Signing It Twice: Mitigating the Effects of State Reuse for Stateful Signatures

Niels Duif
Sentyron B.V.
Delft, The Netherlands
niels.duif@sentyron.com

Daan S. Meijer
Sentyron B.V.
Delft, The Netherlands
daan.meijer@sentyron.com

***Abstract*—stateful hash-based signature schemes like LMS and XMSS are built on the Winternitz one-time signature. The effect of state reuse for these schemes has been shown to be disastrous [1][2]. This paper shows that the signer can mitigate this effect if a state is reused only once. This is achieved by repeating the randomized hashing step until a hash value with specific properties is found. Our results show that at least 80 bits of security can be achieved in 99% of the key reuses, an improvement of 49 bits. This requires the signer to repeat the randomized hashing step 1.4 million times on average. Slightly lower security can be reached with much less hashing.**

Keywords—hash-based signatures, one-time signatures, two-message attacks, post-quantum cryptography

I. Introduction

A. Hash-based signatures

Traditional digital signature schemes are known to be vulnerable to quantum attacks. Hash-based signatures offer a robust alternative grounded in well-established cryptographic primitives. At their core, hash-based signatures rely on the security properties of hash functions. For sufficient output lengths these are widely believed to continue offering sufficient security against both quantum and classical attacks.

Several digital signature schemes have been standardized. Among these are the hash-based signature schemes LMS, XMSS, and SLH-DSA which is also known as SPHINCS+. The lattice-based signature scheme ML-DSA has also been standardized by the National Institute of Standards and Technology (NIST). NIST has extended their post-quantum standardization process with an additional fourth round to produce additional standards for post-quantum signature schemes.

Some hash-based signature schemes like LMS and XMSS are built on one-time signature schemes. This requires the signer to keep track of the *state* of the signature scheme – i.e. which one-time signatures have been used. This is crucial because each underlying one-time signature scheme must only be used once. It is well-known that misuse (state reuse or fault attacks) results in practical forgeries for these signature schemes [1][2].

State management is easy in theory but may prove hard in the real world where backup and restore are required, where disk writes may be just cache changes and where crashes and outages may occur.

To solve these problems, one may move to stateless signature schemes such as SLH-DSA [3], or few-time signatures schemes such as FORS, which is described in [3] but is not a digital signature standard of its own. Stateless schemes do not require keeping any state at all. Few-time signature schemes can be used as a stateful one-time signature for which the security only degrades slightly if a key is reused. However, moving to stateless or few-time signature schemes significantly increases the signature size and signature generation time. In addition, an implementer's choices are often restricted by standards so that not all these options are viable.

In some cases, the downside of statefulness can be controlled. For example, when using specialized cryptographic devices that provide sufficient assurance that the state cannot be reused. State management is also feasible for firmware signing, where the signer usually is in a controlled environment.

Because hash-based signatures are the most trusted post-quantum signatures, they are especially suitable high-security applications. Of course, these cases cannot tolerate the risk of state reuse, and measures will be taken to prevent this. Despite these measures, state reuse may still be conceivable in rare circumstances. These could be (multiple) procedural errors, hardware malfunction or single event upsets. Therefore, defense in depth is desirable or even required.

B. Our contribution

Our contribution changes the signing process to make it computationally much harder to find a forgery in case of state reuse. This requires no changes for the verifier and is fully compatible with the existing schemes LMS and XMSS.

C. Structure of this paper

The structure of this paper is as follows. We give the relevant background on the Winternitz one-time signature (WOTS) and state reuse in Section II. We then describe our approach to mitigate state reuse in Section III, and proceed to discuss the security of this approach in Section IV. Next, we present the resulting security in case of state reuse in Section V, and finally present our conclusions in Section VI.

II. Background

A. The Winternitz One-Time Signature

The Winternitz One-Time Signature (WOTS) [4] is a digital signature scheme. It is a proposal to reduce the signature size of Lamport's one-time signature scheme [5]. WOTS and its variants are used as a building block for hash-based signature schemes such as LMS, XMSS, and SLH-DSA.

This paper was originally presented at the International Conference on Military Communication and Information Systems (ICMCIS), organized by the Information Systems Technology (IST) Scientific and Technical Committee, IST-224-RSY – the ICMCIS, held in Bath, United Kingdom, 12-13 May 2026. This is an updated version from 26 June 2026.

