Bitsliced 4-bit S-boxes with AVX-512 ternary logic

Jussi Kivilinna · November 2026 · version 2026-11-15

> AVX-512 `vpternlogd` computes any 3-input boolean function in one instruction. I searched, with a SAT solver, for the shortest straight-line programs of such gates for the Serpent S-boxes and for every affine equivalence class of 4-bit permutations, and confirmed the gate counts with an exhaustive search. Every Serpent S-box and its inverse needs 7 or 8 gates, against 9–13 logic instructions that GCC generates from the classic Osvik sequences. On Zen 5, where chains of these instructions take 2 cycles per step, the round's critical path matters more than the gate count, so a second search minimises the critical path across the whole round. For the 302 affine classes of 4-bit permutations, the cheapest member of every class needs at most 7 gates, and only two classes need 7; all minimums are proven and all programs are listed in appendix C.

1. Introduction

Serpent is specified in bitslice form: each round applies one of eight 4-bit S-boxes to 32 columns of four 32-bit words, so an S-box is a short sequence of boolean instructions on whole registers. Osvik (2000) found sequences for x86 with only five registers and two-operand instructions; those sequences are still what most Serpent implementations use, including the SIMD ones (see e.g. Speeding up Serpent: SIMD Edition).

AVX-512 changes the cost model. There are 32 vector registers, the two-input logic instructions have non-destructive three-operand forms, and `vpternlogd a, b, c, imm` replaces `a` with an arbitrary 3-input function of `a, b, c`, given by the 8-bit truth table `imm`. A compiler can merge some two-input operations into ternlog instructions, but only greedily: compiled for AVX-512 with GCC 15, Osvik's sequences of 16–20 instructions per S-box (including one or two register copies and up to two NOTs) become 9–13 logic instructions. The question here is how few ternlog gates are needed when the S-box is searched for directly.

Ullrich et al. studied the same question for two-input instructions over all affine equivalence classes of 4-bit permutations, and found the cheapest members of 272 of the 302 classes before their search became too expensive. The ternlog version of that question turns out to be small enough to answer completely.

Heuristic searches with ternlog gates already exist. sboxgates (Dansarie, 2021) can build circuits from 3-input lookup tables, and Sovyn et al. (2023) give ternlog counts for 225 4-bit S-boxes, the Serpent S-boxes among them. Their Serpent counts are the ones proven minimal here (section 3.1). What is new here is the proof of minimality, the depth across the whole round, and the complete search over all classes.

2. Model and search

A *gate* here is one ternary-logic instruction: an arbitrary Boolean function of three inputs, a 3-input lookup table selected by `imm`. The gate count is therefore the circuit size over the full 3-input basis, not a count of traditional two-input gates.

An S-box is a program of k gates t_i `= T(a, b, c; imm)`, where `a, b, c` are earlier values (inputs `x0..x3` or earlier gates) and `T` is the ternlog function. Every two-input operation, including `andnot`, is a special case,

and NOT is free because it folds into `imm`. The cost is k; register allocation is left to the compiler. Each value is a 16-bit truth table over the 16 possible inputs, so the whole search fits in a SAT instance:

- one-hot selector variables pick the three operands of each gate, with `a < b < c` since `imm` can absorb any operand permutation;
- 8 free variables per gate hold `imm`, and 16 clauses per input row tie the gate output to its operands;
- each output must equal one gate's truth table.

Symmetry breaking keeps the instances small: gate values are normalised to 0 at input 0 (complements are free), all gate values are distinct and used, and adjacent independent gates are kept in a fixed order. The gate count is raised until the instance becomes satisfiable, so the first solution is optimal, given that the symmetry breaking keeps at least one optimal program. Appendix D gives the full encoding and the argument for that. The gate counts, and except for S3 the depth results of section 3.3 at the minimum gate count, are also confirmed by a separate exhaustive search (sections 3.3 and 4). The solver is CryptoMiniSat 5.11; every result is checked by simulation. This is the approach of Stoffelen, with ternlog gates and without register modelling.

