-knowledge protocol, as defined in 2.2, relies essentially on the immunity of the cipher against single plaintext/ciphertext attacks. Such immunity depends on the number of cipher's rounds which impact the size of the signatures. Therefore, it is important to discuss how we can reduce the number of rounds without compromising the zero-knowledge property of the proof of knowledge protocol (PKP).

The MPC-PKP requires interplayer messages for each execution of a non-linear component of the cipher; these messages are part of the signature; therefore, because one main objective of such design is to control the signature's size, one must also control the size and number of the interplayer messages. A low S-box algebraic degree along with a low density of non-linear components controls the size of the messages; the number of messages is further controlled by the number of iterations of each S-box: i.e. the number of rounds in the cipher.

The block-ciphers relevant for PICNIC use S-boxes of a small algebraic degree, namely 2. Therefore the number of rounds, as in any SPN architecture, dictates the overall algebraic degree d of the cipher, and this bounds the immunity against algebraic attacks [10]. A simple analysis shows that, in this case, we have $d \simeq O(2^r)$ for r rounds: a reduction of r affects directly the immunity against algebraic attacks.

Moreover, a low value of d will naturally increase the distance of the cipher to an "almost perfect non-linear function" [12] and consequently decreases its immunity against differential and linear attacks. For LowMC instances [11] similar observations can be made in relation to other higher-order attacks.

We can conclude that the same factors contributing to shorter signatures, low algebraic degree of the s-boxes, low density of non-linear components, and a small number of rounds, also contribute to an increasing vulnerability against those attacks which compromise the zero-knowledge property of the PKP.

According to [8, 10], when the LowMC cipher has a full S-box layer it needs only four rounds to achieve the first level of security on the PICNIC scheme. Much like the LowMC's S-box, the modified SKINNY's S-box can also be described as a second-degree boolean function. As such, one can argue that the analyses on the LowMC's S-box can be used to ensure a similar security level on the SKINNY's modified S-box.

The ASCON cipher is not an SPN; is a sponge construction where blocks of 8 bytes are transformed with a different number of rounds. Although each round has also an algebraic degree of 2, it is not possible to define an overall degree of the cipher. The reduction proposed for the number of rounds is just a first approximation, and further analysis must be performed on the security effects of such reduction.