A Practical Collision for 36-Step SHA-512

Junhong Huang
Boston University raymxnd@bu.edu

22 November 2026

Abstract

We give a colliding message pair for SHA-512 reduced to 36 steps, starting from the standard initial value. To our knowledge it is the first practical collision for 36-step SHA-512; the previous practical record was 35 steps [8]. The attack is not new. It uses the 36-step differential characteristic and the memory-efficient two-block framework of Li et al. [9], whose estimated time complexity is $2^{66.2}$. Our contribution is to make it practical by: (i) counting the degrees of freedom and the uncontrolled conditions more precisely; (ii) re-solving (A_0, E_4) on the fly for each first block, which makes the cost of the first block negligible; and (iii) a meet-in-the-middle lookup on 19 bits of E_{16} whose table fits in the GPU's L2 cache. The collision was found after about 3 hours of wall-clock time on three NVIDIA Blackwell GPUs (one RTX 5090 and two RTX PRO 6000), i.e. about 8.9 GPU-hours. At the compression-function throughput we measured on the same hardware, this equals about $2^{49.2}$ evaluations of the 36-step compression function. The measured stage-by-stage pass rates agree with the cost model; for example, the step-22 survivors that also satisfy δW_{23} number 9, against 9.3 expected. We verified the pair with an independent implementation, and independently re-measured the tail probabilities on $2 \cdot 10^{10}$ generated message pairs.

AI-use designation. Predominantly AI-generated work. This work was developed with AI agents under the author's direction. Google Gemini 4 Neon, running in the Antigravity IDE, wrote the code and ran the experiments. Anthropic Claude Opus 5.5 contributed the initial exploration, the independent verification in Sect. 6, and the writing. The author takes responsibility for the results and presentation.

1 Introduction

SHA-512 [2] has 80 steps. Collision attacks on step-reduced SHA-2 measure its security margin. Since 2024, automated searches for signed differential characteristics have combined with memory-efficient two-block frameworks, and the reach of these attacks has grown quickly [5, 6, 7, 8, 9, 4]. Table 1 summarizes the collision attacks on SHA-512 with the standard IV, using the list in [9, Table 1].

For comparison, the 35-step practical collision for SHA-512 took about 2^{54} work and 14 days on one RTX 5060 Ti [8]. For SHA-256, Goldzweig recently gave a 38-step collision from the standard IV, found in 53 GPU-hours after 52,300 vCPU-hours of precomputation [3].

Contributions.

- A colliding message pair for 36-step SHA-512 with the standard IV (Table 4). It satisfies the standard padding as well: two 256-byte messages with equal 36-step SHA-512 digests.

Table 1: Collision attacks on step-reduced SHA-512 with the standard IV (after [9, Table 1], updated with [4]). Time is in calls to the compression function.

Steps	Time	Status	Reference
27	—	practical	[1]
28	—	practical	[5]
29	—	practical	[7]
31	$2^{85.5}$	theoretical	[7]
35	$\approx 2^{54}$	practical	[8]
36	2^{136}	theoretical	[8]
36	$2^{66.2}$	theoretical	[9]
36	$2^{49.2}$ (measured; see Sect. 7)	**practical**	**this work**
37	2^{87}	theoretical	[9]
38	$2^{125.4}$	theoretical	[9]
38	$2^{100.7}$	theoretical	[4]
39	2^{178}	theoretical	[9]
39	$2^{131.3}$	theoretical	[4]

- A practical instantiation of the 36-step attack of [9]. Its expected cost is $2^{49.9}$ compression-function equivalents, against the estimate of $2^{66.2}$. Three changes account for about 12.7 of the 16.4 bits: a refined count of the degrees of freedom and conditions, on-the-fly re-solving of (A_0, E_4) for each first block, and a meet-in-the-middle lookup resident in L2 (Sect. 4). The remaining 3.7 bits reflect that a pair check costs less than a compression call (Sect. 7).
- Measured stage-wise statistics from $2.04 \cdot 10^9$ step-18 survivors, and an independent audit of the probabilistic tail (Sect. 6).

