Oops, I did it again revisited; another look at reusing one-time signatures

Scott Fluhrer
Cisco Systems
sfluhrer@cisco.com

December 11, 2023

Abstract

In "Oops, I did it again" - Security of One-Time Signatures under Two-Message Attacks[GBH18], Bruinderink and Hülsing analyzed the effect of key reuse for several one time signature systems. When they analyzed the Winternitz system, they assumed certain probabilities were independent when they weren't, leading to invalid conclusions. This paper does a more correct characterization of the Winternitz scheme, and while their ultimate conclusion (that key reuse allows for practical forgeries) is correct, the situation is both better and worse than what they concluded.

1 Introduction

With the possibility of Quantum Computers, we need signature algorithms which are postquantum, and one potential answer to this are hash based signatures. These are signature algorithms which are based on a hash function, which rely on the strength of an underlying hash function to provide its security. Among the current hash based signature algorithms are LMS[MCF19], XMSS[HBG$^+$18], and the SLH-DSA[1][Dwo23]. These three all rely on the security of the one time signature primitive they use internally.

This one time signature primitive has the property that a private key can securely sign a single message (assuming the underlying hash function is strong), however the security proof breaks down if two distinct messages are signed by the same private key. Used correctly, LMS, XMSS and SLH-DSA never sign two different messages using the same one time signature private key. However, on misuse (state reuse in the case of LMS and XMSS, fault attack in the case of SLH-DSA), it is possible that two different messages will be signed by the same one time signature private key.

This paper concentrates on the effects of one-time signature reuse in the context of LMS, XMSS, SLH-DSA. As shown in Appendix B, being able to generate such a forgery due to one-time signature reuse allows the attacker to generate forgeries for arbitrary messages for these three signature algorithms.

LMS, XMSS and SLH-DSA all use the same Winternitz one-time signature scheme, and so we will limit our study to that case.

1.1 The Winternitz One-Time Signature Scheme

The Winternitz scheme[Mer90] is a one time signature scheme. Figure 1 shows the overall structure, where the arrows indicate hash functions, and the boxes are internal values. The asterisks indicate values that are exposed as a part of the signature.

The private key is used to generate a number of private values. Each private value is hashed a number of times ($W-1$ times, where W is referred to as the Winternitz parameter)[2], and then tops of all the chains are hashed together to form the public key.

[1] Also known as Sphincs+.

[2] In practice, each invocation of the hash function includes a distinct tweak to avoid potential multitarget preimage attacks; that detail is unimportant for this discussion.

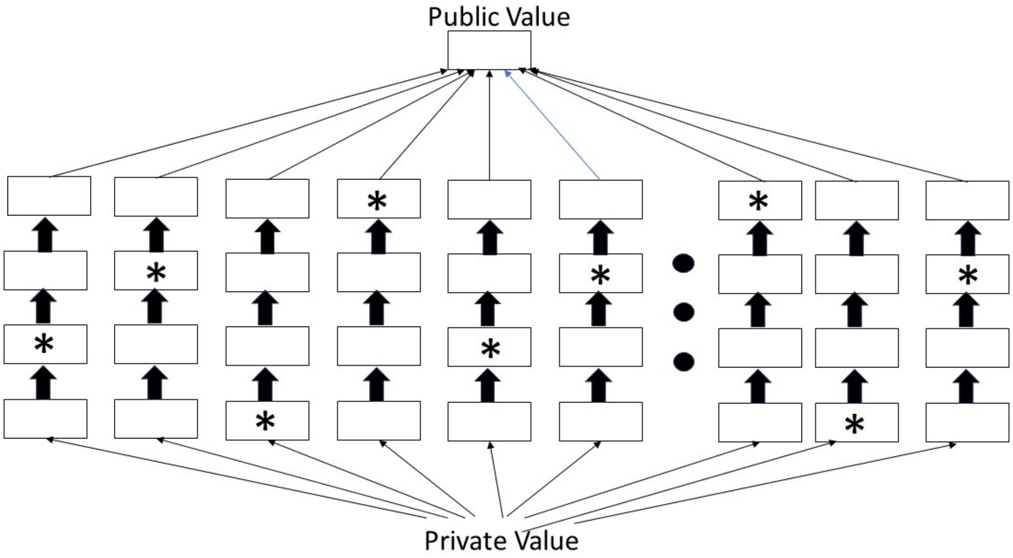

Figure 1: The structure of a Winternitz signature.

