A Kelsey–Kohno Attack Variant against SHA-2 Message Digests in HSS/LMS and XMSS

Falko Strenzke*
fstrenzke@cryptosource.de

October 25, 2026

Abstract

We present a multi-target second-preimage attack on the message hashing of the stateful hash-based signature schemes HSS/LMS and XMSS with SHA-256 and with the truncated SHA-256/192. The attack builds on the herding technique of Kelsey and Kohno: the attacker takes the chaining states that follow the per-signature prefixes of many collected signatures and merges them, level by level through state collisions, into a single chaining state. From that state one search serves all collected signatures at once, which defeats the commitment to the key and to the one-time-signature leaf index at the start of the hash input that is meant to rule out multi-target attacks. The search can match the final digests, which is the better choice against SHA-256/192, or, after appending a Kelsey–Schneier expandable message, the internal chaining states of long signed messages, which is effective for full SHA-256. For SHA-256 with NIST's bound of 2^{64} signatures, which may be spread over any number of keys, the cost in compression-function calls falls from the nominal 2^{256} to about 2^{166} for messages of up to 4 GiB and to about 2^{156} in the formal limit of maximal message length, a loss of 100 bits. For SHA-256/192, digest matching costs about 2^{153} for 2^{64} signatures of any length, a loss of 39 bits. Because the attack pools signatures across keys, even the first signature of an individual key contributes to a potential overall exposure of an ecosystem. The attacks exploit the chaining state of SHA-256, which is narrower than twice the targeted security level; the SHAKE256 parameter sets are not affected.

1 Introduction

NIST SP 800-208 [4] approves the stateful hash-based signature schemes LMS/HSS [19] and XMSS/XMSS$^{\mathrm{MT}}$ [13] with four hash functions: SHA-256, SHA-256/192, SHAKE256/256 and SHAKE256/192. The parameter set names carry the output length of the hash function, and the security level associated with a parameter set is conventionally taken to be that output length: RFC 8391 states that "parameters with $n = 32$ provide a classical security level of 256 bits" [13, Section 5], n being the output length in bytes there. RFC 9858, which defines the SHA-256/192 and SHAKE256 parameter sets of LMS, states the convention explicitly: assuming a random oracle, "the security of the system is at least $8N - 1$ bits" for an output of N bytes, i.e. 255 bits for SHAKE256/256 and 191 bits for SHA-256/192 and SHAKE256/192, "equivalent to the strength of AES-256" resp. AES-192, and it sees "no security ... difference between the SHA-256-based parameter sets ... versus the SHAKE-based parameter sets" [10, Sections 6 and 8]. The XMSS 192-bit sets exist only in SP 800-208.

*This paper was written with substantial conceptual and textual contributions from Anthropic's Claude Fable 5.1 model. The author thanks Anthropic for the grant of API credits under its Claude for Open Source program which supported this work.

These figures rest on the length of the digest, not on an analysis of the hash function's structure. For SPHINCS$^+$, the same convention was shown to be unjustified: Antonov [2] observed that the SHA-256 instantiation lacks a multi-target property that its security proof requires, Perlner, Kelsey and Cooper [24] turned this observation into a forgery for the category-5 parameter sets with about 2^{217} instead of 2^{256} work, and the scheme was changed to compute its message hash, tree hash and public-key compression with SHA-512 before it became FIPS 205 [11, 23]. LMS and XMSS received no such change. The German BSI's Technical Guideline TR-02102-1 does anticipate the transfer: it notes that "theoretically, the idea of this attack can be transferred to LMS/HSS in combination with SHA-256, so that this attack should be taken into account when using such LMS/HSS instances", while continuing to recommend all parameter sets of SP 800-208 on the grounds that their security remains well above its 120-bit target [8, Remark 5.11].

This work chooses a different attack target. It asks what the same family of techniques, the herding attack of Kelsey and Kohno [15] and the long-message second-preimage attack of Kelsey and Schneier [16], achieves against the *message hash* of LMS and XMSS with SHA-256 and SHA-256/192. The result is a degradation of every SHA-2 parameter set. We present three attacks, the second being Kelsey and Schneier's attack applied unchanged to one signature. The *hourglass attack with digest matching* (Section 2) merges the prefix states of 2^{k_1} collected signatures into one state h^* at a cost of about $2^{132+k_1/2}$ compression calls and then tries final blocks against all 2^{k_1} digests at once; against SHA-256/192 it costs about 2^{153} for NIST's 2^{64} signatures, collected under any number of keys, and about 2^{167} for an attacker restricted to a single LMS tree of height 25, against a nominal 2^{192}. The *expandable-message attack* (Section 3.1) is Kelsey and Schneier's attack applied to one signed message under SHA-256: a message of 64 MiB lowers the second-preimage cost of its digest from 2^{256} to 2^{236}, a message of maximal length to 2^{201}. The *hourglass attack with state matching* (Section 3.2) combines the two and targets the internal states of all merged messages, which is effective for SHA-256 because its digest is of the chaining state's size. It costs about 2^{166} for 2^{64} signatures on messages of up to 4 GiB and about 2^{156} for messages of maximal length, against a nominal 2^{256}, and about 2^{176} for a single tree of height 25 with maximal-length messages. None of the attacks is practical, but all fall short of the 256-bit and 192-bit levels that the digest lengths suggest. The SHAKE256 parameter sets are not affected: the attacks need a chaining state narrower than twice the output, which SHA-256 and SHA-256/192 have and SHAKE256 and truncated SHA-512 do not.

The rest of this section recalls the parts of the schemes that matter for the attacks. Section 2 presents the attack against SHA-256/192, where the attacker matches final digests. Section 3 treats SHA-256, where matching internal states of long messages is cheaper. Both sections treat LMS first and XMSS directly afterwards. Section 4 discusses the results.

1.1 LMS and HSS

LMS combines a one-time signature scheme (LM-OTS, a Winternitz construction) with a Merkle tree. Each leaf of the tree is the hash of one LM-OTS public key; the root of the tree is the LMS public key. A signature on a message consists of the index q of the leaf that is used, the LM-OTS signature under that leaf's key, and the authentication path from the leaf to the root. Every leaf, i.e. every value of q, may be used for one signature only, which is why the scheme is stateful. A tree of height $h \in \{5, 10, 15, 20, 25\}$ provides 2^h signatures. The technical details of LM-OTS and of the tree hashing are irrelevant for what follows: all attacks below reuse an honestly generated LM-OTS signature and authentication path unchanged, and only the message hash is attacked.

HSS stacks up to eight levels of LMS trees. The leaves of the top tree sign the public keys of second-level trees, whose leaves sign the public keys of the next level, and so on; the

leaves of the bottom trees sign messages (Figure 1). An HSS instance with levels of heights $h_0, \ldots, h_{L-1}$ can issue $2^{h_0+\cdots+h_{L-1}}$ signatures, far more than the 2^{64} signatures that NIST's evaluation criteria grant an attacker [22, Section 4.A.4]. We therefore use $T = 2^{64}$ as the reference number of signatures available to the attacker. T counts signatures under all keys the attacker targets: the attacks below produce a forgery under the key of whichever collected signature is hit, so an attacker who is content with a forgery under any of many keys can collect 2^{64} signatures without any single signer issuing that many.

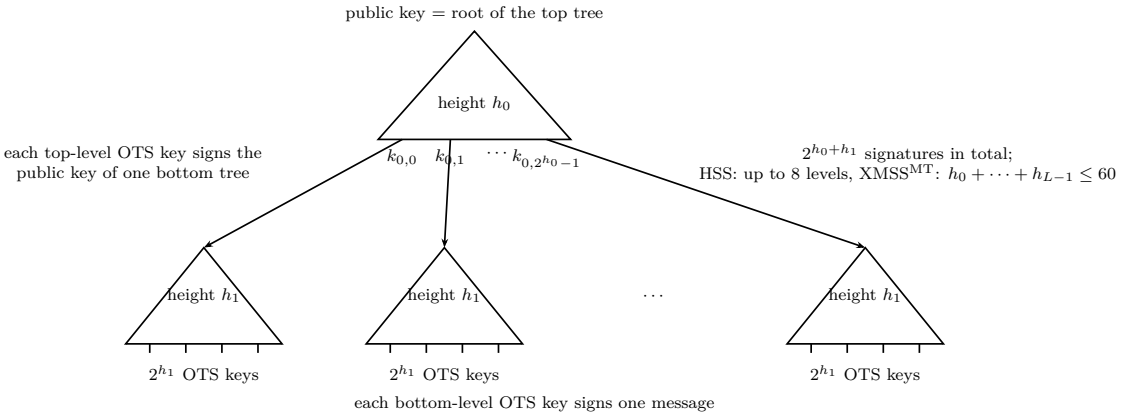


Figure 1: A two-level hierarchy of trees as used by HSS and XMSS$^{\mathrm{MT}}$. The top-level OTS keys sign the public keys of the lower trees; the lower OTS keys sign messages. The number of signatures is the product of the tree sizes.

1.2 The LMS message digest

Throughout, n denotes the output length of the hash function in bits, $n \in \{256, 192\}$. To sign a message M with the LM-OTS key of leaf q, LMS draws an $n/8$-byte randomizer C and computes [19, Section 4.5]

$$Q = \mathsf{H}(I \,\|\, \mathrm{u32str}(q) \,\|\, \mathrm{u16str}(D_{\mathrm{MESG}}) \,\|\, C \,\|\, M), \tag{1}$$

where I is the 16-byte identifier of the key pair and $D_{\mathrm{MESG}} = \texttt{0x8181}$ is a domain separator. The LM-OTS signature then signs Q together with its Winternitz checksum, and C is transmitted as part of the signature. The verifier reads q and C from the signature, recomputes Q from the message it is given, finishes the Winternitz chains, hashes the resulting LM-OTS public key into the leaf and walks the authentication path to the root.

Figure 2 shows the layout of the hash input. The prefix $I \,\|\, q \,\|\, D_{\mathrm{MESG}} \,\|\, C$ occupies $p = 22 + n/8$ bytes, i.e. 54 bytes for $n = 256$ and 46 bytes for $n = 192$, so it lies entirely within the first 64-byte block of SHA-256. The digest of every signature is therefore computed from a chaining state that depends on the leaf index q from the first block on. This is deliberate: RFC 8554 points out that every hash input of the scheme varies within its first 23 bytes, up to repetitions of I, and that this property is needed for its proof of security in the random oracle model [19, Section 9.1]. A forger who wants to reuse the signature of leaf q_i seems bound to produce a second preimage under exactly that prefix, so that holding many signatures would give no advantage. Sections 2 and 3 show that the second half of this sentence is false for Merkle–Damgård hash functions whose chaining state is narrower than twice the targeted security level.

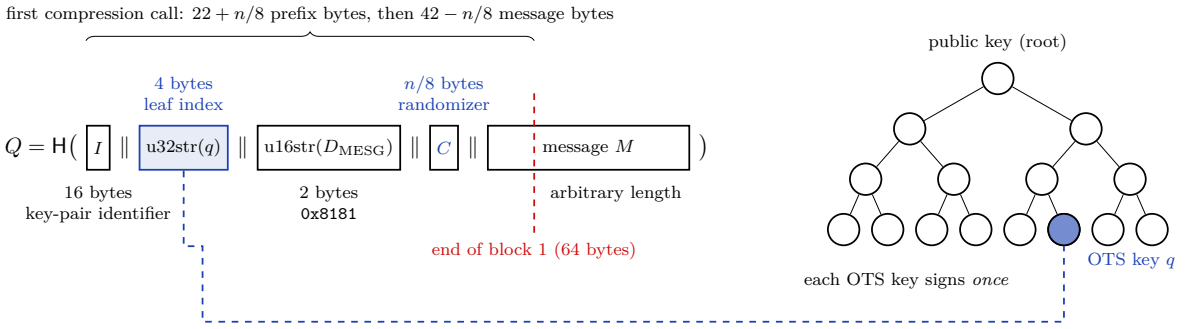


Figure 2: The LMS message digest (1) and the Merkle tree leaf that q selects. The prefix fills the first $22 + n/8$ bytes of block 1, so the chaining state after block 1 already depends on q and C. This is intended to commit a forger to the key and the leaf index and so to prevent straightforward multi-target preimage search. Both values are read from the signature by the verifier.