We stress that the differential characteristic, the message-difference pattern and the attack framework are those of Li et al. [9, 8]. We also reuse the internal-state solution of their published 36-step semi-free-start pair [9, Table 14] instead of solving that part again. None of the techniques in Sect. 4 is new in itself.

2 Preliminaries

SHA-512 processes 1024-bit blocks with 64-bit words. Let $\boxplus, \boxminus$ denote addition and subtraction modulo 2^{64}, and let $\delta x = x' \boxminus x$ be the modular difference. For $16 \le i < r$ the message expansion is

$$W_i = \sigma_1(W_{i-2}) \boxplus W_{i-7} \boxplus \sigma_0(W_{i-15}) \boxplus W_{i-16},$$

with $\sigma_0(x) = (x \ggg 1) \oplus (x \ggg 8) \oplus (x \gg 7)$ and $\sigma_1(x) = (x \ggg 19) \oplus (x \ggg 61) \oplus (x \gg 6)$. We use the usual two-register description, as in [9]. Step i computes

$$E_i = A_{i-4} \boxplus E_{i-4} \boxplus \Sigma_1(E_{i-1}) \boxplus \mathrm{IF}(E_{i-1}, E_{i-2}, E_{i-3}) \boxplus K_i \boxplus W_i,$$
$$A_i = E_i \boxminus A_{i-4} \boxplus \Sigma_0(A_{i-1}) \boxplus \mathrm{MAJ}(A_{i-1}, A_{i-2}, A_{i-3}),$$

where $(A_{-4}, \ldots, A_{-1}, E_{-4}, \ldots, E_{-1})$ is the chaining input. The r-step compression function adds the chaining input to the final state (feed-forward). A collision with the standard IV is a pair of messages $M_0 \| M_1$ and $M_0 \| M_1'$ with $M_1 \neq M_1'$ that share the first block M_0 and give the same chaining value after the second block.

3 The 36-step attack of Li et al.

The characteristic of [9, Tables 12–13] uses the message-difference family $i = 1$ of [8]. It puts differences in $(W_5, \ldots, W_9, W_{13}, W_{14})$. After expansion, the only further non-zero differences among $W_{16}, \ldots, W_{35}$ are $\delta W_{21} = -(2^{57} + 2^{51} + 2^{43})$ and $\delta W_{23} = 2^{59}$. The message differences realized by our pair are listed in Table 4. The attack follows the improved memory-efficient framework of [8]:

1. Find a solution for $(W_i)_{8 \le i \le 13}$, $(A_i)_{0 \le i \le 13}$ and $(E_i)_{4 \le i \le 13}$ that conforms to the middle part of the characteristic. The cost is about $2^{39.4}$ with a SAT solver.

2. For random first blocks M_0, compute the chaining value CV_1. Check the conditions on (E_3, W_5, W_6, W_7), which match (A_{-3}, A_{-2}, A_{-1}); there are $4+4+6+4$ of them, and empirically a valid M_0 occurs with probability 2^{-18}. Then derive $(W_0, \ldots, W_4)$.

3. Use the freedom in (W_{14}, W_{15}), estimated at 2^{21} values, to satisfy the 66 uncontrolled conditions on $(A_i, E_i, W_i)_{i \ge 16}$.

This needs $2^{66-21} = 2^{45}$ valid M_0. The total cost is estimated at $2^{66} + 2^{45} \cdot 2^{18} \approx 2^{66.2}$ [9].

4 Making the attack practical

4.1 Degrees of freedom in the second block

We reuse the Step-1 solution embedded in the published semi-free-start pair [9, Table 14]. We fix $(A_i)_{1 \le i \le 13}$, $(E_i)_{5 \le i \le 13}$ and $(W_i)_{9 \le i \le 13}$ to its values.