To generate a signature of a fixed length message[3], the method we will consider[4] parse that message into a series of $\log_2 W$ bit length values $\mu_0, \mu_1, \ldots$. These values will be referred to as the *message digits*. We also compute the *inverse checksum* of the digits; that is (where c is the number of message digits), that is, $\sum_{i=0}^{c-1} W - 1 - \mu_i$.

That inverse checksum is converted into a base W value, and the result is another set of digits. These digits will be referred to as the *checksum digits*. The number of checksum digits is just large enough so that the maximum value of $c(W-1)$ can be expressed.

The value of each digit (both the message and checksum digits) is understood as a position within the Winternitz chain, and the signature contains the value at that position in the chain.

In Figure 1, if the first digit of the message is a 1, then we place the value at position 1 into the signature (as indicated by the asterisk in the leftmost chain). We do the same for each digit.

Because the hash function (indicated by the arrows) is one-way, someone with the signature is unable to obtain previous values in the chain, and so is unable to sign a message with a smaller digit value in any location. For any message other than the one that is signed, there will always be at least one digit which is smaller. If all the message digits are equal to or larger (and the messages are distinct), then then inverse checksum will be smaller, which implies that at least one if the checksum digits will be smaller. Hence, to generate a forgery from a single signature would require the adversary to find a preimage or second preimage to a hash generated by the hash function, which we assume is infeasible.

The security of this system is more fully analyzed by Buchmann et al[BDE+11].

However, if we generate two signatures for two different messages with the same private key, this analysis does not hold. It may be possible for the adversary to find a message where each digit (for both the message and checksum digits) is larger or equal to the smaller of the corresponding digit in the two signed messages. If this happens, it becomes straightforward to generate a forgery to that message.

One question that we will address is 'how feasible would it be to find such a message?'

[3] In all three signature schemes, this fixed length message will be the output of a previous hash function.

[4] There are alternative methods that implement the same single-message signature protection, but this paper only considers this algorithm used by LMS, XMSS and SLH-DSA.

2 Previous Analysis

Bruinderink and Hülsing analyzed the effect of key reuse[GBH18] for three specific one time signature schemes: the original Lamport scheme[Lam79], an optimized version of Lamport[Mer90][5] and general Winternitz[Mer90].

Table 1 gives their security estimates in the five parameter WOTS parameter sets used in LMS, XMSS, SLH-DSA. These values were computed from the EU-CMA formulae found in section 5.2, with the listed security being $-\log_2 \Pr[\text{Break}(M_1, M_2, M3) = 1]$

n	w	security
128	16	18.21
192	16	26.53
256	16	34.85
192	256	15.10
256	256	19.75

Table 1: Estimated EU-CMA security levels of WOTS parameters for key reuse derived from "Security of One-Time Signatures Under Two-Message Attacks"

Bruinderink et al derives exact probabilities for the Lamport scheme, however when it comes to the others, they note that:

> For the optimized Lamport scheme and WOTS analysis becomes extremely complex when looking at the actual mapping function. This is caused by a checksum which is added to the message. This checksum introduces a lot of dependencies between probabilities, eventually leading to sums with an exponential number of summands. Therefore, we decided to analyze a simplified variant where we assume that the checksums are independent and uniformly distributed.

However, the checksum digits are in fact not independent of the message digits. In particular, the high order checksum digit may be strongly dependent on the message digits. Hence, the variant they analyze may not behave the same as the one-time signature used in practice. As such, it is not clear that such a simplification does not significantly alter the results.