1.3 XMSS and XMSS$^{\mathrm{MT}}$

XMSS [13] has the same two-layer structure as LMS: a Winternitz one-time signature (WOTS^{+}) and a Merkle tree of height $h \in \{10, 16, 20\}$; XMSS$^{\mathrm{MT}}$ stacks up to twelve levels of trees with a total height of 20, 40 or 60. The only difference to LMS that deserves our attention is the input to the message hash. For $n = 256$ the message hash is computed as [13, Section 5.1]

$$\mathsf{H}_{\mathrm{msg}} = \text{SHA-256}(\text{toByte}(2, 32) \,\|\, r \,\|\, \text{root} \,\|\, \text{toByte}(\text{idx}, 32) \,\|\, M), \tag{2}$$

where r is an $n/8$-byte randomizer that the signer derives pseudorandomly and transmits in the signature, root is the public key and idx the index of the OTS key. The prefix has exactly $p = 128$ bytes, two full SHA-256 blocks, so the message starts at a block boundary. For the SHA-256/192 parameter sets SP 800-208 shortens the domain separator to four bytes [4, Section 5.2]: $\mathsf{H}_{\mathrm{msg}} = T_{192}(\text{SHA-256}(\text{toByte}(2, 4) \,\|\, r \,\|\, \text{root} \,\|\, \text{toByte}(\text{idx}, 24) \,\|\, M))$ with a prefix of $p = 76$ bytes, which leaves 52 free bytes in block 2. As in LMS, the verifier takes r and idx from the signature without any check; RFC 8391 derives r from a secret key on the signer's side, but nothing on the verifier's side depends on how r was generated. So the prefix fixes the OTS key in the first blocks, exactly as in LMS, and everything said about LMS below applies to XMSS with $I \,\|\, q \,\|\, D_{\mathrm{MESG}} \,\|\, C$ replaced by $\text{toByte}(2) \,\|\, r \,\|\, \text{root} \,\|\, \text{idx}$, the only difference being that the prefix ends in block 2 instead of block 1, which shifts every block count by one and leaves every cost unchanged.

RFC 8391 makes the intended effect of this prefix explicit: the indexed randomized hashing "makes the hash function calls position- and user-dependent", and "if one assumes that the used hash function is a random function, it can be shown that a multi-user existential forgery attack that targets this message compression has a complexity of 2^n hash function calls" [13, Section 9.1]. The following sections show that this 2^n bound does not carry over to the SHA-2 instantiations, whose hash is not a random function (Section 1.5).

1.4 SHA-256 and SHA-256/192

SHA-256 [21] is a Merkle–Damgård construction [6, 20] with a 256-bit chaining state and 512-bit message blocks (Figure 3). The input is padded with a single `0x80` byte, zero bytes and the 64-bit input length; the padded input is processed block by block by the compression function f, starting from a fixed initial value IV, and the digest is the final chaining state without any further transformation. Because the chaining state and the output have the same size, $m = n = 256$, SHA-256 is a *narrow-pipe* hash in the terminology of Lucks [18]. The length field limits the input to $2^{64} - 1$ bits, i.e. to about 2^{55} blocks or 2^{61} bytes.

SP 800-208 defines SHA-256/192 as $T_{192}(\text{SHA-256}(M))$, the leftmost 192 bits of the SHA-256 digest [4, Section 2.3]. The initial value is unchanged, and so is the whole computation up to the final state; only the output is shortened. Its chaining state is thus wider than its output, $m = 256 > n = 192$, but not twice as wide, $m < 2n$. We call this an *intermediate pipe*: it is not narrow, but it does not have the property of a wide-pipe hash ($m \geq 2n$) that generic attacks on the chaining state cost at least as much as generic attacks on the output.

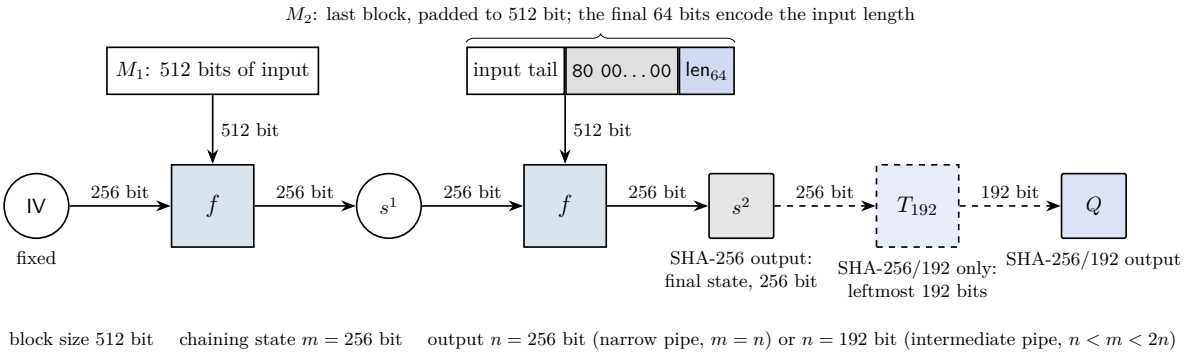


Figure 3: SHA-256 as a narrow-pipe Merkle–Damgård construction, shown for a two-block input, and SHA-256/192 as its truncation. Both share the 256-bit chaining state; only the output differs.

1.5 Related work, adversary model and cost conventions

Generic attacks on Merkle–Damgård hashes. Kelsey and Schneier [16], building on Dean [7], find a second preimage of a 2^k-block message under an n-bit narrow-pipe hash with about $k \cdot 2^{n/2+1} + 2^{n-k+1}$ compression calls: an *expandable message*, a set of messages of every length in a range that all reach the same chaining state, is linked into one of the 2^k internal states of the target. Kelsey and Kohno [15] introduced the *diamond structure*, a complete binary tree of collisions that merges 2^k chaining states into one, originally estimated to cost about $2^{n/2+k/2+2}$, and the resulting *herding attack*: commit to a hash value, later "herd" any prefix into it with 2^{n-k} work. Blackburn, Stinson and Upadhyay [3] corrected the analysis of the original diamond construction by a factor of about $\sqrt{k}$. Kortelainen and Kortelainen [17] subsequently gave an adaptive construction with $O(2^{(n+k)/2})$ expected compression calls; we use this construction with a conservative constant in the estimates below. Andreeva et al. [1] used the diamond in place of the expandable message for long-message second preimages and observed that the linking step may target the internal states of many messages at once, so that its cost falls linearly with the number of target messages. In this paper we call our structure an *hourglass*, because, as the figures show, the collision tree that narrows down to h^* is followed by a search that fans out again from h^*, i.e. resembling an hourglass lying on its side. The narrowing half is Kelsey and Kohno's diamond built from precomputed prefix states.

Hash-based signatures. Katz [14] and Fluhrer [9] analyse LMS in the random oracle model, Fluhrer with the compression function as the oracle; RFC 8554 states the resulting bound as a forgery probability below 2^{-129} for an attacker limited to 2^{120} compression calls and 2^{64} keys [19, Section 9], and does not claim 256 bits. For the SHA-256/192 and SHAKE256 parameter sets, RFC 9858 applies the same random-oracle analysis of Katz, which yields the $8N - 1$ bound quoted in the introduction, and adds that classically "there is no known attack better than performing hashes of a large number of distinct preimages" [10, Section 8]. For the XMSS message hash, the security analysis requires multi-target extended target-collision resistance [12], while RFC 8391 justifies its quantitative security estimate by assuming a random

function [13, Section 9.1]. The attacks below show that the concrete SHA-256 message hash does not provide the required security at the claimed level; they exploit the iterative structure of SHA-256, which is not indifferentiable from a random oracle [5], and for which random-oracle bounds become unreliable beyond about 2^{128} queries, the compression function itself being a Davies–Meyer construction with easily found fixed points [24, footnote 11]. Fluhrer [9, Section 1] had already used internal-state matching against the pre-07 LMS draft, whose tree hashing placed the node index after the child values, so that the SHA-256 state after the children could be matched against all nodes of the tree at once; the draft was revised in response. For SPHINCS$^+$, Antonov [2] showed that the WOTS$^+$ public-key hash under SHA-256 lacks the distinct-function multi-target second-preimage resistance that the proof requires, Perlner, Kelsey and Cooper [24] turned this into a forgery at about 2^{217}, and a weakness of the message hash at category 5 had been reported on the NIST forum before [24, 25]. The tweak adopted for FIPS 205 uses SHA-512 for the message hash, the tree hash, the public-key compression and $\mathsf{PRF}_{\mathrm{msg}}$ at categories 3 and 5, i.e. for every function whose input extends beyond one block after the seed block; F and PRF keep SHA-256 on fixed two-block inputs [11, 23]. The attack of [24] targets a fixed-length input, so it herds the states that precede the last few blocks and links with a single block at one fixed position; it has no use for expandable messages. The message hash studied here is of variable length, which is what lets the *hourglass attack with state matching* (Section 3.2) add Kelsey and Schneier's expandable message after h^* and target the internal states of long messages at any depth.

Adversary model. The attacker holds T valid signatures, with the signed messages, under any set of public keys for which a forgery is useful to the attacker, and wants a valid signature on a message that was not signed, under one of these keys; which key that is, the attacker does not choose. Following NIST [22, Section 4.A.4] we allow $T \leq 2^{64}$ signatures in total; NIST's criteria also ask for resistance to multi-key attacks [22, Section 4.A.6], and RFC 8554's own bound is stated for up to 2^{64} keys [19, Section 9]. RFC 9858 mentions multi-key attacks as a possibility for which "no such method of attack has been proposed" and bounds their loss by K bits for 2^K keys [10, Section 8]. The hourglass attack is such a method: it merges signatures regardless of the keys they belong to. Its loss, however, is a function of the number of collected signatures, not of the number of keys: 2^{64} signatures under a single key already cost the full 39 bits, and spreading them over many keys adds nothing. RFC 9858's bound on the loss attributable to the key count is therefore formally respected and says nothing about the attack. For an attacker restricted to a single key, $T \leq 2^h$ for a single tree and $T \leq 2^{60}$ for XMSS$^{\mathrm{MT}}$. Costs are classical and counted in compression function calls of SHA-256, and security levels and losses are stated against the nominal 2^n hash computations, as RFC 9858 does; Table 5 additionally converts to bit operations at 2^{18} per compression, following [24, Section 5.1]. Memory, parallelism and constant factors are discussed in remarks. The forged signature reuses the OTS signature and authentication path of one collected signature and changes only the randomizer and the message. The attacks require that the application accepts the forged message, which consists mostly of attacker-chosen blocks; as usual for generic attacks we do not model the format of the message.

Notation. f is the compression function, $m = 256$ the state size in bits, $n \in \{256, 192\}$ the output size in bits, and p the length of the prefix in bytes ($22 + n/8$ for LMS, 128 or 76 for XMSS). For a signed message M_i we write ℓ_i for the number of blocks of its padded hash input and s_i^j for the chaining state after block j. P_i is the state after the block that completes the prefix, i.e. s_i^1 for LMS and s_i^2 for XMSS; block counts below are given for LMS. σ_i denotes the OTS signature value together with the authentication path, and I_i (resp. root_i) the key identifier (resp. public key) of the key that produced signature i. k_1 denotes the logarithm of the number of signatures the attacker merges, k_2 the logarithm of the number of eligible

internal states per signed message.

2 SHA-256/192: Hourglass Attack with Digest Matching

The attack is stated for LMS with SHA-256/192; the XMSS variant follows immediately after, and for SHA-256 the same steps apply with 192 replaced by 256 in the linking cost.

Observation 1 (The randomizer is not bound)**.** The verifier takes C (resp. r) from the signature and uses it only inside the message hash. A forged signature may therefore carry any value C' together with the OTS signature and authentication path of a collected signature, and it verifies if and only if the forged message hashes under $I_i \,\|\, q_i \,\|\, D_{\mathrm{MESG}} \,\|\, C'$ to the digest Q_i that the OTS signature signs.

The randomizer does prevent the obvious collision attack, in which the attacker prepares two messages with the same digest and asks for a signature on one of them, because the signer chooses C after the message is fixed. It does not add to second-preimage resistance: once σ_i is public, the attacker is free to choose C', and even reusing C_i costs nothing, since a fixed known prefix is merely a different initial value for the attacker.

Attack 1 (*Hourglass attack with digest matching*)**.** The attacker holds $T = 2^{k_1}$ signatures (q_i, C_i, σ_i) on messages M_i under the targeted keys and stores the records $Q_i \mapsto (I_i, q_i, \sigma_i)$ in a table, so that a hit identifies key and leaf.

① *Prefix states.* For every i choose a randomizer C'_i and $42 - n/8$ bytes m_0 and compute the chaining state $P'_i = f(\mathsf{IV}, I_i \,\|\, q_i \,\|\, D_{\mathrm{MESG}} \,\|\, C'_i \,\|\, m_0)$.

