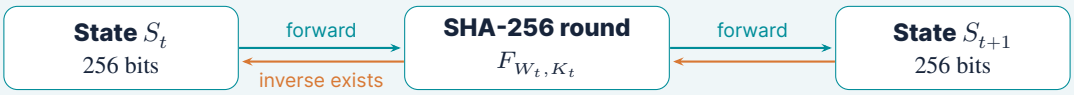
The Reversible Rounds of SHA-256

Why an apparently one-way hash is built from perfectly reversible round transformations

Tim Samshuijzen

31 October 2026 | complexity.zone



A fixed round is a permutation. The complete hash is not: many messages share each digest.

ABSTRACT

SHA-256 is designed as a one-way hash function, yet each of its 64 state-update rounds is an exactly invertible transformation of eight 32-bit registers, provided the round constant and message-schedule word are known. This note derives the inverse directly from the standardized SHA-256 equations, proves that each fixed round acts as a permutation on 2^{256} states, and checks the construction on the first round of the message `abc`. It then identifies the 64-round chain as the block cipher SHACAL-2 and explains why its invertibility does not give preimages: for a one-block message an attacker knows both ends of the chain but not the key that links them, and the Davies–Meyer feed-forward stops the inverse rounds from stepping backward from block to block. The result is an instructive separation between *invertible computation*, *unknown keys*, and *information loss at the hash interface*. The argument holds for the whole SHA-2 family; in its truncated variants, the digest reveals only part of the final state.

Scope. Expository derivation, not a new cryptanalytic attack. Reversible SHA-256 circuits [2, 3] and the block-cipher view of its rounds [4, 5] already appear in the literature.

1 The round function

SHA-256 processes each padded 512-bit message block using 64 rounds. During one round, the working state consists of eight 32-bit words

$$S_t = (a, b, c, d, e, f, g, h) \in (\mathbb{Z}/2^{32}\mathbb{Z})^8.$$

All arithmetic in this note is modulo 2^{32}, unless otherwise stated. Write $x \boxplus y$ for modular addition and $x \boxminus y$ for modular subtraction. Bitwise XOR is denoted $\oplus$; $\wedge$ and $\neg$ denote bitwise AND and NOT; ROTR^k rotates a 32-bit word to the right by k positions.

The Boolean functions and the two large-sigma functions are [1]

$$\mathrm{Ch}(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z), \tag{1.1}$$
$$\mathrm{Maj}(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z), \tag{1.2}$$
$$\Sigma_0(x) = \mathrm{ROTR}^2(x) \oplus \mathrm{ROTR}^{13}(x) \oplus \mathrm{ROTR}^{22}(x), \tag{1.3}$$
$$\Sigma_1(x) = \mathrm{ROTR}^6(x) \oplus \mathrm{ROTR}^{11}(x) \oplus \mathrm{ROTR}^{25}(x). \tag{1.4}$$

For round index $t \in \{0, \dots, 63\}$, let K_t denote its publicly specified constant and W_t its message-schedule word. Define

$$T_1 = h \boxplus \Sigma_1(e) \boxplus \mathrm{Ch}(e, f, g) \boxplus K_t \boxplus W_t, \tag{1.5}$$
$$T_2 = \Sigma_0(a) \boxplus \mathrm{Maj}(a, b, c). \tag{1.6}$$

The forward state update is precisely

$$\boxed{\begin{aligned} a' &= T_1 \boxplus T_2, \quad b' = a, \quad c' = b, \quad d' = c, \\ e' &= d \boxplus T_1, \quad f' = e, \quad g' = f, \quad h' = g. \end{aligned}} \tag{1.7}$$

> **The key structural clue.** Six of the eight new registers merely retain old register values. The two apparently overwritten words, d and h, are embedded in modular sums that can be undone once T_1 is reconstructed. Thus none of the 256 state bits needs to be lost by this round update.

The functions Ch and Maj do not have to be invertible individually. Nor must a modular sum be invertible if *both* operands are unknown. It is the *complete arrangement* in (1.7) that makes the transformation invertible.

2 A closed-form inverse

Use primes to label the state *after* a round. First recover six old registers directly:

$$\boxed{a = b', \quad b = c', \quad c = d', \qquad e = f', \quad f = g', \quad g = h'.} \tag{2.8}$$

Once a, b, c are known, compute the second temporary value:

$$\widehat{T}_2 = \Sigma_0(b') \boxplus \mathrm{Maj}(b', c', d'). \tag{2.9}$$

Then use $a' = T_1 \boxplus T_2$ and $e' = d \boxplus T_1$:

$$\widehat{T}_1 = a' \boxminus \widehat{T}_2, \tag{2.10}$$

$$d = e' \boxminus \widehat{T}_1. \tag{2.11}$$

Finally, the old h is obtained by rearranging (1.5):

$$\boxed{h = \widehat{T}_1 \boxminus \Sigma_1(f') \boxminus \mathrm{Ch}(f', g', h') \boxminus K_t \boxminus W_t.} \tag{2.12}$$

These formulas recover all eight original registers. No guessing, search, or discarded intermediate state is required.