Here is one way to see that it may be inappropriate to treat the checksum digits as random: consider the case that we have the signature to a single message, and we wish to compute the probability that another random message can be forged by the one signature. If we model the checksum digits is random (as in their variant), then the second message can be forged if each digit (both message and checksum digits) in the random message is greater than or equal to the corresponding digit in the known signature message. If there are m message digits and c checksum digits, this probability is $(1/2 + 1/2w)^{m+c}$. However, we know that, in the real system, we can 'forge' only if the random message happens to be exactly the same as the known signature message. This is $(1/w)^m$, which is far smaller.

This discrepancy indicates that their simplification may yield misleading results in the reuse case as well.

3 Our contribution

We first note that in LMS, XMSS, SLH-DSA, the message that is signed by the one time signature is always the hash of some value that the adversary cannot fully control. Even in the case of the hash of the original message to be signed, both LMS and XMSS has the signer include an unpredictable random number in the hash, resulting in a hash output that cannot be predicted or controlled. In the case of SLH-DSA, any one-time-signature reuse would be result of a fault; as long as the fault does not occur in the last stages of the final hash computation, the two signed hashes will be unrelated[6]. In all

[5] Which is essentially Winternitz with $W = 2$.

[6] The situation where the fault is in the final rounds of the hash computation is beyond the scope of this paper. Given an appropriate model of such a fault, the methods used by this paper should be able to address that situation as well.

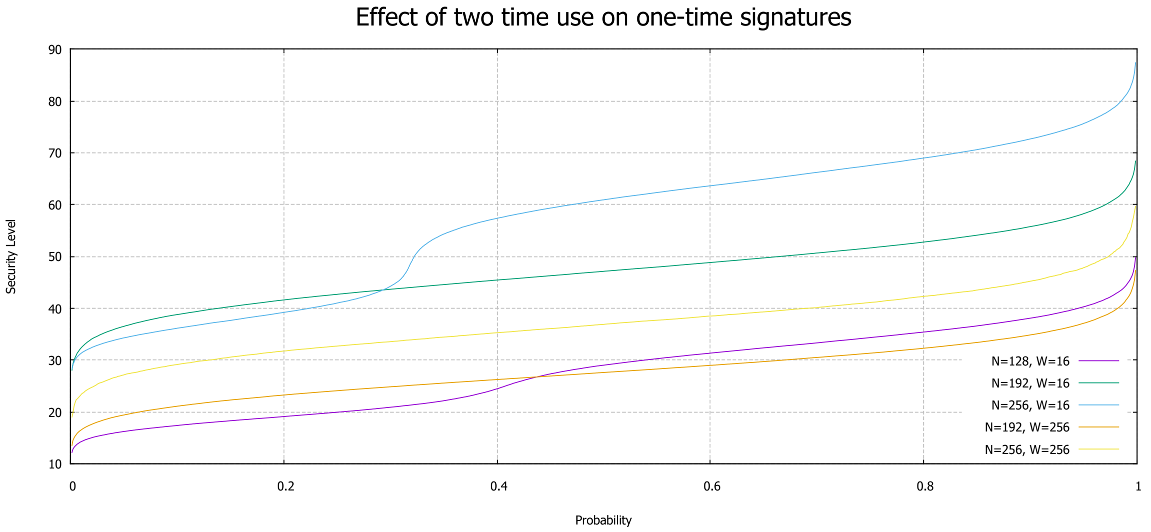

Figure 2: The probability of various security levels after reuse for the one-time signature parameter sets under discussion.

cases, it would appear that the appropriate model when analyzing reuse that may occur in practice is that the one-time signature signs two random unrelated messages.

In addition, because any message that the adversary would use would also be the output of a hash function, the adversary would need to find a preimage to a value whose signature can be generated using the two valid signatures. That is, the adversary cannot arbitrarily select the third message; instead, he needs to do probing to find the appropriate value that hashes to a usable (that is, forgable) message.