Upper part. Four bits of A_{14}, at positions $\{28, 40, 62, 63\}$, carry no condition. Changing A_{14} by Δ changes E_{14} and W_{14} by the same Δ, which gives 16 choices of (A_{14}, E_{14}, W_{14}). For each choice, a bit-level dynamic program over the carries of $E_{15} = A_{15} \boxplus (\cdot)$ enumerates the values of A_{15} that meet all bit conditions on A_{15} and E_{15}, plus three parity conditions. This yields exactly 2^{21} values of A_{15}, and hence of W_{15}, per choice: 2^{25} *upper* candidates in total, against 2^{21} in [9]. All 16 values of W_{14} satisfy both σ-difference conditions on W_{14} that are needed for $\delta W_{16} = 0$ and $\delta W_{29} = 0$; we checked this exhaustively.

Lower part: re-solving (A_0, E_4) on the fly. For a fixed CV_1 we keep $T_4 := E_4 \boxminus A_0$ fixed and let E_4 range over its 29 unconditioned bits, at positions $\{0..3, 5..17, 20..24, 27..29, 34..36, 63\}$. Setting $A_0 = E_4 \boxminus T_4$ leaves A_4 and every state word from step 5 on unchanged. The words $(E_0, \ldots, E_3)$ and $(W_0, \ldots, W_8)$ are recomputed by the formulas of [9, Step 2]. W_8 now depends on E_4, so its σ_0-difference condition must be checked again. This idea is close in spirit to the reuse of state freedom in [8, Sect. 4]. Before the enumeration, a first block is accepted ("golden") if the 14 conditions on E_3, W_6 and W_7 hold at the reference value of E_4. Over the 2^{29} choices of E_4, the conditions on $(E_3, W_5, W_6, W_7, W_8)$ then hold with measured probability 3.85% ($2^{-4.70}$). That gives about $2^{24.3}$ *lower* candidates per golden M_0, and the first block accounts for only 0.02% of the running time. The four conditions on W_5 reduce to the bitwise test $W_5[59] = W_5[2] = 0$, $W_5[52] = W_5[58]$, $W_5[53] \neq W_5[60]$. We checked on $1.3 \cdot 10^8$ samples that this test agrees exactly with $\delta\sigma_0(W_5) = -\delta W_{13}$.

4.2 Meet-in-the-middle on E_{16}

Write $W_{16} = \sigma_1(W_{14}) \boxplus W_9 \boxplus \sigma_0(W_1) \boxplus W_0$. Then

$$E_{16} = \ell \boxplus h_E, \qquad A_{16} = \ell \boxplus h_A, \qquad \ell := \sigma_0(W_1) \boxplus W_0,$$

where ℓ depends only on the lower part, and h_E, h_A depend only on the upper part. The bit conditions on E_{16} and A_{16} therefore turn into a matching problem on $\ell \boxplus h$. We sort the 2^{25} upper candidates by a 19-bit key on h, and file each lower candidate under the corresponding key on ℓ. The key consists of 1 exact bit ($E_{16}[0]$); 7 bits from three carry-guarded slices ($E_{16}[15..16]$, [17..18], [36..38]); 10 bits of E_{16} predicted modulo carries; and $A_{16}[59]$. Of the 2^{19} buckets, 61,365 are non-empty. With at most 64 lower candidates per bucket, the hot table is about 30 MB and stays in the L2 cache of the GPUs used (96 MB on the RTX 5090). It captures 99.0% of the lower candidates that hit a non-empty bucket.

The 19 key bits are not 19 conditions of the characteristic: the guard bits of the slices and the bits predicted modulo carries also reject some valid pairs. Per golden M_0, matching reduces the $2^{24.3} \cdot 2^{25} = 2^{49.3}$ lower–upper combinations to about $2^{30.3}$ enumerated pairs, while the density of solutions among them rises by only about $2^{8.7}$ (Table 6). Roughly $2^{-10.3}$ of the valid pairs of each golden M_0 are thus kept. This costs little, because a new golden M_0 takes only 0.02% of the running time.

4.3 Filtering pipeline and cost model

Every pair of a lower and an upper candidate in the same bucket goes through these filters in order:

1. 15 direct bits of E_{16}: 12 for step 17, plus 3 that step 18 forces ($E_{16}[43] = 1$, $E_{16}[51] = E_{16}[57] = 0$).
2. Two XOR conditions on E_{16}, and $A_{16}[59] = 0$.
3. Compute E_{17}; require that the step-18 difference vanishes and that $A_{17}[59] = 0$. Pairs that pass are called *step-18 survivors*.
4. Compute both messages fully and check steps 18–35. The checks are $\delta A = \delta E = 0$ where the characteristic requires it, $\delta E_{19} = -2^{59}$, then $\delta W_{23} = 2^{59}$, then $\delta W_{25} = 0$, and finally equality of the outputs.