Proposition 1 (Round bijectivity)**.** For every fixed pair $(W_t, K_t) \in (\mathbb{Z}/2^{32}\mathbb{Z})^2$, the round map

$$F_{W_t,K_t} : (\mathbb{Z}/2^{32}\mathbb{Z})^8 \longrightarrow (\mathbb{Z}/2^{32}\mathbb{Z})^8$$

is a bijection. Equations (2.8)–(2.12) describe its inverse.

Proof. Write G for the map defined by (2.8)–(2.12).

At most one preimage. Suppose $F_{W_t,K_t}(S) = S'$. By (1.7), six words of S are shifted words of S', as in (2.8). With a, b, c known, (1.6) fixes $T_2 = \widehat{T}_2$. Then $a' = T_1 \boxplus T_2$ fixes $T_1 = \widehat{T}_1$, $e' = d \boxplus T_1$ fixes d, and (1.5) fixes h as in (2.12). So $S = G(S')$.

At least one preimage. Given any S', let $S = G(S')$. Substituting the recovered a, b, c into (1.6) gives $T_2 = \widehat{T}_2$. Substituting e, f, g and the recovered h into (1.5) gives $T_1 = \widehat{T}_1$, by the construction of (2.12). Hence $T_1 \boxplus T_2 = a'$ by (2.10) and $d \boxplus T_1 = e'$ by (2.11), and the other six output words follow from (1.7). So $F_{W_t,K_t}(S) = S'$.

Every output state has exactly one preimage, so F_{W_t,K_t} is a bijection with inverse G. □

Corollary 1 (The entire round chain is a permutation)**.** For any fixed sequence of message-schedule words $(W_0, \ldots, W_{63})$, the 64-round map

$$P_W = F_{W_{63},K_{63}} \circ \cdots \circ F_{W_0,K_0} \tag{2.13}$$

is a permutation of the 2^{256} possible working states. Its inverse applies the 64 explicit inverse rounds in reverse order.

> **The precise meaning of "reversible."** This is a mathematical statement about the 256-bit *working state*, with the schedule fixed. It says nothing about whether the schedule can be inferred from the output. In fact, $W_0, \ldots, W_{15}$ are the 16 words of the original padded message block.

3 A worked hexadecimal example

Take the one-block message `abc`. Its first 32-bit padded message word is $W_0 = \texttt{61626380}$, while the first SHA-256 round constant is $K_0 = \texttt{428a2f98}$ [1]. The initial working state is the standardized SHA-256 initialization vector (IV).

Word	Before round 0	After round 0
a	6a09e667	5d6aebcd
b	bb67ae85	6a09e667
c	3c6ef372	bb67ae85
d	a54ff53a	3c6ef372
e	510e527f	fa2a4622
f	9b05688c	510e527f
g	1f83d9ab	9b05688c
h	5be0cd19	1f83d9ab

Table 1. The eight state words before and after the first SHA-256 round for the padded message `abc`. Each entry is eight hexadecimal digits.

The forward formulas give

$$\Sigma_0(a) = \texttt{ce20b47e}, \qquad \mathrm{Maj}(a,b,c) = \texttt{3a6fe667},$$
$$\Sigma_1(e) = \texttt{3587272b}, \qquad \mathrm{Ch}(e,f,g) = \texttt{1f85c98c},$$
$$T_1 = \texttt{54da50e8}, \qquad T_2 = \texttt{08909ae5}.$$

The inverse recovers T_1 from the first output register:

$$\texttt{5d6aebcd} \boxminus \texttt{08909ae5} = \texttt{54da50e8}.$$

It then reconstructs the overwritten d from the fifth output register:

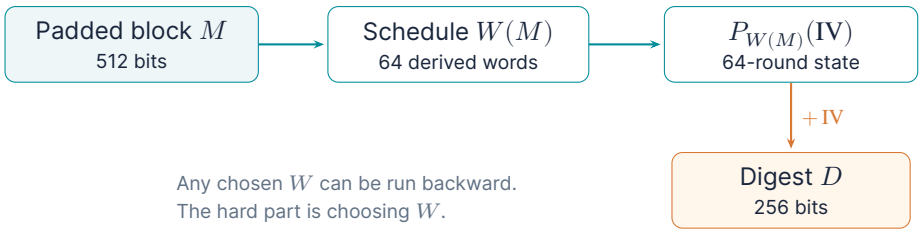
$$\texttt{fa2a4622} \boxminus \texttt{54da50e8} = \texttt{a54ff53a},$$

and reconstructs h by subtracting $\Sigma_1(e)$, $\mathrm{Ch}(e,f,g)$, K_0, and W_0 from T_1, giving `5be0cd19`. The remaining six input words are read directly from the shifted output registers.