If two random messages are selected, it would appear that the difficulty in using those two messages to be able to craft a third would depend on what those two messages are. For example, if the two messages just happen by chance to be similar (differing by a small amount in their digits), it may be difficult to find a preimage to a third message that can be forged (because all the digits would need to fit in narrow ranges). In contrast, if the two messages just happen to be exceeding different (most of the digits of the two messages differ by a large amount), it may be considerably easier (because the range the digits would need to fit in is larger).

This is an unusual situation in cryptography. Normally, the security level of a cryptographical system is a fixed value; for example, if we have an n bit symmetric cipher with no weaknesses, we have a security level of n bits. In the scenario we are discussing, however, we have a random process (that neither the user or the adversary can control) that gives the adversary a task whose expected work effort is variable. Hence, what we have is a probability distribution of the security level; much of our discussion will be about this distribution.

To evaluate this security distribution, we observe that, given two specific messages A, B, it is practical to compute the exact expected work effort needed to find a third message that can forged given the two one-time signatures of A, B. Such an algorithm is described in appendix A. This algorithm takes into account the dependency between the message digits and the checksum digits.

There is no obvious way to convert the outputs of this algorithm into a mathematical closed form. What we can do instead is run the algorithm on a large number of random message pairs, and examine the results, giving us the security distribution to an arbitrary level of precision. We ran the algorithm on a large number of message pairs[7] for the various parameter sets that are used in LMS, XMSS and SLH-DSA, sorted the expected work efforts and graphed them; the results are in Figure 2.

Here is how this figure can be interpreted: the y axis is the security level that results from the information disclosed by the two signatures, and the x axis is the probability that the security will be at most that level. When two random messages are signed, what effectively happens is a uniformly

[7] We generated the message pairs by computing the SHA-256 hash for messages of the form "Good message 12", and "Good message 13", with each pair consisting of consecutive even and odd values.

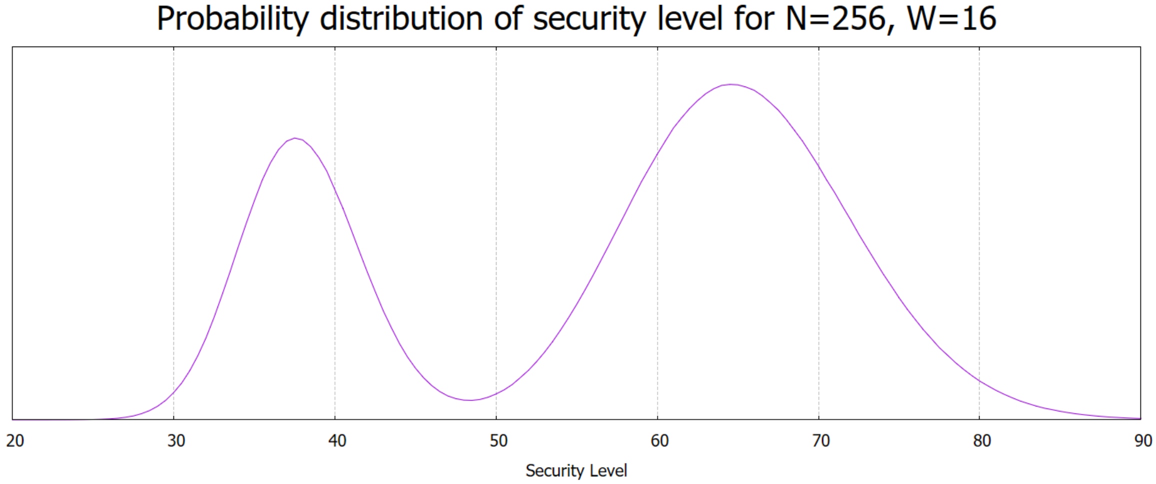

Figure 3: Distribution of security probabilities due to reuse

random position in the x axis is selected; the resulting security level is given by the height of the curve at that point. For example, for the N=256, W-256, there is a probability of approximately 0.2 that the work effort to find a forgery (given the signatures of two random messages) will be at most 2^{40} hash evaluations.