Once δW_{23} and δW_{25} are correct, no difference remains in the state or in $W_{24}, \ldots, W_{35}$, so equal outputs follow. We confirm this empirically in Sect. 6. The expected number of step-18 survivors per collision is

$$2^{16.19} \text{ (steps 19–22)} \cdot 2^{11.51} \ (\delta W_{23}) \cdot 2^{3.83} \ (\delta W_{25}, \text{ probability } 9/128) = 2^{31.53}.$$

The two message-only factors were calibrated by Monte Carlo with $4 \cdot 10^9$ samples each. Steps 19–22 were calibrated on sampled step-18 states. On the hardware below, one step-18 survivor costs $2^{21.98}$ bucket-matched pairs, so a collision is expected after $2^{53.5}$ pairs.

5 Results

Setup. Three GPUs ran disjoint ranges of first-block seeds: one GeForce RTX 5090 (32 GB) and two RTX PRO 6000 Blackwell Server Edition. The code is a single CUDA program compiled for `sm_120`. Measured throughput was 272.6 G pairs/s and 65.8 k step-18 survivors/s on the RTX 5090, and 253.5 G pairs/s and 61.2 k survivors/s on each RTX PRO 6000. At this rate the model predicts 13.1 h on the RTX 5090 alone and 4.6 h on all three GPUs together.

Outcome. The RTX 5090 found the collision after 10,113 s (2.81 h). By then the two other GPUs had run for about 10,950 s each, so the total effort was about 8.9 GPU-hours. Table 2 lists the counters. Up to the collision, about $2.0 \cdot 10^9$ step-18 survivors had been processed. That is 65% of the expected $2^{31.53}$, close to the median of $\ln 2 \approx 69\%$. Table 3 compares the model with the observations. Our pair has $W_{14} = W_{14}^{\mathrm{ref}} \boxminus 2^{40}$, where W_{14}^{ref} is the value in [9, Table 14]; its A_{14} differs from the one in that pair only in bit 40. It therefore lies outside the 2^{21} upper candidates counted in [9], which keep $W_{14} = W_{14}^{\mathrm{ref}}$.

Table 2: Search counters. The "δW_{23}" and "δW_{25}" columns count step-22 survivors that also satisfy each message condition. The RTX PRO 6000 runs continued for about 285 s after the collision, until a stop file arrived; their counters include that time.

GPU	time (s)	golden M_0	step-18	step-22	δW_{23}	δW_{25}	collision
RTX 5090	10,113	2,102,288	665,107,101	8,845	5	1	1
RTX PRO 6000 #1	11,240	2,173,696	687,390,638	9,131	1	0	0
RTX PRO 6000 #2	11,240	2,174,720	687,630,738	9,147	3	0	0
Total	32,593	6,450,704	2,040,128,477	27,123	9	1	1

Table 3: Predicted against observed stage probabilities (whole run).

Stage	Model	Observed
step-18 $\to$ step-22	$2^{-16.19}$	$27{,}123/2.040 \cdot 10^9 = 2^{-16.20}$
step-22 $\to \delta W_{23}$	$2^{-11.51}$ (9.3 expected)	$9/27{,}123 = 2^{-11.56}$
$\delta W_{23} \to \delta W_{25}$	$9/128 = 0.070$ (0.63 expected)	1/9
$\delta W_{25} \to$ collision	1	1/1

6 Independent verification

The pair. We wrote a second implementation of SHA-512 from scratch and derived its round constants and IV from the cube and square roots of the first primes. For the full 80 steps it agrees with Python's `hashlib` on test vectors. Truncated to 36 steps, it confirms the following:

- CV_1 is the compression of M_0 from the standard IV;
- M_1 and M_1' differ exactly in words $\{5, 6, 7, 8, 9, 13, 14\}$;
- both messages compress to the same H;
- the padded 256-byte messages collide as well, while their full 80-step SHA-512 digests differ.