WOTS generates a digital signature on a message m. It has a security parameter n which is equal to the output size of the hash function H that is used to sign the message. In addition, it has a WOTS parameter w such that $n = kw$ is an integer multiple of w. A WOTS instance has k private keys $sk_1, \dots sk_k$ for signing the message digits. Each private key has an associated chain of iterated hashes of length $W = 2^w$. The public key pk_i of each chain is the W-th hash of sk_i. The WOTS public key PK is computed by hashing all the pk_i into a single value. An overview of the hash chains from secret keys to public keys is given in Fig. 1. In most schemes, the secret keys are derived from a private seed to compress them.

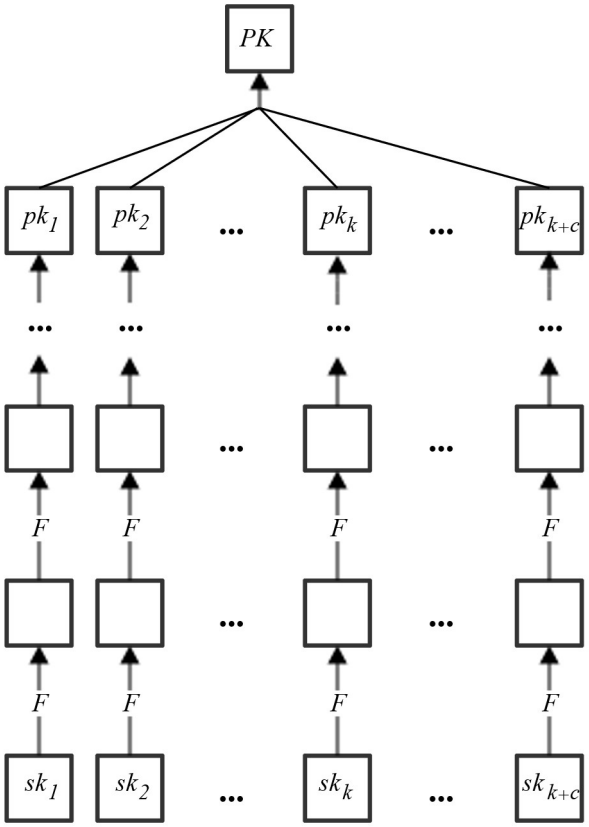

Figure 1: an overview of the Winternitz one-time signature

To sign a message m, the message hash is computed as $H(r|m)$ for a random value r. $H(r|m)$ is then split into digits $B_1, B_2, \dots, B_k$ of w bits each. These will be referred to as *message digits*, although they are actually message hash digits. Each message digit B_i is interpreted as an integer. The corresponding private key sk_i is hashed B_i times with an iterated hash function F. The WOTS signature consists of the values $F^{B_i}(sk_i)$, the value r, and a *checksum* that is introduced below.

For a given signature, it is impossible to compute values that are lower in the hash chains, but an attacker can trivially find all values that are equal or higher in the chain. To prevent an attacker from forging messages, a checksum is computed over the digits B_i. This checksum C is computed as $C = \sum_{i=0}^{k}(W - 1 - B_i)$. C is then expressed as a base W integer using just enough digits that it can fit the maximum checksum value of $k(W-1)$, which is $c = log_w\big(k(W-1)\big)$ digits. The WOTS scheme has c additional private keys $sk_{k+1}, \dots sk_{k+c}$. These are hashed $C_0, \dots, C_c$ times respectively, in the same way as the message digits, and added to the signature. The checksum is such that increasing any message digit decreases at least one digit of the checksum, making it infeasible to find a forgery. A security proof for WOTS can be found in [4].

To verify a WOTS signature, a verifier computes the message digits checksum digits from r and m. For each digit, the verifier hashes the corresponding part of the signature $W - B_i$ (or $W - C_i$) times, which should yield the public key pk_i for that WOTS chain. Finally, the verifier hashes the pk_i and verifies whether the result is equal to the WOTS public key PK. For a scheme that uses WOTS as a building block, the verifier instead uses the obtained WOTS public key as input for an authentication path.

WOTS is a one-time signature scheme, meaning that it can only sign a single message. Hash-based signature schemes typically use multiple instances of WOTS to create a many-time signature scheme. A Merkle tree is then used to combine the WOTS public keys into a single public key.