When we examine this graph, the first thing we notice is that the security levels that we obtain is well below what is considered necessary to be secure. Even the strongest case (N=256, W=16) only has a 1.4% probability of having at least 80 bits of security strength.

If we compare these levels to the values computed by the formula in the Bruinderink work (table 1), we see that their work is, in general, pessimistic. For example, in the W=16, N=192 case, the formula they give shows :a security estimate of 26.53 bits of security. However, examining the curve, we find that the actual security is actually greater almost all the time. Similarly, in the W=256, N=256 case, the formula gives a security level of 19.75, however we find the actual security is greater than that about 99.8% of the time, and in the W=256, N=192 case, the security is greater about 99.3 % of the time. The other two cases (W=16, N=128 and W=16, N=256) aren't nearly as dramatic, but in all cases, the average security is significantly higher than what they have shown. However, even though they are better than previously estimated, they are not good enough to be considered secure.

The next thing we note when we examine this graph is that there is a distinct curve downwards at the left and a curve upwards to the right. This corresponds to a low probability event where (at the right side) the two messages just happen to be quite similar (and hence makes the task of finding a forgable third message harder than expected). And, at the left side, the curve goes downwards in the case of the low probability event the two messages just happen to be mostly complementary, making the task of finding a forgable third message easier.

The downward curve at the left end shows that, given the signatures of two random messages, there is a small but nontrivial probability that the security would be significantly worse than expected. That means that if the adversary gets two random messages signed, he has a small probability of those two messages yielding a significantly easier forgery possibility than expected. Since we desire to have good security with high probability, this indicates that (except in the W=16, N=192 case) the practical security has a small but nontrivial probability of being even less than what the Bruinderlink result indicates.

Another notable aspect of the graph is the N=256 W=16 curve, and the N=128 W=16 curve; instead of having a smooth curve, they have two plateaus. We can see this even better by looking at the probability distributions shown in Figure 3 and Figure 4, where we see two distinct peaks[8].

If we examine the cause of these two peaks, we find that it happens because of the most significant check digit. For N=256 W=16, tt turns out that, given a random hash, the most significant checksum digit is 1 approximately 80.3% of the time, and 2 approximately 19.7% of the time (and all other

[8] These tables were again generated by selecting a large number of random message pairs, and running the algorithm to find the expected security distribution.

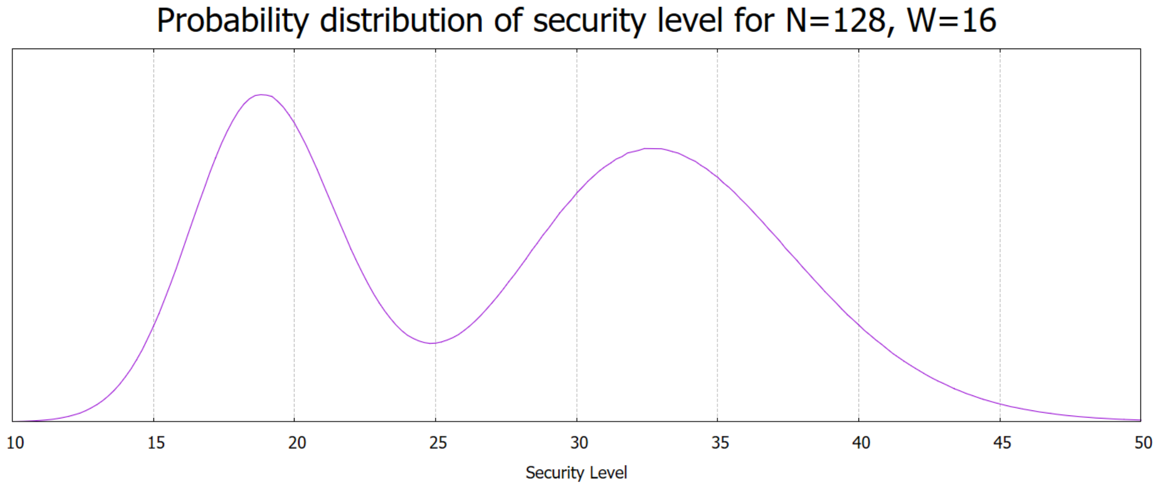

Figure 4: Distribution of security probabilities due to reuse

values happen with negligible probability). What this means is that, given two random messages, the resulting most significant checksum digits will be the same (either both 1 or both 2) about 68.4% of the time and differ (one being 1, the other being 2) about 31.6% of the time.