The script is given in Appendix A.

The tail model. The δW_{23} and δW_{25} conditions depend only on the message words. Their probabilities could therefore be biased by the way the search constructs its candidates. They could also be correlated with the state conditions of steps 17–22. To test both, a separate CPU program generates candidate pairs exactly as the search does: golden first blocks, then random E_4 variants,

Table 4: The colliding message pair for 36-step SHA-512 with the standard IV. Words are 64-bit, big-endian, listed in order. M_0 encodes the seed `0x1e1b34abbd2` followed by the ASCII strings "`SHA51236`" and "`COLLIDE!`".

M_0	000001e1b34abbd2	5348413531323336	434f4c4c49444521	0000000000000000
	(words 4–15 are 0000000000000000)			
CV_1	f1e42e9f77e4cca4	a785f886ea02658d	cf52ba6f18e44e11	644ba4c897d35d12
	c4cd7917344f6760	3f554360df46b76a	1a8281d8e746e746	0ef673a2b4c69e41
M_1	4f0a777b3e20699b	9084d21a7d15ffdb	f3eb37ed7c49a6a9	811b8eea0f813426
	9901a1886b80a163	36982650a4377b49	e2734f1dfc05e77e	65f45efd8b33bb64
	d395fbe4be6d579f	b0f8774e493b43bc	13287fa0a341d877	f0d19e5ae32f6abf
	f26d943cf8041873	06f2b17bb37eb2b7	2cccf09ec39c66dd	3284b0682fc4054b
M_1'	4f0a777b3e20699b	9084d21a7d15ffdb	f3eb37ed7c49a6a9	811b8eea0f813426
	9901a1886b80a163	3e982650a4377b49	e67b4f1dfc05e77e	6df45efd8b33bb64
	f3c5db65be6d579f	f0d8764e493b43bc	13287fa0a341d877	f0d19e5ae32f6abf
	f26d943cf8041873	02fab17bb37eb2b7	24ccf09ec39c66dd	3284b0682fc4054b
H	7f987d4f13fdfebe	99203ca3a1bf5993	1e4de96de6f5fbe5	ba20aadb881fc9cf
	33724a7027ab508b	3e25595a172fb3c8	6b67ab3917239813	209bb9516ff380cf

$\delta W_5 =$ `0800000000000000`, $\delta W_6 =$ `0408000000000000`, $\delta W_7 =$ `0800000000000000`, $\delta W_8 =$ `202fdf8100000000`, $\delta W_9 =$ `3fdfff0000000000`, $\delta W_{13} =$ `fc08000000000000`, $\delta W_{14} =$ `f800000000000000`; all other δW_i ($i < 16$) are zero. With standard padding, the 256-byte messages $M_0 \| M_1$ and $M_0 \| M_1'$ have the same 36-step SHA-512 digest:
`9eabd799baf71c41b426c3833361abcd9b9405a24d6ff6fa41f43e6071e437f3`
`79413d838b9f4634be19868ac36dca673d3e031474cf1a596f05c4e4546e9e84`.

then random upper candidates. Given message 1's real state and the characteristic's modular state difference before each step t, it checks whether step t reproduces the characteristic's difference. Table 5 summarizes $2 \cdot 10^{10}$ such pairs. The conditional probabilities of δW_{23} given steps 17, 19, 20, 21 and 22 all lie within $2^{-11.50}$ to $2^{-11.56}$ (standard errors 0.001–0.10 bits). Very few random pairs satisfy the step-17/18 conditions, so the conditioning on those steps is only checked to about ± 0.1 bit.

Table 5: CPU audit on $2 \cdot 10^{10}$ generated candidate pairs.