This section has introduced the basic version of WOTS where the chaining functions are all the same hash function F. We note that this paper also applies to versions of WOTS where these chaining functions are different, most notably to XMSS.

B. State reuse

The challenge of using a one-time signature as a building block is that it must only be used once. Signing multiple messages with the same private WOTS key is typically disastrous. Fluhrer shows that in most cases forgeries are practical already if two different messages are signed with the same WOTS key [2]. The reason is that an attacker will obtain two values for each WOTS chain, including the chains corresponding to the checksum. For different messages, the message digits will mostly differ. To create a forgery, it suffices to find a message hash for which each digit is at least the minimum of the two corresponding message or checksum digits. In the rest of this paper we assume that if state reuse occurs it occurs once: exactly two messages are signed with the same key.

The complexity of finding a forgery (in other words: the security level) after signing two messages with the same key greatly depends on the similarity of the message digits in the signatures. Security levels range from 10 to 90 bits depending on the signatures and the WOTS parameters, where similar message digits in the signatures increase the security by tens of bits [2]. The strongest effect is in the most significant bit of the checksum: if it is different for both signatures, finding forgeries is significantly easier.

III. Pinning the checksum

Our approach is to make the message digits of different signatures more similar. We keep sampling random values for r until the checksum of $H(r|m)$ has a desired fixed value, thus pinning the checksum. This checksum value is determined to balance the signature generation time and the security level in case of a single state reuse. It depends on the security parameter n and the WOTS parameter w. This is fully compatible with LMS and XMSS that already use randomized hashing. In addition to pinning the checksum, we experiment with bounding message digit values to further reduce the number of possible forgeries.

This approach is similar to WOTS+C [6] where the checksum is fixed to a predetermined value, and some message digits may be set to a fixed value too. In [6] the aim is to omit the values related to the checksum and the fixed message digits from the signature, thereby reducing the signature size. Another approach is described in [7] and [8] where the goal is to generate message digits with high values to reduce the signature verification time. Our goal is different: we pin the checksum to reduce the number of possible forgeries in case of state reuse. Unlike [6] we do not pin the checksum to a value that occurs frequently. Instead, we choose a value that significantly biases the message digits to limit the possibilities for a forgery.

Biasing the message digits means that we may need to try a large number of random values r until a message with the desired checksum is found.

In addition to pinning the checksum, we restrict the values of the first few message digits to be at least some predetermined value. This has the effect of shortening the WOTS chains for these message digits, thereby limiting the options for forgeries. The effect of restricting values and pinning the checksum are dependent: restricting values makes the message digits higher on average, while pinning the checksum makes them lower for high checksums and higher for low checksums. Without pinning the checksum, restricting a message digit to at least $W/2$ doubles the expected number of hashes. If the checksum is pinned to a high value this will more than double the number of hashes.

Like WOTS+C [6] we could omit the values related to the checksum from the signature and thereby reduce the signature size. This would require the verifier to assert that the checksum is equal to the predetermined value. This comes with the additional benefit that the checksums higher up in the WOTS chain cannot be used for forgeries. While this further increases the security in case of state reuse, this option is not fully compatible with LMS and XMSS so it is not our preferred choice.

IV. Security

A. Attack model

We view generating fresh random for randomized message hashing as an integral part of the signing process, which is the same assumption that [2] makes. This means that an adversary can make only random queries to the signing oracle. So, the attack model we are concerned with is existential unforgeability under a random message attack (EU-RMA). As we only consider reusing a WOTS private key once, the adversary is restricted to 2 queries.

We caution implementers that implementing randomized hashing in a trusted component is crucial. For implementations that are restricted in computing power, it is tempting to offload the randomized hashing externally as this is computationally expensive and appears to involve no sensitive data. However, this should not be offloaded to an untrusted component. If an attacker can provide the random used for randomized hashing, the attack changes from a random message attack to a chosen message attack (CMA). The approach of breaking EU-CMA in [1] shows that this makes it a lot cheaper to find an existential forgery: an attacker can prepare a large number of randomized message hashes and choose two of them for querying. These can be selected to maximize the number of possible forgeries. This is much more powerful than being restricted to query two random messages. In a practical attack, an adversarial hashing unit would ensure that consecutive randomized hashes have significantly different message digits. Then, if a state reuse occurs, finding a forgery on average is much less expensive than if the message hashing had been truly randomized.