② *Narrowing half.* Construct a diamond with the 2^{k_1} chosen prefix states as its leaves using the algorithm of Kortelainen and Kortelainen [17, Section 2]. At each level, their adaptive collision search pairs all current states, retaining and extending candidate sets for states that remain unpaired. Repeat until a single state h^* remains. Record the message block on every edge. Every leaf-to-root path contains exactly k_1 blocks.

③ *Widening half.* For $t = 1, 2, \ldots$, choose a 55-byte final message fragment $B(t)$. Let pad be the SHA-256 padding for the complete hash input, including the prefix and all message bytes. Thus, $B(t) \,\|\, \mathsf{pad}$ forms one 512-bit block, and the padded hash input contains $k_1 + 2$ blocks. Test whether $T_{192}(f(h^*, B(t) \,\|\, \mathsf{pad}))$ matches a Q_i in the table.

④ *Output.* On a hit with Q_i, output the message $M' = m_0 \,\|\, X_{i,1} \,\|\, \ldots \,\|\, X_{i,k_1} \,\|\, B(t)$, where $X_{i,1}, \ldots, X_{i,k_1}$ are the blocks on the path from P'_i to h^*, and the signature (q_i, C'_i, σ_i).

The verifier computes $T_{192}(\text{SHA-256}(I_i \,\|\, q_i \,\|\, D_{\mathrm{MESG}} \,\|\, C'_i \,\|\, M'))$. Block 1 leads to P'_i, the path blocks lead to h^*, the final block to Q_i, so the LM-OTS signature and the authentication path of leaf q_i verify unchanged. Figure 4 shows the two halves of the hourglass below the signer's or verifier's chains.

XMSS. For XMSS with SHA-256 the prefix $\mathrm{toByte}(2, 32) \,\|\, r'_i \,\|\, \mathrm{root}_i \,\|\, \mathrm{toByte}(\mathrm{idx}_i, 32)$ fills blocks 1 and 2 completely, so P'_i is the state after block 2, m_0 is empty and the randomizer r'_i is the attacker's 32 free bytes of the prefix. For SHA-256/192 the 76-byte prefix leaves m_0 with 52 bytes in block 2. In both cases the forged hash input has $k_1 + 3$ blocks, the forged signature carries r'_i in place of r_i, and the verifier recomputes $\mathsf{H}_{\mathrm{msg}}$ under idx_i and reaches Q_i exactly as above. Nothing else changes.

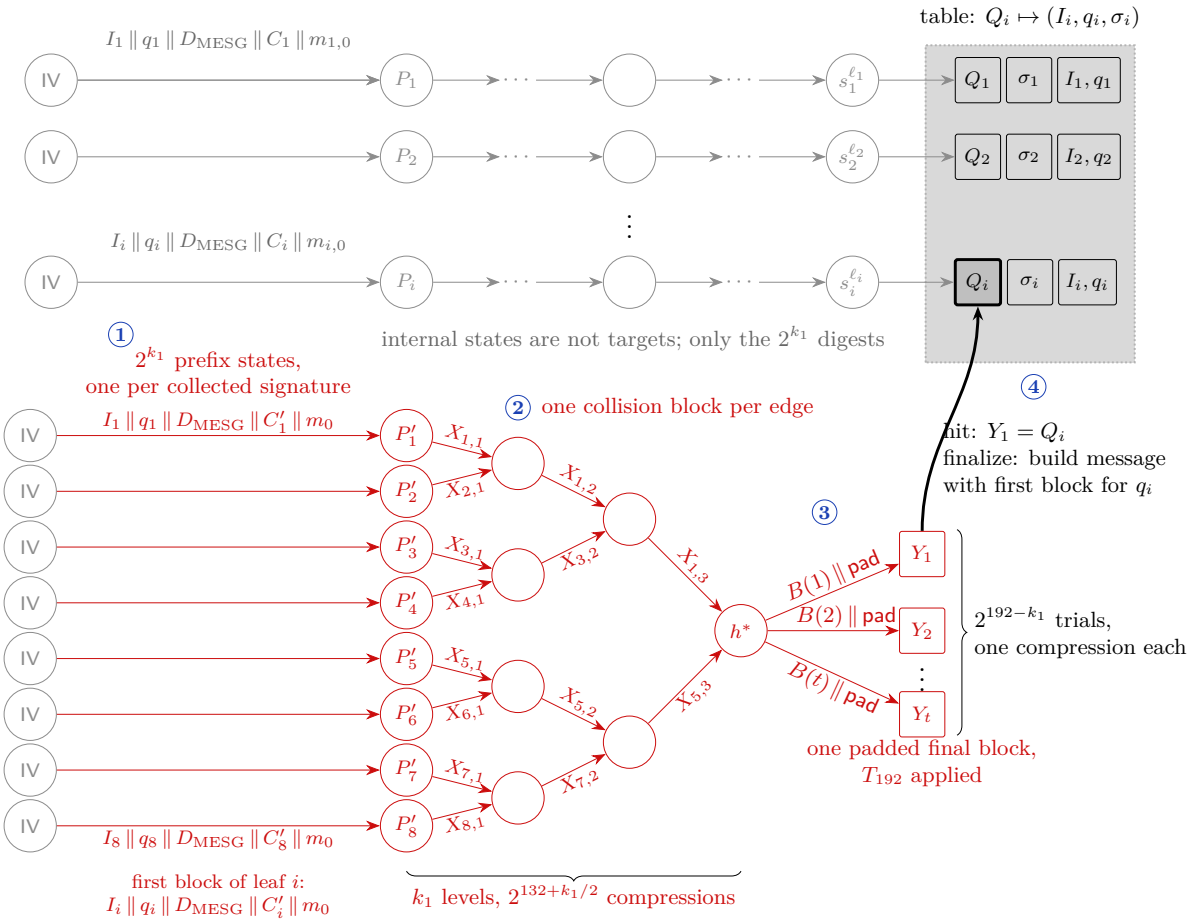


Figure 4: The *hourglass attack with digest matching*. Top: the signer's or verifier's hash chains; only the final digests are targets. Bottom: the attacker's hourglass. The narrowing half merges the 2^{k_1} prefix states, with $k_1 = 3$ depicted in the figure, each carrying a different leaf index q_i, into h^*; the widening half tries final blocks from h^* against all 2^{k_1} digests. Whichever digest Q_i is hit, the path from P'_i to h^* supplies a message that hashes to it under the prefix of leaf q_i. IV is the initial state of the hash function; the labelled first edges show the first block, which carries the LMS prefix together with the first $42 - n/8$ message bytes $m_{i,0}$ (for XMSS the prefix fills two blocks, Section 1.3). The attacker chooses C'_i and m_0 freely. The circled numbers mark the steps of the attack statement. $X_{i,y}$ is the y-th block on the path from P'_i to h^*; paths that have merged share their later blocks, e.g. $X_{1,2} = X_{2,2}$.

Why the prefix does not help. Without the narrowing half, a trial from the prefix state of leaf q_i can only be compared with Q_i, since a hit with Q_j, $j \neq i$, is useless: the forged message would have to be hashed under the prefix of leaf q_j. This is the multi-target protection that the distinct prefixes are designed to give, and for a random function it is sound. The hourglass circumvents it: after the narrowing half, one state h^* is reachable from *every* prefix, so one trial from h^* is simultaneously a trial under all 2^{k_1} prefixes, and the full multi-target gain returns in the widening half. The price is the narrowing half, whose cost grows with k_1. The narrowing half is equally indifferent to the key identifier I_i (resp. root_i) in the prefix: the merged signatures may belong to different keys, and the forgery lands under the key of the signature that is hit.

Cost. The analysis of Kortelainen and Kortelainen [17, Section 2.4] gives an upper cost estimate for Step 2 of approximately $8 \cdot 1.582 \cdot 2^{(m+k_1)/2} \approx 12.66 \cdot 2^{(m+k_1)/2}$ expected compression calls. The constant prefactor is less than 16; for numerical estimates we use the conservative value $16 \cdot 2^{(m+k_1)/2} = 2^{132+k_1/2}$ for $m = 256$.[1] We consider this a sufficiently conservative attack-cost estimate under the generic-hash assumptions of the analysis, not a claim of optimality. Step 3 costs 2^{n-k_1}, i.e. 2^{192-k_1} for SHA-256/192 and 2^{256-k_1} for SHA-256; Steps 1 and 4 are negligible.

$$W_{\text{digest}}(k_1) \approx 2^{132+k_1/2} + 2^{n-k_1}. \tag{3}$$

The attacker chooses $k_1 \leq \log_2 T$ to minimise the estimate (3). For $n = 192$ the two terms balance at $k_1 = 40$ and the minimum over integers lies at $k_1 = 41$, where the total is about $2^{152.9}$; collecting more than 2^{41} signatures does not reduce this estimate further, because the narrowing half then dominates. For $n = 256$ the balance point $k_1 \approx 82.7$ lies beyond the 2^{64} signatures the model allows, so the attacker uses all signatures and the linking step dominates: 2^{192} at $T = 2^{64}$. Table 1 lists the cost for the parameter sets of SP 800-208.

			Cost [compression-function calls]			
Scheme	T	k_1	Narrowing $2^{132+k_1/2}$	Final block 2^{n-k_1}	Total	Nominal
LMS, $h = 5$	2^5	5	$2^{134.5}$	2^{187}	2^{187}	2^{192}
LMS, $h = 10$	2^{10}	10	2^{137}	2^{182}	2^{182}	2^{192}
LMS, $h = 15$	2^{15}	15	$2^{139.5}$	2^{177}	2^{177}	2^{192}
LMS, $h = 20$	2^{20}	20	2^{142}	2^{172}	2^{172}	2^{192}
LMS, $h = 25$	2^{25}	25	$2^{144.5}$	2^{167}	2^{167}	2^{192}
XMSS, $h = 10$	2^{10}	10	2^{137}	2^{182}	2^{182}	2^{192}
XMSS, $h = 16$	2^{16}	16	2^{140}	2^{176}	2^{176}	2^{192}
XMSS, $h = 20$	2^{20}	20	2^{142}	2^{172}	2^{172}	2^{192}
XMSS$^{\text{MT}}$, $h = 40$	2^{40}	40	2^{152}	2^{152}	2^{153}	2^{192}
XMSS$^{\text{MT}}$, $h = 60$	2^{60}	41	$2^{152.5}$	2^{151}	$2^{152.9}$	2^{192}
HSS, or many keys (NIST bound)	2^{64}	41	$2^{152.5}$	2^{151}	$2^{152.9}$	2^{192}
XMSS$^{\text{MT}}$, $h = 60$, SHA-256	2^{60}	60	2^{162}	2^{196}	2^{196}	2^{256}
HSS or many keys, SHA-256	2^{64}	64	2^{164}	2^{192}	2^{192}	2^{256}

Table 1: The *hourglass attack with digest matching* against the SHA-256/192 parameter sets of SP 800-208 ($n = 192$) and, in the last two rows, against SHA-256 ($n = 256$). k_1 is the logarithm of the number of merged signatures chosen by the attacker, at most $\log_2 T$. Rows with $T = 2^h$ or 2^{60} assume an attacker restricted to one key; with signatures from many keys every parameter set reaches the $T = 2^{64}$ rows.

The loss against the nominal 192 bits is thus about 39 bits for HSS and for XMSS$^{\text{MT}}$ with

[1] This is an upper estimate of the attacker's cost. The best known construction, by Weizmann, Dunkelman and Haber [27], needs about $1.254 \cdot 2^{(m+k_1)/2+2} \approx 2^{130.3+k_1/2}$, about 1.7 bits less; using it would lower the SHA-256/192 figures of this paper by about one bit.

$h \geq 40$, 25 bits for a single LMS tree of height 25 and 20 bits for a single XMSS tree of height 20, remarkably close to the 40 bits that [24] found for SPHINCS$^+$ at category 5.

Remark 1 (Precision of the cost estimate). Blackburn, Stinson and Upadhyay [3] identified a missing $\sqrt{k}$ factor in the analysis of the original Kelsey–Kohno construction. The later construction of Kortelainen and Kortelainen [17] avoids this overhead. We use that construction and the conservative constant 16 throughout. The widening half needs a table of 2^{k_1} digests; the narrowing half needs, per level, a table of the candidate outputs of the surviving states, and both can be organised with parallel collision search [26] as in [24], where a memory-aware cost model is applied. The attacker could also start the first collision level inside block 1, which offers $8 \cdot 42 = 336$ free bits (C'_i and m_0); this saves one block of message length, not work. None of these corrections changes the conclusions.