Quantity	Model	Measured
$\Pr[\delta W_{23} = 2^{59}]$	$2^{-11.5114}$	$2^{-11.5104 \pm 0.0006}$
$\Pr[\delta W_{25} = 0 \mid \delta W_{23} = 2^{59}]$	$9/128 = 0.07031$	0.07041
steps 24–35 hold | both	1	482,706/482,706
steps 0–16 hold for generated candidates	1	always
W_5 bitwise test $\equiv \sigma_0$ condition	exact	0 mismatches in $1.28 \cdot 10^8$

7 Comparison with the $2^{66.2}$ estimate

The estimate in [9] counts calls to the compression function. The search up to the collision enumerated about $8.3 \cdot 10^{15} \approx 2^{52.9}$ bucket-matched pairs. A pair check costs a handful of 64-bit integer operations, not a compression call. To convert between the two, we benchmarked a straightforward CUDA implementation of the 36-step compression function on the RTX 5090. Each thread compresses one block at a time, the message expansion is included, and all output words are

consumed. It runs at $2.15 \cdot 10^{10}$ compressions per second ($2^{34.3}$). The full 80-step function runs at $8.76 \cdot 10^9$ per second, the same order as public hashcat figures for SHA-512. We did not benchmark the RTX PRO 6000 directly; we assume it is 0.93 times as fast, which is the ratio observed in the search itself. On the RTX 5090, one bucket-matched pair check thus costs $2^{-3.66}$ compression calls. The work up to the collision amounts to $6.6 \cdot 10^{14} \approx 2^{49.2}$ compression calls, and the expected cost is $2^{53.51} \cdot 2^{-3.66} \approx 2^{49.9}$. At the same throughput, $2^{66.2}$ compression calls would take about 44 years on the three GPUs. Table 6 attributes the resulting gap of about $2^{16.4}$ approximately.

Table 6: Approximate attribution of the gap between $2^{66.2}$ and the expected cost of $2^{49.9}$ (in bits).

Source	bits
Uncontrolled conditions recounted on real transitions ($66 \to 62.2$)	≈ 3.8
Bucket-matched enumeration: $2^{62.2}$ combinations covered by $2^{53.5}$ enumerated pairs	≈ 8.7
Cost of one pair check, relative to one compression call (measured on RTX 5090)	≈ 3.7
First-block term 2^{63} removed by on-the-fly re-solving of (A_0, E_4)	≈ 0.2
Total	≈ 16.4

The largest share comes from the standard meet-in-the-middle matching. The original authors could have applied it themselves. The $2^{66.2}$ figure is best read as a conservative upper estimate, not as an error.

8 Conclusion

36-step SHA-512 with the standard IV is now practically broken: the collision took about 3 hours on three commodity Blackwell GPUs. The measured pass rates agree with the cost model to within statistical error. Full SHA-512 has 80 steps, and this result has no implication for its security. The next steps of the same family, 37 steps at 2^{87} [9] and 38 steps at $2^{100.7}$ [4], are outside the reach of the techniques used here; we have not tested whether similar refinements reduce those estimates.

Data availability. The CUDA search code, the CPU audit, the compression benchmark and the verification script are available at `https://github.com/Raymondlol/sha2-collide`.

References

[1] C. Dobraunig, M. Eichlseder, F. Mendel. Analysis of SHA-512/224 and SHA-512/256. ASIACRYPT 2015, LNCS, pp. 612–630. Springer, 2015.

[2] National Institute of Standards and Technology. FIPS 180-4: Secure Hash Standard (SHS), 2012.

[3] O. Goldzweig. A collision for 38-step SHA-256. Cryptology ePrint Archive, Paper 2026/2913, 2026.

[4] J. Gong, Y. Wei, F. Zhang, B. Wei. Collision attacks on SHA-2: 39-step SHA-256 below the birthday bound. Cryptology ePrint Archive, Paper 2026/2867, 2026.

[5] Y. Li, F. Liu, G. Wang. New records in collision attacks on SHA-2. EUROCRYPT 2024, LNCS, pp. 158–186. Springer, 2024.

[6] Y. Li, F. Liu, G. Wang, X. Dong, S. Sun. The first practical collision for 31-step SHA-256. ASIACRYPT 2024, LNCS, pp. 237–266. Springer, 2024.

[7] Y. Li, F. Liu, G. Wang, H. Qian, K. Jia, X. Kong. New collision attacks on round-reduced SHA-512. CRYPTO 2025, LNCS, pp. 200–229. Springer, 2025.