Side-channels are not a concern for the repeated randomized hashing, provided the data that is signed is not sensitive. Repeated use of the private key should be prevented as this could make the implementation more vulnerable to side-channel attacks. Therefore, only the randomized hashing should be repeated until the desired checksum is found, not the complete signing step.

B. Biasing signatures

While our approach improves security in case of state reuse, it also biases the signatures. This has no effect on security. The proof is similar to that of SPHINCS+C [6].

V. Results

We determine the EU-RMA security after a single state reuse by counting the number of possible forgeries for many random message pairs. This models a state reuse where two messages are signed by the same private WOTS key. We divide the number of forgeries by the total number of possible messages to obtain a security level. Details of counting the number of possible forgeries are given in Appendix A.

For the results, we focus on the commonly used parameters $n = 256$ and $w = 4$ ($W = 16$). Among the parameters studied in [2], these give the highest security in case of state reuse when no checksum or bits are pinned, and they also give the highest security in our case. We did some experiments for other parameters. While our approach substantially increases security for all WOTS parameters, a significant fraction of messages remains practically forgeable for $n < 256$ or $w > 4$. Smaller values of w give better security, but these are not supported for XMSS and are not often used for LMS, as they produce significantly larger signatures. This is often not worth the reduced signing and verification time.

We successfully reproduced the results of [2] for $n = 256$ and $w = 4$ and included these in Fig. 2 for comparison against our other results. This figure is based on 1.1 million message pairs.

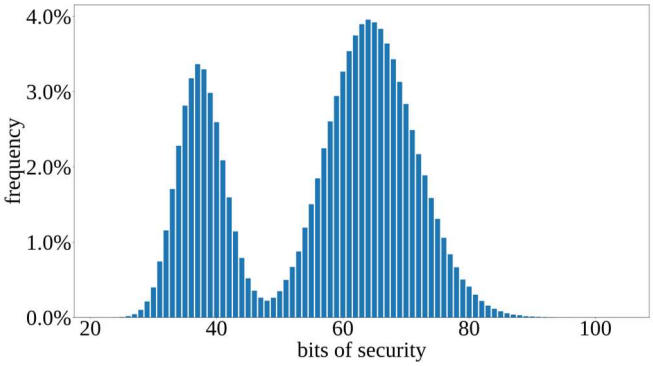

Figure 2: the security of signing two WOTS messages with the same private key for n=256 and w=4.

When we only pin the checksum, we get an increase in security compared to [2], see Fig. 3. The average security is around 86 bits, but there is a non-negligible probability that key reuse results in a security level below 70 bits. While one

could argue that not all adversaries may be able to break 70 bits of security, anything below 80 bits is commonly regarded as not offering any security at all.

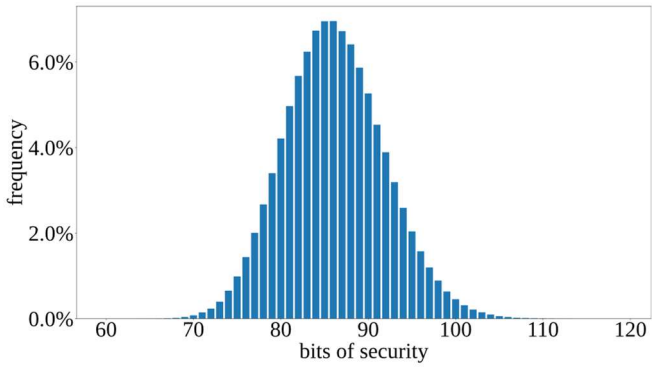

Figure 3: the security of signing two WOTS messages with the same private key when the signer fixes the checksum to 0x15f for n=256 and w=4.

Judging from the distribution in Fig. 3, we do not expect the probability of a practically forgeable message pair to become cryptographically negligible. Rather than looking for cryptographic security we focus on the median (P50) and the 1st percentile (P1) of the experimentally obtained distributions. At best, this gives us parameters that offer security in 99% of state reuses. In our simulations we could not determine percentiles below P1 reliably. Judging from Fig. 4, this would require more simulation runs. For Fig. 4 we simulated 55,000 message pairs for each checksum value. Fig. 3 are the data for the checksum value 0x15f presented as a histogram. For each checksum value, we try each of the 2 × 55,000 simulations with different random until the desired checksum is found.