Remark 2 (Message lengths). The *hourglass attack with digest matching* does not depend on the lengths of the signed messages M_i; it only uses their digests and prefixes. The padded hash input of the forged message has $k_1 + 2$ blocks (LMS), of which the first contains $42 - n/8$ free bytes, the last at most 55 free bytes, and the middle ones are collision-search output. Its length is known before the search, so the padding of the final block is fixed.

3 SHA-256: Hourglass Attack with State Matching

For the SHA-256 parameter sets the output is the final chaining state, so the attacker may match *internal* states of the signed messages instead of digests. A long signed message then offers one target per block. This is the setting of Kelsey and Schneier [16], whose expandable-message construction is spelled out in Appendix A; we first apply their attack to a single signature and then combine it with the hourglass. For SHA-256/192 the internal states are 256 bits wide while the digests are 192 bits, so matching internal states costs $2^{256-k_1-k_2}$ against 2^{192-k_1} for matching digests, and the attacks of this section never pay off there since $k_2 \leq 55$.

3.1 Kelsey–Schneier against one signature

Attack 2 (*Expandable-message attack* against one LMS signature using SHA-256 as message digest function)**.** The attacker holds one LMS signature (q_i, C_i, σ_i) on a message M_i whose padded hash input has $\ell_i \approx 2^{k_2}$ blocks.

1. Recompute the chaining states s_i^j, $1 \leq j \leq \ell_i$, of $I_i \,\|\, q_i \,\|\, D_{\mathrm{MESG}} \,\|\, C_i \,\|\, M_i$ and store those with $3 \leq j < \ell_i$ in a table ($k_2 + 2 \leq j \leq \min(\ell_i - 1,\, k_2 + 2^{k_2} + 1)$ with the generic expandable message of Step 2; the padded final block $j = \ell_i$ is excluded).[2]

2. Choose C' and m_0, compute P'_i and build from it an expandable message E_L that reaches a common state s_E for every length $L \in [k_2, k_2 + 2^{k_2} - 1]$ [16] (Appendix A); with Dean's fixed-point construction for Davies–Meyer compression functions [7, 16] (Appendix B) the range is $L \geq 1$ and the cost is about $2^{m/2+1} = 2^{129}$ (about $3 \cdot 2^{129} \approx 2^{130.6}$ with the accounting of [16, Section 4.2]).

3. Try blocks B until $f(s_E, B) = s_i^j$ for some j in the table.

4. Output $M' = m_0 \,\|\, E_{j-2} \,\|\, B \,\|\, M_i[\,64j - p :\,]$, i.e. the expandable message expanded to $j-2$ blocks followed by the suffix of M_i that occupies blocks $j+1, \ldots, \ell_i$ of the hash input, and the signature (q_i, C', σ_i).

[2] When the padding of M_i spills into a separate final block (the last data block holds 56 to 63 message bytes [21, Section 5.1.1]), the state after the penultimate block is not usable with an unconstrained bridge block: the forged message would end at a block boundary and change the length field. The attacker drops that one state, or restricts the bridge block to the original layout of message bytes followed by padding. This removes at most one target per signed message, throughout the state-matching attacks, and does not affect the estimates.

The padded hash input of M' has exactly ℓ_i blocks, so the final padded block of M_i, which contains the length, is copied unchanged and the chaining values coincide with the original ones from block j on; the digest is Q_i. The number of eligible states is $N = \ell_i - 3 \approx 2^{k_2}$ (or $\min(2^{k_2}, \ell_i - k_2 - 2)$ with the generic expandable message) and the linking step costs $2^{256}/N$. Figure 5 shows the attack. For XMSS the expandable message starts from the state after block 2 and the output is $m_0 \,\|\, E_{j-3} \,\|\, B \,\|\, M_i[\,64j - p :]$ with m_0 empty for SHA-256; the eligible states are those with $4 \le j < \ell_i$, and the cost is the same.

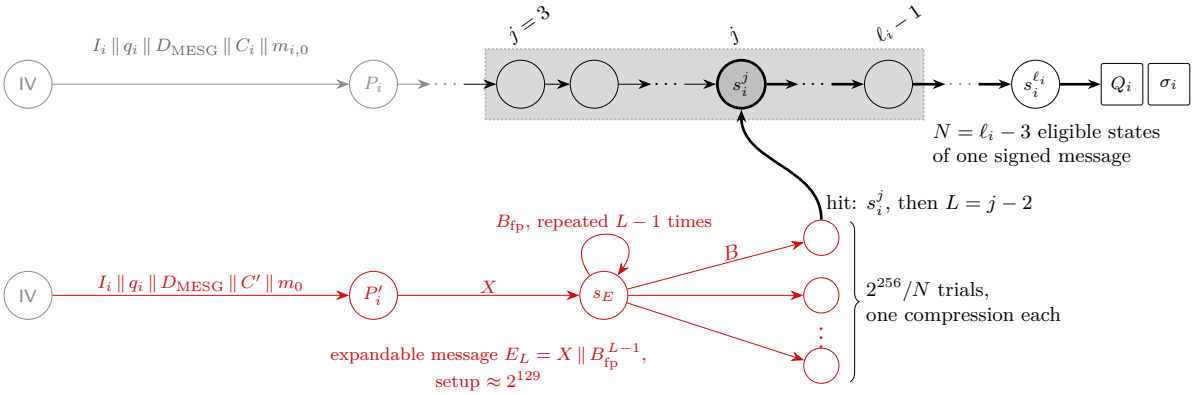


Figure 5: The *expandable-message attack* against the message hash of one LMS signature. The attacker's expandable message starts from a prefix state of leaf q_i and is linked into one of the eligible internal states; the tail of M_i is copied. The eligible band and the expandable message are drawn for the fixed-point construction (Appendix B); with the generic construction of k_2 stages the band is $k_2 + 2 \le j \le k_2 + 1 + 2^{k_2}$. Conventions as in Figure 4.

		Cost [compression-function calls]			
Message length	k_2	Setup, generic $k_2 2^{129}$	Setup, fixed points	Linking 2^{256-k_2}	Total
64 MiB (2^{26} bytes)	20	$2^{133.3}$	2^{129}	2^{236}	2^{236}
4 GiB (2^{32} bytes)	26	$2^{133.7}$	2^{129}	2^{230}	2^{230}
maximal (2^{61} bytes)	55	$2^{134.8}$	2^{129}	2^{201}	2^{201}

Table 2: The *expandable-message attack* against one LMS or XMSS signature with SHA-256, for three message lengths. The maximal length is the $2^{64}-1$ bit limit of SHA-256 [21]; RFC 8554 and RFC 8391 impose no limit of their own. The setup terms are derived in Appendix A.

Table 2 gives the cost for three message lengths. Already a single signature on a 64 MiB message lowers the second-preimage cost of its digest from 2^{256} to 2^{236}, and a message of maximal length to 2^{201}; the latter requires the signer to hash 2^{61} bytes for one signature, which is included for completeness because neither the RFCs nor NIST's model bound the message length.

3.2 The hourglass attack with state matching

The narrowing half of Section 2 and the expandable message of Section 3.1 compose: the expandable message is built from h^* instead of from one prefix state, and the linking step then targets the eligible internal states of *all* merged signed messages. This is the multi-target variant of the Kelsey–Schneier attack observed by Andreeva et al. [1, Section 6], applied to targets with distinct prefixes that the Kelsey–Kohno tree resolves.

Attack 3 (*Hourglass attack with state matching*)**.** The attacker holds $T = 2^{k_1}$ signatures on messages M_i whose padded hash inputs have about 2^{k_2} blocks each.

1. Recompute all chaining states s_i^j with $k_1 + 3 \le j < \ell_i$ and store them, keyed by the full 256-bit state, with (i, j) as payload: about $2^{k_1+k_2}$ targets.
2. Compute the prefix states P'_i and merge them into h^* as in the *hourglass attack with digest matching*.
3. Build an expandable message from h^* with the fixed-point construction, reaching s_E for every length $L \ge 1$.
4. Try blocks B until $f(s_E, B) = s_i^j$ for some entry (i, j) of the table.
5. Output the message

$$M' = m_0 \,\|\, X_{i,1} \,\|\, \ldots \,\|\, X_{i,k_1} \,\|\, E_{j-k_1-2} \,\|\, B \,\|\, M_i[\,64j - p :]$$

and the signature (q_i, C'_i, σ_i).

The block count of the padded hash input of M' is $1+k_1+(j-k_1-2)+1+(\ell_i-j) = \ell_i$, so the final padded block of M_i is reproduced and the digest is Q_i; the expandable message absorbs the difference between the depth j of the hit and the fixed length of the hourglass path. This is the role the expandable message plays in [16] and that is absent in the fixed-length setting of [24]. Figure 6 shows the attack. For XMSS, P'_i is the state after block 2, the eligible states are those with $k_1 + 4 \le j < \ell_i$, and the expandable message is expanded to $j - k_1 - 3$ blocks; the costs are unchanged.

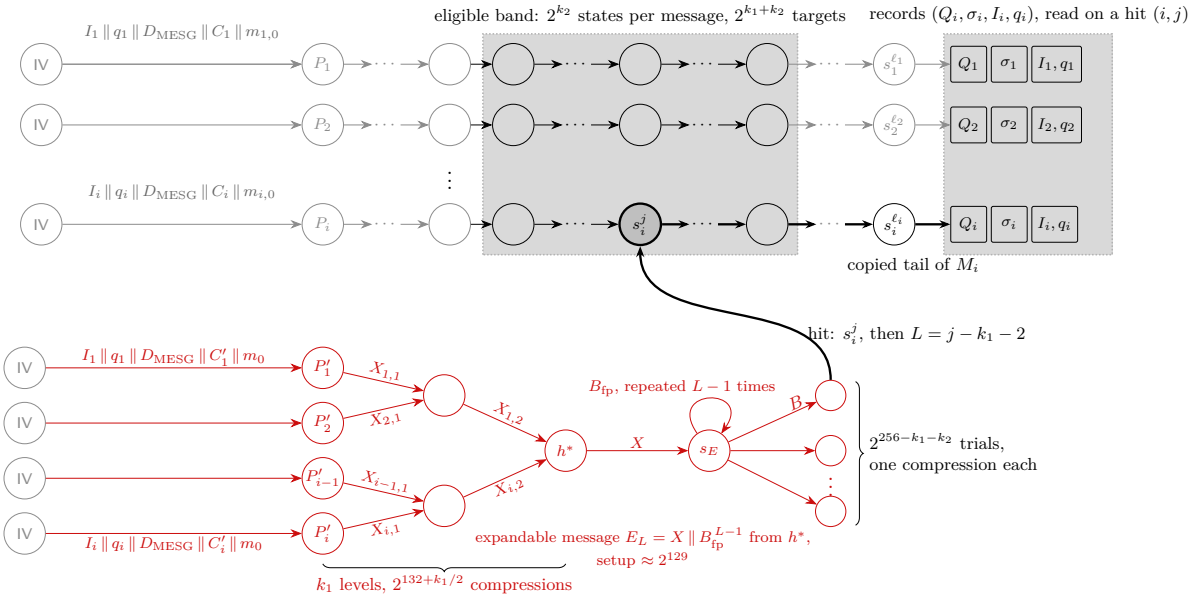


Figure 6: The *hourglass attack with state matching*. The narrowing half of the hourglass merges the attacker's prefix states of 2^{k_1} collected signatures into h^*; an expandable message from h^* makes every internal state in the eligible band of every merged message a usable target; the widening half links into one of them, and the tail of that message is copied. Conventions as in Figure 4. The expandable message is shown in its fixed-point form: one block X reaches s_E, and the fixed-point block B_{fp}, repeated $L-1$ times, keeps the state there (Appendix B).

Cost.

$$W_{\text{state}}(k_1, k_2) \approx 2^{132+k_1/2} + 2^{129} + 2^{256-k_1-k_2}, \tag{4}$$

plus $2^{k_1+k_2}$ compression calls and about $2^{k_1+k_2+5}$ bytes for the target table; for HSS with 4 GiB messages this table has 2^{90} entries, and [24, Section 5.2] argue that memory-access costs

of multi-target search against SHA-256 remain negligible for far larger target sets. For the maximal-length settings with $k_1 = 47$ in Table 4 the table has 2^{102} entries, close to the 2^{106} bound up to which [24] consider memory access negligible; those rows are formal limits in any case. The attacker chooses $k_1 \le \log_2 T$; the two large terms balance at $k_1 = \frac{2}{3}(124 - k_2)$, which for messages of up to 2^{32} bytes ($k_2 \le 26$) exceeds 64, so all available signatures are merged and the linking step dominates. Only for messages near the maximal length does the narrowing half limit the attack, with the integer optimum at $k_1 = 47$ for $k_2 = 55$. Table 3 lists moderate settings and Table 4 the extreme ones.