[8] Y. Li, F. Liu, G. Wang, J. Shi. Pushing the limit of memory-efficient collision attack framework for SHA-2. CRYPTO 2026. Cryptology ePrint Archive, Paper 2026/1080, 2026.

[9] Y. Li, Z. Zhang, M. Li, F. Liu, H. Qian, J. Zhu. Pushing collision attacks on SHA-2 to 39 steps. Cryptology ePrint Archive, Paper 2026/1120, 2026.

A Verification script

The script below needs only the Python standard library. It reads the record of the pair in JSON form, the file `results/collision_36step_sha512_std_iv.json` in the repository, given as its only argument.

```
# usage: python3 verify.py collision_36step_sha512_std_iv.json
import json, sys, hashlib, struct
M64 = (1 << 64) - 1
def primes(n):
    ps, x = [], 2
    while len(ps) < n:
        if all(x % p for p in ps if p * p <= x): ps.append(x)
        x += 1
    return ps
def iroot(n, k):
    lo, hi = 0, 1 << (n.bit_length() // k + 1)
    while lo < hi:
        mid = (lo + hi + 1) // 2
        lo, hi = (mid, hi) if mid ** k <= n else (lo, mid - 1)
    return lo
P = primes(80)
K  = [iroot(p << 192, 3) & M64 for p in P]      # frac(cbrt p) * 2^64
IV = [iroot(p << 128, 2) & M64 for p in P[:8]]  # frac(sqrt p) * 2^64
rotr = lambda x, n: ((x >> n) | (x << (64 - n))) & M64
S0 = lambda x: rotr(x, 28) ^ rotr(x, 34) ^ rotr(x, 39)
S1 = lambda x: rotr(x, 14) ^ rotr(x, 18) ^ rotr(x, 41)
s0 = lambda x: rotr(x, 1) ^ rotr(x, 8) ^ (x >> 7)
s1 = lambda x: rotr(x, 19) ^ rotr(x, 61) ^ (x >> 6)
def compress(cv, M, steps):
    W = list(M) + [0] * (steps - 16)
    for i in range(16, steps):
        W[i] = (s1(W[i-2]) + W[i-7] + s0(W[i-15]) + W[i-16]) & M64
    a, b, c, d, e, f, g, h = cv
    for t in range(steps):
        t1 = (h + S1(e) + ((e & f) ^ (~e & g)) + K[t] + W[t]) & M64
        t2 = (S0(a) + ((a & b) ^ (a & c) ^ (b & c))) & M64
        h, g, f, e, d, c, b, a = g, f, e, (d + t1) & M64, c, b, a, (t1 + t2) & M64
    return [(x + y) & M64 for x, y in zip(cv, [a, b, c, d, e, f, g, h])]
def blocks(m):
    L = len(m) * 8
    m += b"\x80" + b"\x00" * ((112 - (len(m) + 1) % 128) % 128) + L.to_bytes(16, "big")
    return [list(struct.unpack(">16Q", m[i:i+128])) for i in range(0, len(m), 128)]
def sha512r(m, steps):
    cv = IV
    for B in blocks(m): cv = compress(cv, B, steps)
    return b"".join(x.to_bytes(8, "big") for x in cv)
for m in [b"", b"abc", bytes(range(256)) * 3]:
    assert sha512r(m, 80) == hashlib.sha512(m).digest()
J = json.load(open(sys.argv[1])); h = lambda L: [int(x, 16) for x in L]
M0, M1, M2 = h(J["M0"]), h(J["M1"]), h(J["M2"])
cv = compress(IV, M0, 36)
assert M1 != M2 and compress(cv, M1, 36) == compress(cv, M2, 36)
a = b"".join(x.to_bytes(8, "big") for x in M0 + M1)
b = b"".join(x.to_bytes(8, "big") for x in M0 + M2)
assert sha512r(a, 36) == sha512r(b, 36)
assert hashlib.sha512(a).digest() != hashlib.sha512(b).digest()
print("36-step SHA-512 collision verified:", sha512r(a, 36).hex())
```