The checksum value of 0x15f in Fig. 3 was not chosen by chance. We see in Fig. 4 that that this checksum value achieves a relatively high security. The red dots are the P50 and the green dots are the P1.

The two main effects in Fig. 4 are best explained by looking at the blue dots. These are the P50 when the checksum is restricted to a single value by the signer and also by the verifier. We find that:

- Selecting checksums away from the middle reduces the number of possible forgeries. This is because a high checksum corresponds to a low average value for the message digits, while a low checksum corresponds to a high average value for the message digits. This gives a forger fewer options for the message digits, combinatorically. This effect is symmetric around the average checksum value, but the graph is not, due to a second effect:
- Lower checksum digits correspond to a higher average value of the message digits in the signature. This means that a forger will have shorter WOTS chains for the message digits to work with. Therefore, forgeries for low checksums are more costly.

A third effect occurs when the checksum is not restricted by the verifier. This can be seen in the red dots. The red dots from 0x240 to 0x24f are increasing. If all signed messages have checksum 0x240, valid forgeries can have a forged checksum equal to 0x240 or higher in the WOTS chain. In this case, we find the majority of possible forgeries have forged checksum 0x240. A smaller fraction of the possible forgeries has forged checksum 0x241, followed by even smaller contributions with forged checksums 0x242, 0x243, etc. Instead, if signed messages have checksum 0x24f, we find the majority of the forgeries will have forged checksum 0x24f. There is a very small contribution of forgeries with the next forgeable checksum 0x25f, and even smaller contributions with larger forgeable checksums. For message checksum 0x24f, forgeries with checksum 0x24f are the only significant contribution. This can be seen in Fig. 4, where visually the red dot at 0x24f coincides with the blue dot at that value.