				Cost [compression-function calls]			
Signatures T	Messages	k_1	k_2	Narrowing	Expandable	Linking	Total
2^{20} (XMSS $h = 20$)	64 MiB	20	20	2^{142}	2^{129}	2^{216}	2^{216}
2^{20} (XMSS $h = 20$)	4 GiB	20	26	2^{142}	2^{129}	2^{210}	2^{210}
2^{25} (LMS $h = 25$)	64 MiB	25	20	$2^{144.5}$	2^{129}	2^{211}	2^{211}
2^{25} (LMS $h = 25$)	4 GiB	25	26	$2^{144.5}$	2^{129}	2^{205}	2^{205}
2^{40} (XMSS$^{\mathrm{MT}}$ $h = 40$)	64 MiB	40	20	2^{152}	2^{129}	2^{196}	2^{196}
2^{40} (XMSS$^{\mathrm{MT}}$ $h = 40$)	4 GiB	40	26	2^{152}	2^{129}	2^{190}	2^{190}
2^{60} (XMSS$^{\mathrm{MT}}$ $h = 60$)	64 MiB	60	20	2^{162}	2^{129}	2^{176}	2^{176}
2^{60} (XMSS$^{\mathrm{MT}}$ $h = 60$)	4 GiB	60	26	2^{162}	2^{129}	2^{170}	2^{170}
2^{64} (HSS or many keys)	64 MiB	64	20	2^{164}	2^{129}	2^{172}	2^{172}
2^{64} (HSS or many keys)	4 GiB	64	26	2^{164}	2^{129}	2^{166}	$2^{166.3}$
2^{64} (HSS or many keys), digest matching	any	64	–	2^{164}	–	2^{192}	2^{192}

Table 3: The *hourglass attack with state matching* against SHA-256 parameter sets with messages of ordinary length. The last row repeats the *hourglass attack with digest matching* for comparison: with SHA-256, state matching gains the factor 2^{k_2} at no extra cost in the narrowing half. Rows with $T = 2^h$ or 2^{60} assume an attacker restricted to one key.

				Cost [compression-function calls]			
Signatures T	Messages	k_1	k_2	Narrowing	Expandable	Linking	Total
2^{20} (XMSS $h = 20$)	maximal	20	55	2^{142}	2^{129}	2^{181}	2^{181}
2^{25} (LMS $h = 25$)	maximal	25	55	$2^{144.5}$	2^{129}	2^{176}	2^{176}
2^{40} (XMSS$^{\mathrm{MT}}$ $h = 40$)	maximal	40	55	2^{152}	2^{129}	2^{161}	2^{161}
2^{60} (XMSS$^{\mathrm{MT}}$ $h = 60$)	maximal	47	55	$2^{155.5}$	2^{129}	2^{154}	$2^{155.9}$
2^{64} (HSS or many keys)	maximal	47	55	$2^{155.5}$	2^{129}	2^{154}	$2^{155.9}$

Table 4: The *hourglass attack with state matching* against SHA-256 parameter sets with signed messages of maximal length (2^{55} blocks). For $T \ge 2^{60}$ the minimum of the cost estimate merges only 2^{47} of the signatures. The signer must hash 2^{55} blocks per signature, i.e. 2^{102} compression calls for the 2^{47} merged signatures, which is below the attacker's work in every row but far outside any practical use. Rows with $T = 2^h$ or 2^{60} assume an attacker restricted to one key.

At the NIST bound of 2^{64} signatures the SHA-256 parameter sets of HSS thus offer about 2^{172} against signatures on 64 MiB messages, 2^{166} against 4 GiB messages and 2^{156} in the formal limit of maximal-length messages, against a nominal 2^{256}; XMSS$^{\mathrm{MT}}$ with 2^{60} signatures is at 2^{176}, 2^{170} and 2^{156}. A single LMS tree of height 25 is at 2^{211}, 2^{205} and 2^{176} respectively, a single XMSS tree of height 20 at 2^{216}, 2^{210} and 2^{181}. Even without the expandable message, i.e. with the *hourglass attack with digest matching*, SHA-256 is by (3) at 2^{192} for 2^{64} signatures, at 2^{196} for 2^{60}, at 2^{231} for one LMS tree of height 25 and at 2^{236} for one XMSS tree of height 20. The single-tree figures hold for an attacker restricted to one key; against signatures collected under many keys every SHA-256 parameter set is at the 2^{64} figures.

4 Discussion

SHAKE256 parameter sets. SHAKE256/256 and SHAKE256/192 are not affected. Merging two sponge states requires a collision of the 512-bit inner part of the state (the capacity), which an absorbed block cannot influence, at a cost of 2^{256} per collision; internal-state targets are likewise 512-bit values. This prevents the herding technique and therefore multi-target attacks in our approach. The SHAKE128 parameter sets of RFC 8391 for 32-byte outputs, which SP 800-208 does not approve, have a 256-bit capacity and are limited to 2^{128} for every property by the flat sponge claim, independently of the attacks discussed here.

Resulting security levels. Table 5 collects the most pessimistic figures for the SHA-2 parameter sets of SP 800-208 under the adversary model of Section 1.5, with the signature bound of each family for an attacker restricted to one key, the 2^{64} bound for signatures collected under many keys, and, for SHA-256, two message-length assumptions.

Widening the target scope. The hourglass attacks are not tied to LMS and XMSS. The narrowing half only needs public prefix states of a Merkle–Damgård hash with a chaining state narrower than twice the targeted level, and the widening half only needs a table of target digests or internal states. The collected targets may therefore come from different signature schemes, from different protocols, or from uses of the hash function that are not signatures at all, as long as a second preimage under the respective prefix yields something the attacker wants; the attack framework combines the preimage searches against all these instances into one search. Even SHA-256 and SHA-256/192 digests can be targeted simultaneously in both hourglass attack variants, since chain processing is equal for the two and the digest matching can naturally be adjusted to match only the relevant bits. An attacker who can benefit from any one of a range of forgeries thus increases T, and with it the multi-target gain, beyond what a single target key, scheme, or protocol offers. The *hourglass attack with state matching* has a further property that widens the class of message formats it applies to: because it links into an internal state and copies the tail of the originally signed message, it preserves contents that possibly constrain the end of the message to a certain domain (e.g. a protocol or key identifier) associated with the matched digest. In contrast, the *hourglass attack with digest matching* ends the forged message with a block the attacker has to choose prior to knowing the matched digest and thus possibly violates such a domain constraint.

Deliberate limitation to 2^{64} messages in total. We deliberately cap our pessimistic estimates at 2^{64} signatures in total, although the hourglass attacks pool signatures under many keys without any extra effort. The reason is that the security frameworks state no bound, and there is no typical bound, on the number of keys an attacker may target; NIST's 2^{64} is stated for the single-key EUF-CMA game. Taking 2^{64} keys as a bound would turn the signature bound into $T = 2^{128}$ and remove it as the limiting quantity: the attack would then be limited only by the tradeoff between the narrowing half $2^{132+k_1/2}$ and the linking step. For the *hourglass attack with digest matching* against SHA-256 the terms balance at $k_1 \approx 82.7$ and the minimum of the estimate over integers lies at $k_1 = 83$, so the attacker would merge 2^{83} of the 2^{128} signatures and pay about $2^{174.3}$ instead of the 2^{192} of Table 1; for the *hourglass attack with state matching* with 4 GiB messages the minimum moves from $k_1 = 64$ to $k_1 = 66$ and the cost from $2^{166.3}$ to about $2^{165.6}$. Against SHA-256/192 nothing changes, because $T = 2^{64}$ already reaches the minimizing value $k_1 = 41$ of Table 1. We regard 2^{64} signatures in total as the defensible reading of the standard adversary model and leave the key count unbounded only in the sense that the 2^{64} may be spread over any number of keys.

Parameter family	Nominal	T	Cost [compression-function calls] msg. length $\leq$ 4 GiB	Cost [compression-function calls] msg. length max.	Matching	Security [bits] (bit operations) ($\leq$ 4 GiB / max.)
LMS-SHA256, $h = 25$	2^{256}	2^{25}	2^{205}	2^{176}	state	223 / 194
HSS-SHA256	2^{256}	2^{64}	2^{166}	2^{156}	state	184 / 174
XMSS-SHA2_20_256	2^{256}	2^{20}	2^{210}	2^{181}	state	228 / 199
XMSS$^{\mathrm{MT}}$-SHA2_60/*_256	2^{256}	2^{60}	2^{170}	2^{156}	state	188 / 174
any SHA-256 set, many keys	2^{256}	2^{64}	2^{166}	2^{156}	state	184 / 174
LMS-SHA256/192, $h = 25$[†]	2^{192}	2^{25}	2^{167} (any)		digest	185
HSS-SHA256/192[†]	2^{192}	2^{64}	2^{153} (any)		digest	171
XMSS-SHA2_20_192[†]	2^{192}	2^{20}	2^{172} (any)		digest	190
XMSS$^{\mathrm{MT}}$-SHA2_60/*_192[†]	2^{192}	2^{60}	2^{153} (any)		digest	171
any SHA-256/192 set, many keys[†]	2^{192}	2^{64}	2^{153} (any)		digest	171
all SHAKE256/256 sets	2^{256}	any	2^{256} (any)		–	–
all SHAKE256/192 sets	2^{192}	any	2^{192} (any)		–	–

Table 5: Most pessimistic classical cost of a forgery by reuse of a collected signature, in compression-function calls (SHAKE rows: generic bound), for the parameter families of SP 800-208. T is the number of signatures assumed available over all targeted keys (2^h for a single tree and 2^{60} for XMSS$^{\mathrm{MT}}$ if the attacker is restricted to one key; NIST's 2^{64} for HSS or for signatures collected under many keys); the "Matching" column names the hourglass attack variant, state matching or digest matching. Exponents are rounded to the nearest integer. The last column gives the estimated security upper bounds in bits from these attacks, measured as the base-2 logarithm of their work in bit operations. Following [24, Section 5.1], we count 2^{18} bit operations per SHA-256 compression, adding 18 to the compression-cost exponents shown in the table. For state matching the two numbers in the last column refer to messages of up to 4 GiB and to messages of maximal length, in the order of the two preceding columns. The SHAKE entries in the last column are left unspecified because this conversion applies to SHA-256. For digest matching and for the SHAKE sets the cost is independent of the message length, so one value is given for both columns ("any"). [†]The *hourglass attack with state matching* applies to these sets with exactly the costs of the corresponding SHA-256 rows, because it targets the 256-bit chaining state before the truncation to 192 bits; it is omitted here because digest matching is cheaper for every SHA-256/192 row (e.g. 2^{153} against 2^{156} for HSS with maximal-length messages). Smaller trees lose correspondingly less; see Tables 1 to 4.

5 Conclusion

The attacks of this work exploit the known structural weaknesses of Merkle–Damgård hashes whose chaining state is narrower than twice the targeted level, in their role as the message digest of HSS/LMS and XMSS. Hash-based signature schemes sign a message through a Winternitz one-time signature on its digest, whose size grows roughly quadratically, and whose signing time linearly, with the digest length, so LMS and XMSS have every reason to keep the digest as short as the security level allows. To make an n-bit digest sufficient they prepend a randomizer to the message, which removes the need for collision resistance, and the key identifier and leaf index, which are meant to deny the attacker any multi-target advantage. Our results show that this design does not reach n bits for an n-bit digest with a Merkle–Damgård hash whose chaining state is narrower than $2n$ bits. A wide pipe, as SHAKE256 with its 512-bit capacity and truncated SHA-512 provide, restores it. SHA-256 is a narrow pipe and loses up to 100 bits against 2^{64} signatures; SHA-256/192, whose state is wider than its digest but less than twice as wide, still loses 39 bits to the *hourglass attack with digest matching*. These attacks do not contradict the security proofs, whose guarantees are conditional on the stated hash-function

properties and idealizations. They instead expose limitations of applying those guarantees to the concrete SHA-256 message-hash instantiations at the claimed security levels.