> **Independent check.** The standard SHA-256 digest of `abc`, after all 64 rounds *and* the final feed-forward addition, is
>
> ```
> ba7816bf8f01cfea414140de5dae2223
> b00361a396177a9cb410ff61f20015ad
> ```
>
> This well-known test vector is unrelated to a shortcut for inversion: it is simply a convenient check of the full forward implementation.

4 Why this does not reverse the hash

The apparent paradox disappears once we say what an attacker knows and what they must find.



4.1 The 64-round chain is a block cipher

The map P_W of (2.13) is a block cipher: a family of permutations of a 256-bit block, selected by a key. The key is the 512-bit message block, which the message schedule expands into $W_0, \ldots, W_{63}$. This cipher has a name, SHACAL-2, and was submitted to the NESSIE project as a block cipher in its own right [4]. Its decryption is exactly the inverse of §2. SHA-256, like MD5 and SHA-1, builds its compression function from a block cipher (§4.3), and a block cipher must be invertible. The reversibility of the rounds is a design requirement, not an accident.

The first sixteen schedule words are the padded block itself, and all later words are derived from them [1]. To know the key is to know the message. Inverting P_W with a known W is therefore not a way of discovering an unknown message.

4.2 Both ends are known; the key is not

For a one-block message, the initial state $H^{(0)} = \mathrm{IV}$ is public, and the digest is

$$D = H^{(0)} \boxplus P_{W(M)}(H^{(0)}), \tag{4.14}$$

where $\boxplus$ is applied word by word. Subtracting the IV from D gives the final working state outright:

$$P_{W(M)}(\mathrm{IV}) = D \boxminus \mathrm{IV}. \tag{4.15}$$

This step needs no candidate message. An attacker who wants a one-block preimage therefore knows *both* ends of the 64-round chain, and can run the chain forward or backward for any W they like. What they lack is the key: they must find a padded block M that satisfies (4.15). In cipher terms, they need a SHACAL-2 key that is consistent with one known plaintext–ciphertext pair. Any such key will do; it need not be the one that was used.

The inverse rounds do not help with that search. For a chosen W, running the rounds backward from $D \boxminus \mathrm{IV}$ gives the starting state $P_W^{-1}(D \boxminus \mathrm{IV})$. Unless W is one of the correct keys, nothing makes that state equal the IV. If SHACAL-2 behaves like an ideal cipher, each guess succeeds with probability about 2^{-256}, and generic search costs about 2^{256} evaluations. The reversibility of the rounds was never the obstacle. Finding the key is.

4.3 What the feed-forward is for

For a message of several blocks, the chaining relation is

$$H^{(i)} = H^{(i-1)} \boxplus P_{W(M^{(i)})}(H^{(i-1)}). \tag{4.16}$$

The addition of $H^{(i-1)}$ after the rounds is the Davies–Meyer feed-forward [5]. It does not hide the key: for one block, §4.2 already showed that the key search is the obstacle. Its job is to stop the inverse rounds from stepping backward from one chaining value to the one before it.

Without the feed-forward, each compression step would be $H^{(i)} = P_{W(M^{(i)})}(H^{(i-1)})$. An attacker could choose any final block and run backward from D, which gives a pseudo-preimage (a chaining value and a block that hash to D) for free. Two-block preimages would then fall to a meet-in-the-middle search: run about 2^{128} first blocks forward from the IV, run about 2^{128} correctly padded final blocks backward from D, and look for a common chaining value. That costs about 2^{128} time and memory instead of 2^{256}.

With the feed-forward, stepping backward through (4.16) means solving $H \boxplus P_W(H) = H^{(i)}$ for H. Unlike P_W, the map $H \mapsto H \boxplus P_W(H)$ has no reason to be a permutation, and no efficient way to invert it is known.