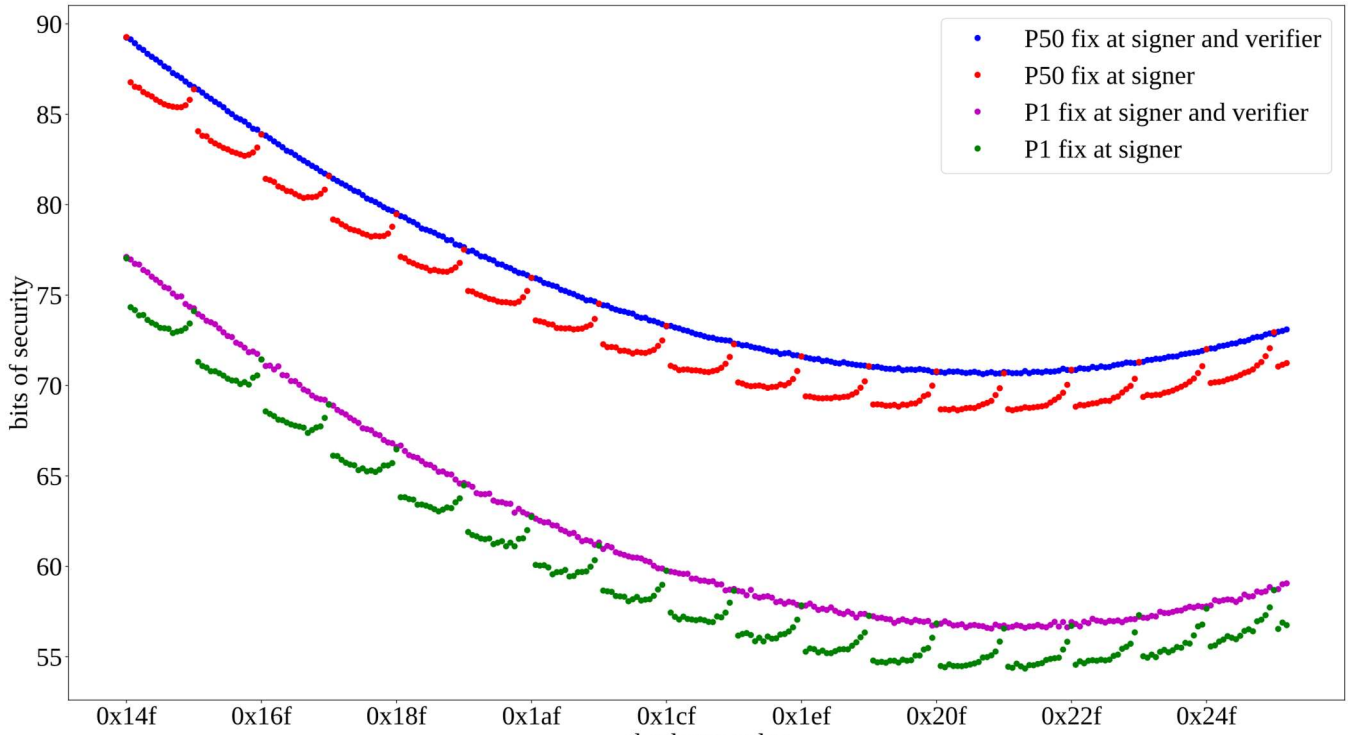

Figure 4: P1 and P50 security levels when fixing the checksum to a fixed value for n=256 and w=4.

In short, most forgeries have a forgeable checksum close to the original checksum. This can be explained as follows. All message digits in a forgery are lower bounded by the minimum of the two corresponding signature digits. On the other hand, the sum of the forged message digits is upper bounded by (the inverse of) the forged checksum. The lower the forged checksum, the higher this upper bound, and the higher the number of options for the forged message digits. So, the highest number of forgeries is at the lowest possible checksum, which is the original checksum.

According to our simulations, Fig. 4 covers 99.98 % of all checksums. The average checksum is 0x1e0. 0.004% of the checksums are below 0x14f, while 0.02% are above 0x262.

We chose the checksum of 0x15f for Fig. 3 as the checksum with the highest security that requires at most 2^{16} hashing attempts on average.

In Fig. 4 we include the case where the verifier requires the checksum to be equal to the given value. These are the blue dots for the P50 and the pink dots for the P1. This increases security as a forger now only has one checksum to work with. For this case, checksums ending in 0xf are no longer local maxima. We see that for checksums ending in 0xf, pinning the checksum by the verifier hardly improves security.

While a checksum of 0x15f gives good security, it also requires many hashing attempts to find a message hash with that checksum. When we take a checksum that occurs more frequently, we can increase security by fixing some values of the message digits. We tried a range of parameter sets that all take about 2^{16} hashing attempts to find the set with the highest security. The results are given in Table I. We also included some options that require more than 2^{16} hashing attempts for those who can spare the extra computational effort. The average number of hashing attempts to find a message with the desired properties follows an exponential distribution. We determined the mean of this distribution by dividing the median by $\ln(2)$, as the median is more stable to outliers. The number of hashing attempts reported in Table I is the mean value. For the restrictions imposed on the message digits, the notation "$\geq v \geq v$" means that the first two digits are both restricted to at least v.

We conclude from Table I that restricting message digits is less effective than choosing lower checksums. When spending about 45,000 hashing attempts the best results for $w = 4$ are found if the checksum is pinned to 0x15f with no restrictions on the message digits. In fact, we see that pinning message digits increases computational costs faster than security. For example, for the checksum 0x1ff, restricting seven message digits costs about 2^8 times as many hashes while it only increases security with 4 bits. Restricting message digits to higher or lower values did not give significantly different results.

We also see in Table I that lower bounding some message digits costs fewer additional hashing attempts if the checksum is low. This makes sense, since a low checksum means a high average value for the message digits.

Security for $w = 2$ is better than for $w = 4$, but this comes at the cost of roughly doubling the signature size. Also, this choice for w is only compatible with LMS, not with XMSS.

Table I shows that lower bounding some message digits costs fewer additional hashing attempts if the checksum is low. For the checksum 0x16f the restrictions on the message digits alone would reduce the message space by a factor of 10.7. The number of hashes increases roughly by a factor 4. For the higher checksum 0x1ff restrictions on the message digits would reduce the message space by a factor 128. The number of hashes increases roughly by a factor of 350. It makes sense that lower bounding message digits is cheaper for low checksums, because a low checksum means a high average value for the message digits.