In practical terms no concrete threat results from our findings. 2^{153} compression calls are far beyond reach, and the attacks with maximal-length messages require a signer who hashes exabytes per signature. The relevant conclusion is that with our formally justifiable reductions of the security applying to either of the SHA-2 variants, no SHA-2 parameter set achieves 192 bits against NIST's adversary. Since one typical incentive to use hash-based signatures is the requirement for high or long-term security, such as embedded firmware update with unmodifiable public keys as the root-of-trust, our results are relevant.

References

[1] Elena Andreeva, Charles Bouillaguet, Pierre-Alain Fouque, Jonathan J. Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. Second preimage attacks on dithered hash functions. In *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 270–288. Springer, 2008. doi: 10.1007/978-3-540-78967-3_16.

[2] Sydney Antonov. Round 3 official comment: SPHINCS+. NIST pqc-forum mailing list, 2022. URL https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/FVItvyRea28/m/mGaRi5iZBwAJ.

[3] Simon R. Blackburn, Douglas R. Stinson, and Jalaj Upadhyay. On the complexity of the herding attack and some related attacks on hash functions. *Designs, Codes and Cryptography*, 64(1–2):171–193, 2012. doi: 10.1007/s10623-011-9527-8. Also Cryptology ePrint Archive, Report 2010/030.

[4] David A. Cooper, Daniel C. Apon, Quynh H. Dang, Michael S. Davidson, Morris J. Dworkin, and Carl A. Miller. Recommendation for stateful hash-based signature schemes. Technical Report NIST Special Publication 800-208, National Institute of Standards and Technology, October 2020.

[5] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-damgård revisited: How to construct a hash function. In *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448. Springer, 2005. doi: 10.1007/11535218_26.

[6] Ivan Damgård. A design principle for hash functions. In *Advances in Cryptology – CRYPTO '89*, volume 435 of *LNCS*, pages 416–427. Springer, 1990. doi: 10.1007/0-387-34805-0_39.

[7] Richard Drews Dean. *Formal Aspects of Mobile Code Security*. PhD thesis, Princeton University, January 1999.

[8] Federal Office for Information Security (BSI). Cryptographic mechanisms: Recommendations and key lengths. Technical Report BSI TR-02102-1, Version 2026-01, BSI, January 2026.

[9] Scott Fluhrer. Further analysis of a proposed hash-based signature standard. Cryptology ePrint Archive, Report 2017/553, 2017. URL https://eprint.iacr.org/2017/553.

[10] Scott Fluhrer and Quynh Dang. Additional parameter sets for HSS/LMS hash-based signatures. RFC 9858, October 2025.

[11] Andreas Hülsing. Round 3 official comment: SPHINCS+. NIST pqc-forum mailing list, 2022. URL https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/Ca4zQeyObOY.

[12] Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. In *Public-Key Cryptography – PKC 2016*, volume 9614 of *LNCS*, pages 387–416. Springer, 2016. doi: 10.1007/978-3-662-49384-7_15.

[13] Andreas Hülsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: extended merkle signature scheme. RFC 8391, May 2018.

[14] Jonathan Katz. Analysis of a proposed hash-based signature standard. Cryptology ePrint Archive, Report 2016/1004, 2016. URL https://eprint.iacr.org/2016/1004.

[15] John Kelsey and Tadayoshi Kohno. Herding hash functions and the Nostradamus attack. In *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 183–200. Springer, 2006. doi: 10.1007/11761679_12.

[16] John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 474–490. Springer, 2005. doi: 10.1007/11426639_28.

[17] Tuomas Kortelainen and Juha Kortelainen. On diamond structures and Trojan message attacks. In *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 524–539. Springer, 2013. doi: 10.1007/978-3-642-42045-0_27. URL https://www.iacr.org/archive/asiacrypt2013/82700519/82700519.pdf.

[18] Stefan Lucks. A failure-friendly design principle for hash functions. In *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 474–494. Springer, 2005. doi: 10.1007/11593447_26.

[19] David McGrew, Michael Curcio, and Scott Fluhrer. Leighton-micali hash-based signatures. RFC 8554, April 2019.

[20] Ralph C. Merkle. One way hash functions and DES. In *Advances in Cryptology – CRYPTO '89*, volume 435 of *LNCS*, pages 428–446. Springer, 1990. doi: 10.1007/0-387-34805-0_40.

[21] National Institute of Standards and Technology. Secure hash standard (SHS). Technical Report FIPS PUB 180-4, U.S. Department of Commerce, August 2015.

[22] National Institute of Standards and Technology. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. Technical report, NIST, December 2016. Section 4.A.5, security strength categories.

[23] National Institute of Standards and Technology. Stateless hash-based digital signature standard. Technical Report FIPS PUB 205, U.S. Department of Commerce, August 2024.

[24] Ray Perlner, John Kelsey, and David Cooper. Breaking category five SPHINCS+ with SHA-256. In *Post-Quantum Cryptography – PQCrypto 2022*, volume 13512 of *LNCS*, pages 501–522. Springer, 2022. doi: 10.1007/978-3-031-17234-2_23. Also Cryptology ePrint Archive, Report 2022/1061.

[25] M. Stern. Re: Diversity of signature schemes. NIST pqc-forum mailing list, 2021. URL https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/2LEoSpskELs/m/LkUdQ5mKAwAJ.

[26] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999. doi: 10.1007/PL00003816.

[27] Ariel Weizmann, Orr Dunkelman, and Simi Haber. Efficient construction of diamond structures. In *Progress in Cryptology – INDOCRYPT 2017*, volume 10698 of *LNCS*, pages 166–185. Springer, 2017. doi: 10.1007/978-3-319-71667-1_9.

A Constructing Expandable Messages for SHA-256

The expandable message used by the *expandable-message attack* and the *hourglass attack with state matching* is expensive to obtain. It requires k birthday-scale collision searches on a 256-bit chaining state (one, for the Davies–Meyer compression of SHA-2; see Step 6). The saving is that these k collision pairs describe 2^k different lengths; the attacker does not search independently for a collision at every length. This appendix spells out the generic expandable-message method of Kelsey and Schneier [16], its use against the message hash of LMS and XMSS, and an explicit table-based implementation of the searches. None of the computations described here is feasible for full SHA-256 with current resources. The stage count k of this appendix is the k_2 of the main text.

A.1 Step 1: Fix the starting state and keep track of the block count

Write $f(h, B)$ for SHA-256 compression, where h is a 256-bit chaining state and B is a 512-bit (64-byte) block. For a string of r whole blocks $M = B_1 \parallel \cdots \parallel B_r$ let F denote the iteration

$$F(h, M) = f(\ldots f(f(h, B_1), B_2) \ldots, B_r), \qquad \text{so } F(h, B) = f(h, B).$$

Neither f nor F applies SHA-256's final padding; that happens only once, on the complete forged input. F returns only the chaining state; the number of processed blocks must be recorded separately. SHA-256 compression does not take the block number as an input, so the same state can be reached at different depths.

First consider an arbitrary fixed starting state h_0. For an unkeyed hash this can be IV. For the LMS message hash the first block must carry the prefix of the targeted signature, so the attacker processes a fixed first block

$$P_0 = I_i \parallel q_i \parallel D_{\mathrm{MESG}} \parallel C' \parallel m_0, \qquad h_0 = f(\mathsf{IV}, P_0),$$
$$|I_i \parallel q_i \parallel D_{\mathrm{MESG}} \parallel C'| = 22 + n/8 \text{ bytes}, \qquad |m_0| = 42 - n/8 \text{ bytes}.$$

Here m_0 becomes the start of the forged message, and C' is the randomizer that the forged signature will carry. Denote the number of fixed prefix blocks by d; this construction has $d = 1$ (for XMSS, whose prefix fills two blocks, $d = 2$, and m_0 is empty for SHA-256). With the *hourglass attack with state matching*, h_0 is replaced by the hourglass state h^* and the fixed prefix by the hourglass path, so that $d = 1 + k_1$. The expandable portion itself will have lengths k through $k + 2^k - 1$. Including the prefix shifts the range to $d + k$ through $d + k + 2^k - 1$. This explicit offset avoids having to constrain the collision-search blocks to contain the prefix. The unshifted range quoted in the main text describes the expandable portion.

A.2 Step 2: Construct one short/long collision pair

At stage t, for $t = 0, \ldots, k - 1$, the starting state h_t is already known. Construct two paths that end at one state h_{t+1}, but differ in length by exactly 2^t blocks:

1. Choose a fixed filler string Φ_t of 2^t blocks, for example all-zero blocks. Compute $g_t = F(h_t, \Phi_t)$ once, at a cost of 2^t compressions.

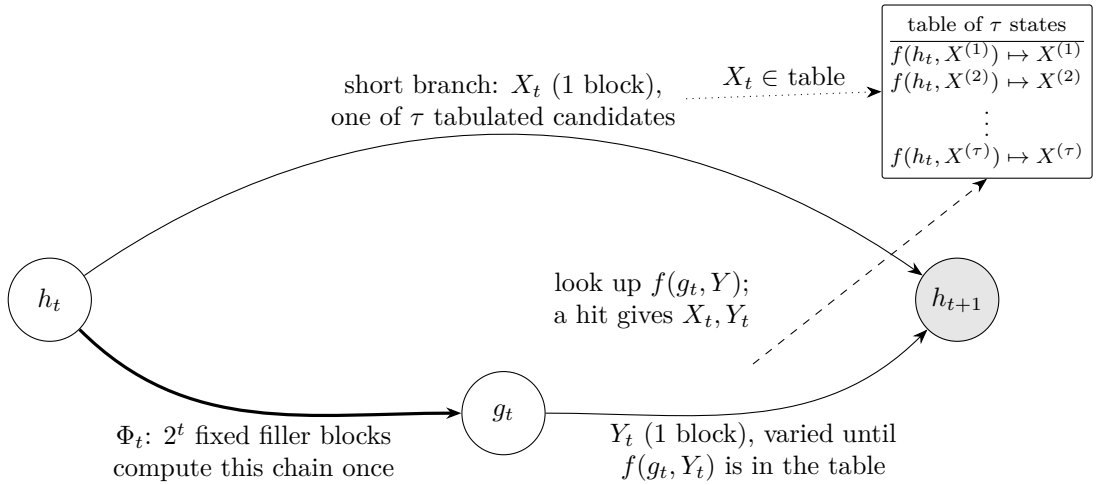
2. Enumerate τ distinct one-block candidates X (the table size τ is chosen below), compute $f(h_t, X)$, and store the resulting states with enough information to recover the corresponding X.

3. Enumerate one-block candidates Y, compute $f(g_t, Y)$, and look for a match in the table. The search varies only this last block; it does not recompute Φ_t per trial.

4. On a match, save the blocks X_t, Y_t and define

$$h_{t+1} = f(h_t, X_t) = f(g_t, Y_t) = F(h_t, \Phi_t \,\|\, Y_t).$$

The short alternative X_t has one block. The long alternative $\Phi_t \,\|\, Y_t$ has $2^t + 1$ blocks. Discard the search table before building the next stage.



$f(h_t, X_t) = f(g_t, Y_t)$; long branch length $= 2^t + 1$

Figure 7: One stage of the construction. Each arrow is labeled with the block or block string processed. The attacker computes the filler chain once and searches for a collision between the two final compression outputs. The meeting state is discovered by the search, not specified in advance. No SHA-256 final padding is applied on either branch. Thick arrows process several blocks.

The common endpoint is free to be any 256-bit value. This makes the search a collision between two sets of outputs, rather than a preimage search for a prescribed endpoint. Assuming compression outputs behave randomly for these candidates and the table contains approximately τ distinct states, each long-side trial succeeds with probability $\tau/2^{256}$. The expected stage work is therefore

$$W_t(\tau) \approx 2^t + \tau + \frac{2^{256}}{\tau}. \tag{5}$$

Choosing $\tau = 2^{128}$ balances table construction and probing, giving approximately $2^t + 2^{129}$ compression calls per stage. Thus the objection that collisions require extreme effort is correct: the construction pays that effort explicitly.

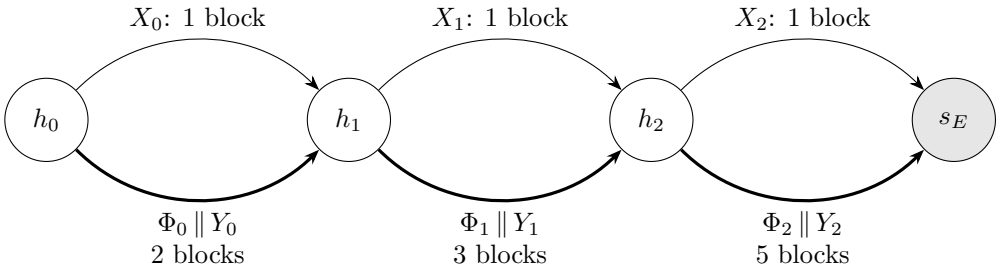
A.3 Step 3: Chain the pairs and obtain every length

Build the stages sequentially: stage $t+1$ starts at the common endpoint found at stage t. After k stages, let $s_E = h_k$. At every stage either branch arrives at the same next state, so every combination of branches reaches s_E.