The reversible core still leaves a visible trace. Choose any block M and let $H = P^{-1}_{W(M)}(0)$, the state that the 64 rounds send to zero. Then

$$H \boxplus P_{W(M)}(H) = H \boxplus 0 = H, \tag{4.17}$$

so H is a fixed point of the compression function for that block. Every block has one, and the inverse rounds compute it in 64 steps. This does not give preimages: to use the fixed point, a message must first lead from the IV to H, and finding one is itself a preimage problem.

4.4 Reversibility in published attacks

Published preimage attacks on step-reduced SHA-256 use the inverse of §2 directly. They split the steps into two chunks, compute one chunk forward and the other backward with (2.8)–(2.12), and look for a match where the chunks meet. Meet-in-the-middle attacks of this kind reach 43 of the 64 steps [6]. Biclique attacks reach 45 steps for preimages, and 52 steps for pseudo-preimages of the compression function [7]. All of these cost only slightly less than the generic 2^{256}, and none reaches the full 64 steps. Reversibility is a real tool for the cryptanalyst; it is not enough on its own.

4.5 Recovering the message versus finding a preimage

"Reversing a hash" can mean two different things.

Recovering the message. Even among messages of exactly 55 bytes, there are 2^{440} possibilities, all of which fit in one padded block. SHA-256 produces only 2^{256} possible digests. By the pigeonhole principle, at least one digest corresponds to at least 2^{184} distinct 55-byte messages. A digest therefore cannot in general identify *the* message. This is an information-theoretic limit, and no algorithm, fast or slow, gets around it.

Finding a preimage. An attacker who wants *some* message with digest D is not hindered by this limit. The many messages that share a digest are many correct keys for the search of §4.2. Whether one of them can be found efficiently is a computational question. That SHA-256 resists it is a cryptographic security assumption, supported by decades of cryptanalysis such as that of §4.4, and not a theorem established by the counting argument.

Two meanings that must not be confused. The round inverse is an efficient algorithm *given* W. Recovering *the* message from a digest is impossible in general, because many messages share each digest. Finding *any* message with a given digest is a key search like that of §4.2, and no efficient method is known for full SHA-256.

5 From mathematical to gate-level reversibility

An algebraically invertible function is not automatically a literal circuit of reversible elementary gates. Its internal implementation may overwrite temporary data. Reversible circuit design preserves or subsequently cleans that intermediate information.

For example, the map $(x, y) \mapsto (x, x \boxplus y)$ is bijective, since the original y is recovered by subtraction. At the bit level, CNOT and Toffoli gates are reversible building blocks:

$$\begin{aligned} \text{CNOT} &: (x, y) \mapsto (x, y \oplus x), \\ \text{Toffoli} &: (x, y, z) \mapsto (x, y, z \oplus (x \wedge y)). \end{aligned}$$

Both gates are their own inverses. A generic calculation can be embedded into a reversible computation by retaining its input and using an output register. Scratch registers can be cleaned through *uncomputation*:

$$\underbrace{(x, 0, 0)}_{\text{start}} \longrightarrow (x, f(x), 0) \longrightarrow (x, f(x), f(x)) \longrightarrow \underbrace{(x, 0, f(x))}_{\text{scratch cleared}}.$$

The middle step copies a *classical computational-basis output* into a fresh target using reversible XOR operations; the final step runs the first computation backward. This is the compute–copy–uncompute pattern used in reversible and quantum-oracle construction. Published SHA-256 circuit designs work at this lower level and analyze the cost of additions, ancillas, and cleanup [2, 3].

Three levels of reversibility

State map	For fixed W_t, K_t, the 256-bit round transformation is a permutation.
Circuit map	Reversible gates implement calculations with appropriate storage and cleanup.
Hash interface	Many messages share each 256-bit digest. Finding one means searching for a key, which the state inverse does not supply.

6 The rest of the SHA-2 family

FIPS 180-4 defines six SHA-2 hash functions [1]. All six use the round of (1.7): eight working words, the same Ch and Maj, and the same six shifts and two modular additions. They differ in word size, number of rounds, rotation amounts, constants and IV, and in how much of the final state they output.

Function	Word	Rounds	Block	State	Digest
SHA-224	32	64	512	256	224
SHA-256	32	64	512	256	256
SHA-384	64	80	1024	512	384
SHA-512	64	80	1024	512	512
SHA-512/224	64	80	1024	512	224
SHA-512/256	64	80	1024	512	256

Table 2. The six SHA-2 functions. Sizes are in bits. The block is the key of the round chain, the state is its working state, and the digest is the part of the final chaining value that is output.