TABLE I. WOTS SECURITY FOR N=256 UNDER ONE-TIME PRIVATE KEY REUSE WITH FIXED CHECKSUM AND RESTRICTED DIGIT VALUES

w	Check-sum	Message digit restrictions	Security (bits)		Hashing attempts
			P1	*P50*	
2	0x08f	-	92	104	60,700
2	0x0a3	-	81	94	440
4	0x13f	-	80	92	1,470,000
4	0x14f	-	77	89	230,000
4	0x15f	-	74	86	46,400
4	0x16f	-	71	84	10,700
4	0x16f	$\geq 8 \geq 8 \geq 8 \geq 4$	72	85	42,000
4	0x1ff	-	58	72	136
4	0x1ff	$\geq 8 \geq 8 \geq 8 \geq 8 \geq 8 \geq 8 \geq 8$	62	76	45,000
8	0x0aff	-	49	59	100,000

Instead of pinning the checksum to a single value, we can save hashing attempts by allowing multiple values. For $w = 4$, instead of pinning the checksum to 0x15f, we can allow all checksums ending in 0xf that are at most 0x15f. This reduces the number of hashing attempts from 46,400 to 36,700, while it does not affect security. We obtain the same security for this case as for $w = 4$ and checksum 0x15f.

While pinning the checksum makes signing much more expensive, signature verification becomes slightly faster. This is because we pin the checksum to a small value, so the message digits are above average. This makes shorter WOTS chains for signature verification.

We see in Table I that reaching at least 80 bits of security in 99% of key reuses requires 1,470,000 hashing attempts on average. This is the same order of magnitude as the number of hashes required for key generation of a height 10 LMS or XMSS tree, or the number of hashes required for signing if the signer uses no caching. On the other hand, a signer who caches the entire LMS tree needs $n \cdot 2^w/2w$ hashes on average plus a few additional hashes for the message and the LMS tree. For $n = 256$ and $w = 4$ this is just over 512 hashes. So, for some signers the cost of pinning the checksum may be too large. In that case a signer either needs to accept a lower security level in case of key reuse or should move to even smaller values of w at the cost of increasing the signature size. We note that using the P1 as a cutoff is quite arbitrary, so a signer can choose to have reasonable security in only 95% of the cases instead of 99%. We expect a signer willing to hash even more messages can reach 80 bits of security in 99,9% of all cases, but it would require additional simulations to accurately

determine for which checksum this is reached. While distributions like that in Fig. 3 look normal at first glance, it is risky to assume normality and extrapolate the security level to low percentiles. We recommend performing extensive simulations to determine the security of other parameters than those given in this work.

For the multi-layer schemes HSS and XMSS^{MT}, the approach in this paper offers limited defense in depth. The benefits of pinning checksums only exist at the lowest layer of the tree. Reusing a signing key at any of the higher levels remains as catastrophic as usual. Signatures at higher levels in the tree do not use randomized hashing so our approach cannot be applied there. Without changing the specifications of HSS and XMSS^{MT}, pinning the checksum at higher levels would require redoing (part of) the generation of the subordinate tree until a public key is found that gives a desirable checksum.

VI. Conclusions

We show that a signer can mitigate the effects of a single private key reuse for one-time signatures. This significantly improves security. A security level of 80 bits can be reached in 99% of all key reuses at the cost of additional message hashing.

We emphasize that implementers still need to avoid reuse at all costs. While 80 bits of security cannot be broken by all adversaries, it is generally not regarded as a future-proof security level. More importantly, having reasonable security in 99% of all reuses sounds nice, but has no cryptographic relevance: 1% of the reuses may still be exploitable. If an attacker can somehow trigger a scenario where reuse is likely, this quickly leads to practical attacks. Therefore, our approach should only be used as defense in depth after all reasonable measures have been taken to avoid reuse.

Acknowledgment

We thank Scott Fluhrer for providing the source code he used for [2] and for sharing his ideas on our research.

References

[1] L. Groot Bruinderink and Andreas Hülsing, “Oops, I did it again – security of one-time signatures under two-message attacks”, SAC 2017. LNCS vol 10719. Springer, Cham. https://doi.org/10.1007/978-3-319-72565-9_15.