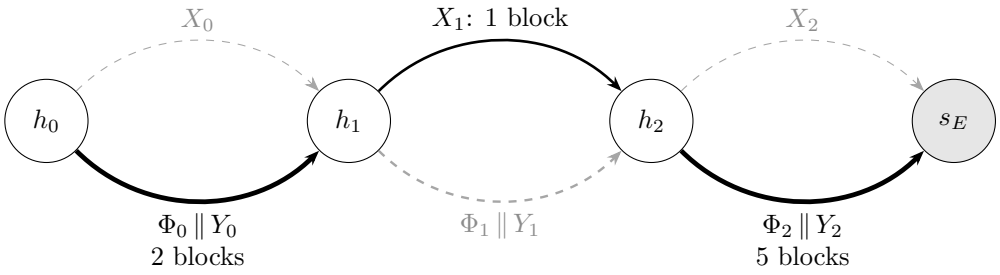
Let $\epsilon_t = 0$ select the short branch and $\epsilon_t = 1$ the long branch. The total length of the expandable portion is

$$L = \sum_{t=0}^{k-1} (1 + \epsilon_t 2^t) = k + \sum_{t=0}^{k-1} \epsilon_t 2^t. \tag{6}$$

There are therefore 2^k selectable lengths, $2^3 = 8$ in Figure 8a; Figure 8b exercises $\epsilon = (1, 0, 1)$, i.e. $L - k = 5 = 101_2$, giving $L = 8$. Every integer $L - k$ between 0 and $2^k - 1$ has a k-bit



(a) All $2^3 = 8$ routes. Each upper branch is one block; the lower branches are respectively 2, 3, and 5 blocks. Every route ends at the same s_E, while its length is $3 + \epsilon_0 + 2\epsilon_1 + 4\epsilon_2$ blocks, covering every length from 3 to 10.



(b) The route for $\epsilon = (1, 0, 1)$: long, short, long. Dashed branches are not taken. The selected blocks are concatenated to the $L = 8$-block expandable portion.

Figure 8: Three stages produce $2^k = 8$ selectable lengths (a); one of them exercised (b). Thick arrows process several blocks.

binary representation. Its bits are precisely the required branch choices. No further collision search is needed when selecting a length. For $k = 3$, the complete enumeration is:

ϵ_0	ϵ_1	ϵ_2	Expandable blocks L	With $d = 1$ fixed block
0	0	0	3	4
1	0	0	4	5
0	1	0	5	6
1	1	0	6	7
0	0	1	7	8
1	0	1	8	9
0	1	1	9	10
1	1	1	10	11

Only the k pairs and descriptions of their filler strings need to be retained. All-zero fillers can be regenerated from their lengths. It is unnecessary to store or enumerate all 2^k expanded messages. Producing one selected message still takes time proportional to its length. These are

collisions in the *chaining state* (Step 1), not in finalized digests: finalizing routes of different lengths would pad differently.

A.4 Step 4: Collect eligible states of the signed target inputs

For each known signed message M_i with hash input $x_i = I_i \,\|\, q_i \,\|\, D_{\mathrm{MESG}} \,\|\, C_i \,\|\, M_i$, compute the states s_i^j after j complete blocks from IV. Retain a state only when both conditions hold:

$$d + k + 1 \le j \le d + k + 2^k, \qquad j \le \ell_i - 1.$$

The first condition (the dotted box in Figure 9) ensures that a prefix of $j-1$ blocks can be selected before adding one bridge block. The second excludes the padded final block, whose state is the digest itself. Store each retained state with its target index i and block index j. Let N be the number of *distinct* retained states. Each input contributes about $\min(2^k, \ell_i - d - k - 1)$ of them, so $N \approx T \cdot \min(2^k, \bar{\ell})$ if duplicate states are rare; the main text writes this as $2^{k_1+k_2}$, the case $2^k \ge \bar{\ell}$ in which almost all message blocks are eligible. Padding states are not counted here. For $T > 1$ the signed inputs have distinct prefixes, and a state of input i is only useful if it is reached from a prefix of signature i; pooling the states of several inputs therefore requires the hourglass of Section 3.2, which merges the prefix states before the expandable message is built.

A.5 Step 5: Find a bridge, choose the length, and copy the tail

Enumerate one-block candidates B. For each, compute the single compression $f(s_E, B)$, which is the state reached after $P_0 \,\|\, E_L \,\|\, B$ for every length L of the expandable portion, and look the result up in the N-state target table. The table is indexed by state value, so one lookup tests the candidate against all N targets at once; a miss discards B, a hit returns (i, j). An independent trial succeeds with probability approximately $N/2^{256}$. After an expected $2^{256}/N$ trials, suppose the hit is

$$f(s_E, B^*) = s_i^j.$$

Figure 9 shows the search space and the flow of this step. Figure 5 of the main text shows the single-target case, where the N targets are the eligible states of one input, and Figure 6 the hourglass variant that pools the inputs of many signatures.

Now choose the expandable portion to have exactly $L = j - d - 1$ blocks. To do so, write $j - d - 1 - k$ in binary and select the branches specified by its bits. Append the bridge and the original input bytes starting at byte offset $64j$:

$$x^* = P_0 \,\|\, E_{j-d-1} \,\|\, B^* \,\|\, x_i[64j : |x_i|].$$

Here E_L denotes the selected L-block expansion. After the bridge, both chains have processed exactly j blocks and have the same state. Their remaining bytes and total lengths are identical. Applying ordinary SHA-256 padding therefore gives $\mathsf{H}(x^*) = \mathsf{H}(x_i) = Q_i$.

For example, with $d = 1$, $k = 3$, and a hit at $j = 9$, the expandable portion must have $L = 7$ blocks. Select short, short, long (the row $0, 0, 1$ in the table). The fixed first block, seven expandable blocks, and bridge total nine blocks; the copied tail starts at byte offset $9 \cdot 64 = 576$ of the original hash input, i.e. at byte $576 - (22 + n/8)$ of M_i.

For LMS, the forged message M' consists of the bytes of x^* after the $22 + n/8$ prefix bytes; the forged signature reuses σ_i and the leaf index q_i and carries C' in place of C_i. Reject an accidental output whose message was already signed. The construction permits arbitrary binary content; it does not guarantee a meaningful message or a valid application-specific format.

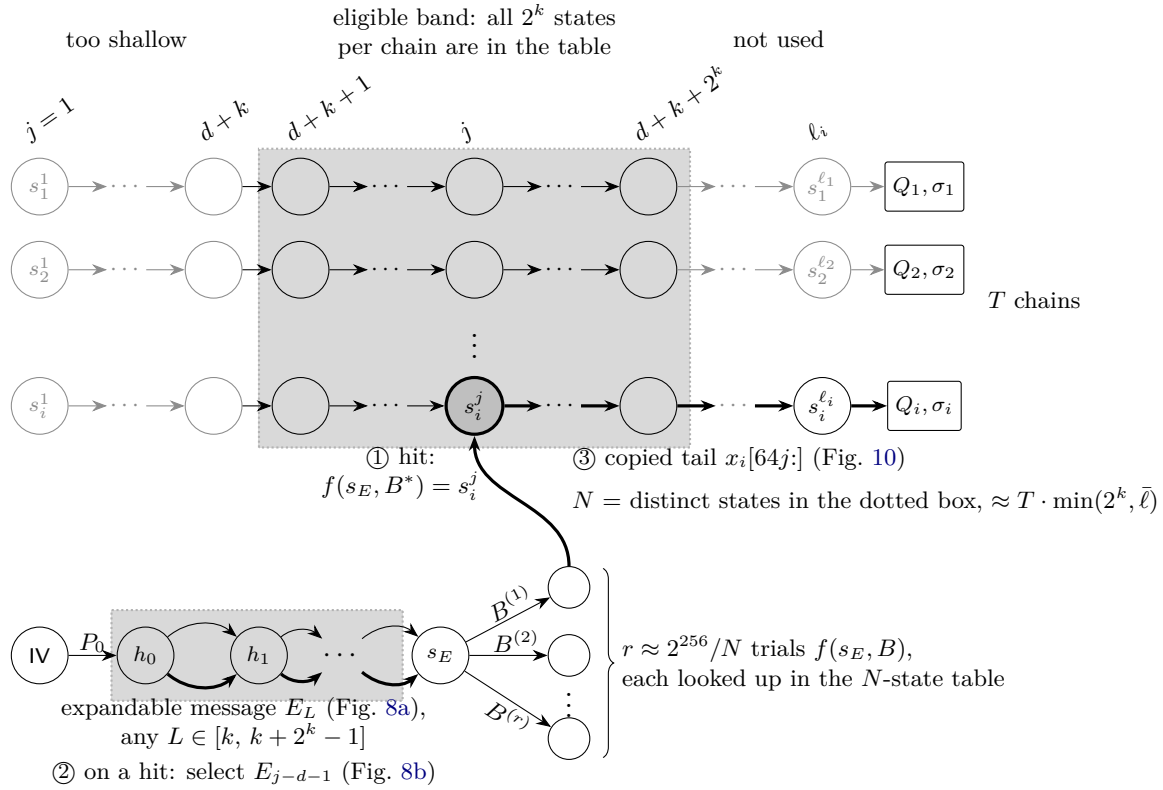


Figure 9: State matching against T inputs (for $T > 1$ after the hourglass, i.e. with $h_0 = h^*$). Bottom left: the fixed prefix and the expandable message give one state s_E, from which every trial block B produces one candidate. Top right: the table holds every state of every signed input whose depth lies in the dotted box (cut at $\ell_i - 1$ when the input is shorter); each candidate is looked up in this table, which tests it against all N states at once. On a hit ① at (i, j) the attacker selects the expansion length ② and copies the tail of input i ③.

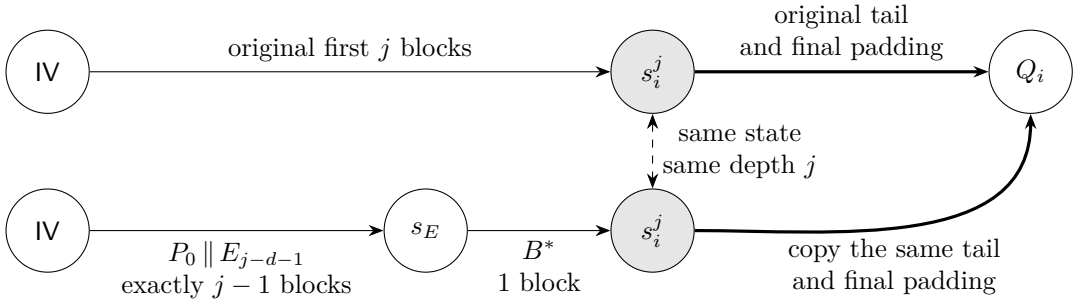


Figure 10: The bridge determines the target and its depth (continuing Figure 9). Only then does the attacker select an expansion of the required length. The two chains coincide in state and block count at the marked point; copying the original remaining bytes preserves the final digest.

A.6 Step 6: Account for setup, search, and memory

For an m-bit chaining state, the same table-based construction uses $\tau = 2^{m/2}$ states per collision search. Summing Equation (5) over the k stages gives

$$W_{\text{setup}} \approx d + (2^k - 1) + k\,2^{m/2+1}. \tag{7}$$

Let W_{scan} count the compression calls needed to scan the known target inputs to the relevant depths, and let ℓ_{out} be the output length in blocks. The overall work is

$$W_{\text{total}} \approx W_{\text{scan}} + d + (2^k - 1) + k\,2^{m/2+1} + \frac{2^m}{N} + O(\ell_{\text{out}}). \tag{8}$$

This counts compression calls and output work with efficient table access; signature acquisition, sorting or indexing overhead, and the physical cost of memory access are additional resources. In the examples below, the bridge search dominates the setup. The setup term is nevertheless mandatory; increasing the target count cannot make it vanish.