If we investigate the pairs that fall into the smaller left hand peak, we find that they correspond to message pairs where the most significant checksum digits differs. And, the pairs that fall into the larger right hand peak are message pairs where the most significant checksum digit is the same.

When the two most significant checksum digits differs, then what usually happens is that is that the second digit of the checksum with a 2 is quite small, which significantly increases the probability that a forged message would have a checksum that can be forged. So, this case resembles the scenario that Bruinderink analyzed, where the digits were mostly independent.

In contrast, when the two most significant checksum digits are the same, this significantly reduces the probability that a third message would be acceptable.

The case of N=128, N=16 has a similar (if less dramatic) ”two-peak” probability distribution.

In this case, the most significant checksum digit is 0 with probability 72.3% and 1 with probability 27.7% (and all other values with negligible probability). This yields the probability of two random messages having the same most significant checksum digit with probability 59.9% and differing with probability 40.1%.

The same effect yields the two observed peaks.

4 Conclusions

We see that the Bruinderink paper underestimates the average security that still remains when you sign two different messages with the same one time private key. On the other hand, for most of the parameter sets examined, there is a small but nontrivial probability that the security will end up being worse; hence in that sense, the situation is even worse than what was originally portrayed.

On the other hand, their main conclusion - that it is imperative to avoid reuse - still remains. The security level that remains is usually within the capabilities of practical adversaries. And, this reduction in security level directly translates to forgeries being practical within SLH-DSA, LMS and XMSS.

References

[BDE$^+$11] Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the winternitz one-time signature scheme. In Abderrahmane Nitaj and David Pointcheval, editors, *Progress in Cryptology – AFRICACRYPT 2011*, pages 363–378, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Dwo23] Morris Dworkin. Stateless hash-based digital signature standard, 2023-08-24 2023.

[GBH18] Leon Groot Bruinderink and Andreas Hülsing. “oops, i did it again” – security of one-time signatures under two-message attacks. In Carlisle Adams and Jan Camenisch, editors, *Selected Areas in Cryptography – SAC 2017*, pages 299–322, Cham, 2018. Springer International Publishing.

[HBG+18] Andreas Huelsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, May 2018.

[Lam79] Leslie Lamport. Constructing digitia signatures from a one way function. Technical report, Computer Science Laboratory, SRI International, 1979.

[MCF19] David McGrew, Michael Curcio, and Scott Fluhrer. Leighton-Micali Hash-Based Signatures. RFC 8554, April 2019.

[Mer90] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO’ 89 Proceedings*, pages 218–238, New York, NY, 1990. Springer New York.

A Computing the expected security level given two messages signed with the same private key

Here is the scenario that we are estimating: we are given two messages M_1, M_2 and their signatures with the same one-time private key. The adversary has the capability of generating random messages M_3, and we want to obtain the expected number of messages he must generate before he finds a message that can be forged by the joint information in the two signatures.

If a potential M_3 is n bits, and of those 2^n possible messages, there are k messages that can be forged, then the expected number of samples needed before finding a forgable message is $2^n/k$.

For notation, we will say that the message M_i consists of the c digits $\mu_{i,j}$ for $0 \leq j < c$. In addition, message M_i has the d checksum digits $\nu_{i,j}$ for $0 \leq j < d$.

Then, a message M_3 can be forged if all the following are true:

1. $\mu_{3,j} \leq \min(\mu_{1,j}, \mu_{2,j})$ for $0 \leq j < c$
2. $\nu_{3,j} \leq \min(\nu_{1,j}, \nu_{2,j})$ for $0 \leq j < d$

It turns out the number of messages k that meet these criteria can be evaluated by this simple procedure.