[2] S. Fluhrer, “Oops, I did it again revisited; another look at reusing one-time signatures”, Cryptology ePrint Archive, paper 2023/1905. Available: https://eprint.iacr.org/2023/1905.pdf

[3] NIST and D. Cooper, “Stateless hash-based digital signature standard”, 2024, Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD, [online], https://doi.org/10.6028/NIST.FIPS.205, https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=958464 (Accessed December 9, 2025)

[4] Ralph C. Merkle. “A certified digital signature”, in G. Brassard, editor, Advances in Cryptology — CRYPTO’ 89 Proceedings, pp.218–238, NY, 1990.

[5] L. Lamport, “Constructing digital signatures from a one way function”, Technical report, Computer Science Laboratory, SRI International, 1979.

[6] M. Kudinov, A. Hülsing, E. Ronen, and E. Yogev, “SPHINCS+C: compressing SPHINCS+ with (almost) no cost”, Cryptology ePrint Archive, paper 2022/778. Available: https://eprint.iacr.org/2022/778.

[7] L. P. Perin, G. Zambonin, D. M. B. Martins, R. F. Custódio, and J. E. Martina, “Tuning the Winternitz hash-based digital signature scheme”, in Symposium on Computers and Communications – ISCC, pp.537–542. IEEE, 2018.

[8] J. Bos, A. Hülsing, J. Renes, and C. Vredendaal, “Rapidly verifiable XMSS signatures”, IACR Transactions on Cryptographic Hardware and Embedded Systems. 2021. 137-168. 10.46586/tches.v2021.i1.137-168.

Appendix A: Computing the Number of Possible Forgeries

To compute the security level for two messages signed with the same key we adopt the strategy of [2] . We have rewritten Appendix A of [2] as it reuses the variable j resulting in a minor error in the index of the summation.

Given two messages m_1 and m_2 signed with the same private WOTS key, we want to compute the probability $P_{\text{forge}}(m_1, m_2)$ that the information in the signatures can be used to forge a signature on third message m_3. We model the hash of m_3 as a sequence of random digits due to randomized hashing.

The message digits of m_1 are $B_{1,1}, \dots, B_{1,k}$ and the message digits of m_2 are $B_{2,1}, \dots, B_{2,k}$, where $k = n/w$. We denote candidate message digits for a forgery as $B_{3,i}$ for $1 \le i \le k$.

We compute the number of possible forgeries for each forgeable checksum C separately. To simplify the equations, we work with the message sum $S = \sum_{i=1}^{k} B_i$. This can be computed from the checksum as $S = k \cdot (W-1) - C$.

A forged message digit $B_{3,i}$ must be at least the minimum of $B_{1,i}$ and $B_{2,i}$, so we have

$$B_{3,i} \ge \min\left(B_{1,i}, B_{2,i}\right) \text{ for } 1 \le i \le k. \tag{1}$$

Given a message sum S, we work back from the last message digit $B_{3,k}$. Each possible value v for this digit gives a number of forgeries equal to the number of possible forgeries with sum $S - v$ for the remaining digits. We denote the number of forgeries for i message digits with sum s as $F_{i,s}$. We then obtain the number of forgeries as

$$F_{i,s} = \sum_{v=\min\left(B_{1,i}, B_{2,i}\right)}^{W-1} F_{i-1,s-v} \quad \text{for} \quad i > 1,$$

$$F_{1,s} = 1 \quad \text{if} \quad \min\left(B_{1,1}, B_{2,1}\right) \le s \le W-1,$$

$$F_{1,s} = 0 \text{ otherwise.} \tag{2}$$

Equation (2) suggests computing $F_{k,S}$ recursively. It is more efficient to start with the list of nonzero $F_{1,s}$ and then iteratively compute all nonzero $F_{2,s}$, $F_{3,s}$, and so forth.

To find the total number of forgeries, we enumerate all forgeable message sums S and sum the corresponding $F_{k,S}$. The probability that a random message m_3 is a valid forgery equals the probability that it satisfies the requirements of (1) while it also has a forgeable checksum. This is the number of possible forgeries divided by the number of possible messages 2^n, so

$$P_{\text{forge}}(m_1, m_2) = 2^{-n} \sum_{\text{forgeable } S} F_{k,S}\,. \tag{3}$$

The associated security level is the expected number of random messages for a valid forgery, which is $-\log_2 P_{\text{forge}}$ bits.