The explicit collision search above needs $2^{m/2}$ table entries at a time, in addition to the separate N-entry target table if both are kept concurrently. For SHA-256, the collision table's 256-bit states alone occupy 2^{133} bytes, before storing candidate identifiers. The N-state target table alone occupies $32N$ bytes, before storing (i, j). Thus the target-table size in the main text is *not* the peak memory requirement of this particular implementation. Other collision-search methods can trade memory and communication against computation; they require their own resource analysis. Even within the simple method, reducing the table to τ entries changes each stage's search term to $\tau + 2^{256}/\tau$ rather than leaving the 2^{129} estimate unchanged.

Appendix B gives a detailed setup procedure and figure for the fixed-point shortcut summarized here. The k-stage construction is the generic method of Kelsey and Schneier [16], which assumes nothing about the compression function. SHA-256 and SHA-512 use the Davies–Meyer mode, $f(h, B) = E_B(h) \boxplus h$, where E_B is the invertible 64- or 80-round state update keyed by the message schedule of B. A fixed point $f(h, B) = h$ therefore requires $E_B(h) = 0$, and $h = E_B^{-1}(0)$ is obtained for any chosen B by running the rounds backwards from the all-zero state, at the cost of one compression. Dean's construction [7], which Kelsey and Schneier generalize, exploits this: tabulate $2^{m/2}$ states $f(h_0, X)$ over random first blocks X, then compute fixed points $E_{B_{\text{fp}}}^{-1}(0)$ for random blocks B_{fp} until one lies in the table. The string $X \,\|\, B_{\text{fp}}^r$ reaches the common state for every $r \geq 0$. One birthday search of about $2^{m/2+1}$ compressions replaces the k searches, the filler term $2^k - 1$ disappears, every length from $d + 1$ upward is available (the depth window of Step 4 widens accordingly), and only X and B_{fp} have to be stored. Equation (7) becomes $W_{\text{setup}} \approx d + 2^{m/2+1}$, about 2^{129} for SHA-256 instead of the $2^{133.32}$ of the example below. The forged message then contains r copies of the 64-byte block B_{fp}, which is no worse than the random filler it carries anyway. The figures in this appendix keep the generic construction; for SHA-2 they are therefore upper bounds on the setup cost, and the bridge term $2^m/N$ dominates in every case either way. The sponge-based SHA-3 functions have no comparable fixed-point shortcut, and none would matter given $c \geq 512$.

A.7 Concrete SHA-256 example: 64 MiB messages

Take $k = 20$, $d = 1$, and messages of exactly 2^{26} bytes (64 MiB). With the 54-byte prefix the hash input has 2^{20} full blocks before the padded final block, which holds the last 54 message bytes and the padding. The eligible target depths are $j = 22, \ldots, 2^{20}$, giving $N_1 = 2^{20} - 21 =$ 1,048,555 states per message, assuming no duplicates. The expandable portion supports 2^{20} lengths, from 20 through 1,048,595 blocks. Only 20 collision pairs are built. For $T = 2^{64}$ messages the hourglass of Section 3.2 precedes the construction, with $h_0 = h^*$ and $d = 65$. The eligible depths then start at $j = d + k + 1 = 86$, giving $N_{\text{hg}} = 2^{20} - 85 = 1{,}048{,}491$ states per message and $N = 2^{64} N_{\text{hg}}$ targets in total, assuming no duplicates.

Σ	Term of Eq. (8)	Quantity	$T = 1$	$T = 2^{64}$, hourglass
+	d	fixed prefix blocks	1	65
+	$2^k - 1$	filler chains, 20 stages	$2^{20} - 1$	$2^{20} - 1$
+	$k\,2^{m/2+1}$	collision searches, generic	$20 \cdot 2^{129} \approx 2^{133.3}$	$20 \cdot 2^{129} \approx 2^{133.3}$
or	$2^{m/2+1}$	fixed-point shortcut (Step 6)	2^{129}	2^{129}
+	$2^{132+k_1/2}$	narrowing half of the hourglass	–	2^{164}
+	W_{scan}	target preprocessing	2^{20}	2^{84}
+	$2^m/N$	bridge search	$2^{256}/N_1 \approx 2^{236}$	$2^{256}/(2^{64}N_{\text{hg}}) \approx 2^{172}$
+	$O(\ell_{\text{out}})$	writing the forged input	$\approx 2^{20}$	$\approx 2^{20}$
=	W_{total}	compressions	$\approx 2^{236}$	$\approx 2^{172}$
	memory in bytes, not summed			
		collision table, 2^{128} states of 32 bytes	2^{133}	2^{133}
		target table, $32N$ bytes	$\approx 2^{25}$	$\approx 2^{89}$

In both columns the bridge search dominates the total: for $T = 1$ the setup terms lie 2^{103} below it, and for $T = 2^{64}$ the narrowing half of the hourglass lies 2^8 below it.

The generic collision setup is approximately $1.36 \cdot 10^{40}$ compression calls ($6.8 \cdot 10^{38}$ with the fixed-point shortcut). Even granting an illustrative rate of 10^{18} calls per second and ignoring all memory costs, setup alone would take about $4.3 \cdot 10^{14}$ years ($2.2 \cdot 10^{13}$ with the shortcut). The single-target and multi-target bridge searches are much larger still. The construction therefore explains a theoretical loss in the exponent of second-preimage resistance; it provides no practical means to forge an LMS or XMSS signature. The common endpoint s_E is obtained by paying the collision cost, and the range of lengths follows cheaply from the binary branch choices only afterwards.

B Fixed-Point Setup for Expandable Messages

The fixed-point shortcut mentioned in Appendix A.6 replaces the k collision searches of the generic expandable-message construction with one search. This is Dean's construction [7], discussed by Kelsey and Schneier [16]. In the main text and in Figures 5 and 6 the fixed-point block is written B_{fp} and the bridge block B; in this appendix, following Dean, the fixed-point block is B and the bridge block Z. We use the notation of Appendix A: f is compression without final padding, F iterates it over whole blocks, m is the chaining-state width, and h_0 is the state after a chosen d-block prefix. For SHA-256, $m = 256$ and each message block contains 64 bytes.

B.1 Compute a fixed point for a chosen block

SHA-256 compression has the Davies–Meyer form

$$f(h, B) = E_B(h) \boxplus h,$$

where E_B is the block cipher operation inside the compression function that is keyed by the message block B and $\boxplus$ adds the eight 32-bit words componentwise modulo 2^{32}. For a fixed message block B, its message schedule is known and the 64-round transformation E_B is invertible. The inverse here is the inverse of the round transformation, not of the compression function with feed-forward. Consequently, the state

$$h_B = E_B^{-1}(0) \quad \text{satisfies} \quad f(h_B, B) = 0 \boxplus h_B = h_B. \tag{9}$$

Running the rounds backwards from the all-zero state computes h_B with work comparable to one forward compression. SHA-512 admits the same construction with 80 rounds and eight 64-bit words.

This gives a block and its associated fixed-point state cheaply. It does not give a block that fixes a prescribed state, nor a way to reach h_B from the chosen h_0. Connecting the chosen prefix to one such state is the expensive part of setup.

B.2 Match a reachable state to a fixed point

Use two sets of candidates, as illustrated in Figure 11:

1. Choose τ distinct one-block candidates X, compute $f(h_0, X)$, and store each output with its corresponding block (or a recoverable candidate identifier), indexed by state value.

2. For fresh candidate blocks B, compute $h_B = E_B^{-1}(0)$ and look up h_B in the table. No chain of repeated blocks is evaluated during this search.

3. On a hit, retain the matching blocks X, B and define

$$s_E = f(h_0, X) = h_B, \qquad f(s_E, B) = s_E.$$

   The search table can now be discarded. The endpoint s_E is discovered by the search; it is not selected in advance.

Assuming the two sets behave as independent, approximately uniform m-bit states and the table contains approximately τ distinct values, each fixed-point trial hits with probability $\tau/2^m$. Counting forward and inverse round evaluations as comparable units, the expected setup work is

$$W_{\text{setup}}(\tau) \approx d + \tau + \frac{2^m}{\tau}. \tag{10}$$

The balanced choice $\tau = 2^{m/2}$ gives $d + 2^{m/2+1}$ work: approximately 2^{129} for SHA-256. The explicit table still needs about 2^{128} entries; the states alone occupy 2^{133} bytes. Keeping only X and B after the hit does not remove this peak setup memory requirement. A smaller table changes the work according to Equation (10).

B.3 Select the length after the bridge search

Let B^r mean r concatenated copies of B, with B^0 the empty string. The expandable portion is

$$E_L = X \,\|\, B^{L-1}, \qquad F(h_0, E_L) = s_E, \qquad L \geq 1. \tag{11}$$

Thus one saved pair supports every positive block length, within the hash's message-length limit. There is no k-stage construction or filler-chain precomputation. Emitting a selected message still takes time proportional to its length. These are equal intermediate states at different depths, not equal finalized digests: SHA-256 padding encodes the total input length.

For LMS with SHA-256, choose the same prefix-bearing first block as in Appendix A,

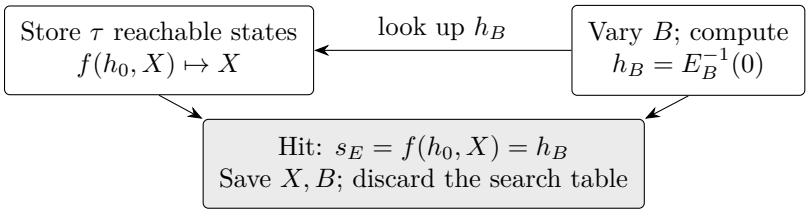
$$P_0 = I_i \,\|\, q_i \,\|\, D_{\text{MESG}} \,\|\, C' \,\|\, m_0, \qquad h_0 = f(\mathsf{IV}, P_0), \quad d = 1.$$

All candidate blocks X and B then lie within the forged message. For a signed input $x_i = I_i \,\|\, q_i \,\|\, D_{\text{MESG}} \,\|\, C_i \,\|\, M_i$, retain target states s_i^j whose depths satisfy

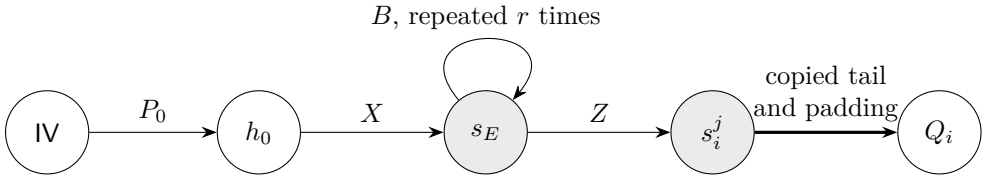
$$j \geq d + 2, \qquad j \leq \ell_i - 1.$$

The lower bound leaves room for the fixed prefix, X, and one bridge block; the upper bound excludes the padded final block. There is no additional upper depth bound imposed by an expansion parameter k. Let N count the distinct retained states. Search over bridge blocks

(a) Setup: search for a common state



(b) Use: repeat B to reach the required depth



$d + 1 + r + 1 = j$, so choose $r = j - d - 2$ after the bridge hits

Figure 11: Fixed-point expandable-message setup and use. (a) Match a state reachable from h_0 in one block to the fixed point computed for a candidate block B. (b) After the match, X enters s_E and every repetition of B leaves the state unchanged. A separate bridge block Z reaches a target state; its depth j determines how many copies of B to emit. All arrows represent unpadded compression steps.

Z until $f(s_E, Z) = s_i^j$, at expected cost $2^{256}/N$ compressions under the same random-state model. Then set $r = j - d - 2$ and form

$$x^* = P_0 \parallel X \parallel B^{j-d-2} \parallel Z \parallel x_i[64j : |x_i|].$$

The prefix through Z has exactly $d + 1 + (j - d - 2) + 1 = j$ blocks. It therefore matches the target in both state and block count. Copying the remaining bytes and applying final padding reproduces the signed digest. For example, with $d = 1$ and a hit at $j = 9$, emit P_0, X, six copies of B, and Z, then copy the target from byte offset 576 onward.

With W_{scan} denoting target scanning work and ℓ_{out} the output length in blocks, the balanced-table total is

$$W_{\text{total}} \approx W_{\text{scan}} + d + 2^{m/2+1} + \frac{2^m}{N} + O(\ell_{\text{out}}).$$

For SHA-256 this removes the generic setup terms $2^k - 1$ and $k\,2^{129}$ in favor of one 2^{129} search. It does not remove the bridge search or the separate target-table storage. The resulting message contains repeated binary blocks; no useful message semantics are guaranteed. Setup alone remains infeasible at full SHA-256 width.