The 64-bit functions work modulo 2^{64} and use [1, §4.1.3]

$$\Sigma_0(x) = \mathrm{ROTR}^{28}(x) \oplus \mathrm{ROTR}^{34}(x) \oplus \mathrm{ROTR}^{39}(x), \tag{6.18}$$

$$\Sigma_1(x) = \mathrm{ROTR}^{14}(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{ROTR}^{41}(x). \tag{6.19}$$

Nothing in §2 depends on these choices. The inverse (2.8)–(2.12) reads six words directly, recomputes T_2 and subtracts. The proof of Proposition 1 uses only the shape of the round, never the word size, the rotation amounts or the constants. Every SHA-2 round is therefore a permutation of its state, and the 80-round chain of the 64-bit functions is a permutation of 2^{512} states for every fixed schedule.

The block-cipher view carries over too. SHA-224 runs the SHA-256 rounds from a different IV, so its core is SHACAL-2 itself. The four 64-bit functions share one 80-round chain: a block cipher with a 512-bit block and a 1024-bit key. It is not SHACAL-2, but its decryption is again the inverse of §2, and the feed-forward of §4.3 plays the same role. The attack papers of §4.4 cover SHA-512 as well: meet-in-the-middle reaches 46 of its 80 steps [6], and bicliques reach 50 steps for preimages and 57 for pseudo-preimages of the compression function [7].

Truncation changes one step. Four of the six functions output only the left-most d bits of the final chaining value, which has $n = 256$ or 512 bits [1, §§6.3, 6.5–6.7]. For a one-block message, (4.15) then no longer gives the whole final working state. Because the feed-forward adds word by word, each complete word of D still gives the matching word of $P_{W(M)}(\mathrm{IV})$ outright, but the other $n - d$ bits are not revealed. The attacker knows the IV end of the chain fully, and the other end in only about d of its n bits. To run the rounds backward, they must first fill in the missing bits. That is extra freedom, not a shortcut: any filling may be used, but the chain must still lead back to the IV, and generic search for a preimage costs about 2^d evaluations. The counting argument of §4.5 holds for each function with its own sizes, and truncation only strengthens it: the same messages share fewer digests.

> **One argument for six functions.** Every SHA-2 round is a permutation for fixed W_t, K_t, and every 64- or 80-round chain is a block cipher keyed by the message block. What the digest hides is the key and, in the truncated functions, part of the final state as well.

7 Conclusion

SHA-256 has not secretly become reversible. Its compression function wraps a precisely reversible 256-bit *core*, the block cipher SHACAL-2, in a Davies–Meyer feed-forward. Six register shifts preserve most of the state, while two modular additions retain enough structure to reconstruct the overwritten words. The 64-round chain is therefore a permutation for every fixed message schedule. But the schedule is the key, and the key is the message an attacker is looking for. Even with both ends of the chain known, as they are for a one-block message, nothing reveals a key that links them. The feed-forward stops the inverse rounds from stepping backward from block to block, and the message-to-digest map cannot be injective. The same holds, with other sizes, for every function of the SHA-2 family (§6).

This makes SHA-256 a particularly vivid demonstration of a broader idea: *a computation may be reversible when all relevant inputs are retained, even though the external function it realizes is not uniquely invertible*.

REFERENCES

[1] NIST. *Secure Hash Standard (SHS)*. FIPS PUB 180-4 (2015), §§4.1.2, 4.2.2, 5.1.1, 5.3.3, 6.2.2. doi:10.6028/NIST.FIPS.180-4.

[2] M. Amy, O. Di Matteo, V. Gheorghiu, M. Mosca, A. Parent, and J. Schanck. "Estimating the Cost of Generic Quantum Pre-image Attacks on SHA-2 and SHA-3." *Selected Areas in Cryptography*, 2016. arXiv:1603.09383, §5.

[3] P. Kim, D. Han, and K. C. Jeong. "Time–space complexity of quantum search algorithms in symmetric cryptanalysis: applying to AES and SHA-2." *Quantum Information Processing* **17**, 339 (2018). doi:10.1007/s11128-018-2107-3, §6.

[4] H. Handschuh and D. Naccache. *SHACAL*. Submission to the NESSIE project, 2000.

[5] B. Preneel, R. Govaerts, and J. Vandewalle. "Hash Functions Based on Block Ciphers: A Synthetic Approach." *CRYPTO '93*, LNCS 773, pp. 368–378, 1994.

[6] K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki, and L. Wang. "Preimages for Step-Reduced SHA-2." *ASIACRYPT 2009*, LNCS 5912, pp. 578–597, 2009.

[7] D. Khovratovich, C. Rechberger, and A. Savelieva. "Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family." *FSE 2012*, LNCS 7549, 2012. IACR ePrint 2011/286.