We first note that the value of the checksum depends only on the sum of the message digits μ, and is otherwise independent of what those values are.

Hence, we can compute the number of messages that conform to the forgery requirements on the message digits and has digits that sum to a particular value. That is, we define the value $\alpha_{i,j}$ to be the number of ways i digits, all of which meet criteria 1, sum to the value j.

Specifically:

$$\begin{aligned} \alpha_{0,0} &= 1 \\ \alpha_{0,j} &= 0 \text{ for } j \neq 0 \\ \alpha_{i,j} &= \sum_{k=\min(\mu_{1,j},\mu_{2,j})}^{w-1} \alpha_{i-1,j-k} \text{ for } i > 0 \end{aligned}$$

This can be efficiently evaluated for the values of c, w in question.

Then, we sum each nonzero $\alpha c, j$ where j corresponds to checksum dugits that meet criteria 2. That sum is the value k.

This gives us an efficient way to count the number of forgable preimages k, and hence allows us to compute the precise security level.

B Using a one-time signature forgery in SLH-DSA, LMS, XMSS

This appendix shows how to generate forgeries to SLH-DSA, LMS, XMSS, assuming that you have two valid signatures to random messages that were signed by the same one-time signature key, and one of those messages[9]. Hence, the security of those systems requires you to avoid reusing those private keys.

For the bottom-most LMS, XMSS WOTS signatures (the ones that directly sign the message), the approach is obvious: the attack selects his message, and then tries various message randomizers (the value C for LMS (see section 4.5 of [MCF19] or the value r for XMSS (see section 4.1.9 of [HBG$^+$18]), and compute the initial message hash (including the values from the public key and the index of the reused leaf). Then, if the resulting hash can be forged using the two signatures, we can construct a forgery of the attacker selected message.

For the non-bottom-most cases of SLH-DSA, LMS, XMSS, that one-time signature signs the next lower Merkle tree. So, what we do is generate our own Merkle tree where we know the leaf one-time signatures and authentication path for half the leaf nodes. That is, we generate the entire left-side of that Merkle tree (so that we know the first half of those one-time private keys). Then, we select an arbitrary value for the right-side subtree just below the root, and hash that with the root of our left-side subtree to come up with that root value of our artificial tree. If that root value can be signed using our two signatures, then we can use that to generate a forgery; the authentication path can be constructed by using the internal nodes on the left-side that we constructed, and the arbitrary value we selected as the last element of the authentication path.

For LMS and XMSS, using this is straight-forward: we can select a leaf in the hypertree that resides within the left hand side of the Merkle tree we generated, generate all the parts of the hypertree we need that lies below that Merkle tree, and generate a valid looking authentication path (and WOTS signatures as needed). For SLH-DSA, the process is similar, except that we cannot arbitrarily select the hypertree leaf. Instead, we need to try various R randomizers until it points us to a hyperleaf covered by the left side of the Merkle tree we generated, and then generating the signature is straight-forward.

The bottom-most case of SLH-DSA is similar; instead of signing a Merkle tree root, the one-time signature in question sign a FORS tree. So, we construct our own FORS trees; that is, the first $k-1$ FORS trees entirely, and the left hand side of the last one. Then, we select various values for the right side node and compute the corresponding FORS root until we get one that can be used as a forgery. Then, we take the message we want to forge and use various R randomizers until it points us that that hypertree leaf and the left side of the final FORS tree; once we have that, we can forge.

[9] In the SLH-DSA fault attack case, you won't neccessarily have both messages (depending on where the fault occurred). However, because the other message can be reconstructed, that does not prevent an attack. If we don't have both messages, the one we don't have can be reconstructed from the signatures and the message we do have. If a digit from the unknown message is greater than or equal to the corresponding digit from the known message, we can deduce the difference by advancing the hash from the known message signature. If a digit from the unknown message is less than then corresponding digit from the known message, we can deduce that difference by advancing the hash from the unknown message signature.