3. Serpent

3.1 Gate count

All sixteen Serpent S-boxes (S0–S7 and their inverses) need 7 gates, except S3, S7, SI3 and SI7 which need 8. In every case one gate fewer is impossible: the SAT search finds no program, and neither does the exhaustive search of section 4 run on the S-box itself, which takes under a minute per S-box. The heuristic search of Sovyn et al. reaches the same counts. The programs are in appendix A.1. Compiled one S-box at a time with GCC 15 (inputs loaded from memory, outputs stored), they give exactly 7–8 logic instructions, against 9–13 for the Osvik versions: 58 instead of 88 for the encryption S-boxes and 58 instead of 92 for decryption.

These counts leave out register copies. `vpternlogd` overwrites its first operand, so when that value is still needed the compiler first copies it with `vmovdqa64`. On Zen 5 a copy adds no measurable latency (it is handled at register rename), but it takes a front-end slot; section 3.4 counts them.

3.2 Latency

Gate count is the right measure when a CPU is limited by instruction throughput. On Zen 5, however, a chain of dependent 512-bit operations, each using the previous result, takes about 2 cycles per operation (timed against a chain of scalar adds):

instruction (zmm)	Zen 5, per step of a dependent chain
`vpxord`, `vpandd`, `vpandnd`, `vprold`, `vpslld`	≈2 cycles
`vpternlogd` (any imm)	≈2 cycles
`vmovdqa64` register copy	≈0 (renamed)

This is a property of such chains rather than one latency per instruction: uops.info measures 1 or 2 cycles on Zen 5, depending on which operand carries the dependency. A bitsliced S-box, the linear transform and the key mixing are all chains of this kind. An implementation that interleaves only a few independent blocks per round is then latency bound, and a gate-count optimal S-box can have a longer dependency chain through the round than a less compact one. On Zen 4 and Tiger Lake, uops.info lists 1 cycle for these operations.

3.3 Depth across the round

The S-box depth alone is not the right measure, because the chain runs through the linear transform (LT) and key mixing. For each S-box the search minimises

```
max over outputs k ( max over inputs j (A[j] + D[j][k]) + T[k] + P[k] )
```

where `D[j][k]` is the longest gate path from input *j* to output *k*, `A` the relative arrival time of each input from the previous LT, `T` the remaining path from each output through the next LT, and `P[k]` = 1 when output *k* is not a native two-input operation: a following XOR, such as the round key, can be merged into a native operation as one ternlog, but not into a ternlog. The values assume the usual round structure (key XOR, S-box, LT for encryption; inverse LT, inverse S-box, key XOR for decryption) with a 3-way XOR as one ternlog and a rotate or shift as one operation. A and T are computed from the LT alone, as if all its inputs arrived together; the real arrival times also depend on the output depths of the neighbouring S-box, which the model ignores. Encryption has A = (3,0,3,1), T = (2,1,2,0), P = 0, and decryption has A = (0,0,0,1), T = (1,0,0,0). The search keeps a bound on this objective and lowers it until the instance is unsatisfiable.

Every output bit of a Serpent S-box depends on all four inputs. For encryption this bounds the objective at 6: every output depends on x0, which arrives at 3, so output 0 has depth at least 4, plus T[0] = 2. For decryption the same argument gives only 3. Objective 3 would need output 0 to be a native operation on x3 and a one-gate function of x0, x1, x2; the output is balanced, so that operation must be XOR, and no inverse Serpent S-box has `y0 = x3 ⊕ f(x0, x1, x2)`. The bound for decryption is therefore 4.

Decryption reaches 4 for every inverse S-box when one or two extra gates are allowed. Encryption reaches 7 with at most one extra gate, and 6 is proven impossible even with two extra gates.

The objective at the minimum gate count is confirmed by the exhaustive search of section 4, extended to this model. A value's depth is the smallest over all operand triples that compute it, counting only the operands it depends on; whether an output is a native operation is decided once the program is complete, over every polarity of the hidden gates; and a gate is moved before its predecessor only if it reaches the same depth, and for an output the same native options, without it. Every program found is checked with the SAT searcher's own objective. For the twelve S-boxes with 7 gates the search finds a program at the value in the table and none one level lower. For S7 and SI7 at 8 gates, where the SAT search timed out, it finds none one level lower (in 56 and 35 minutes on 32 threads), so their values are exact as well. SI3 is at the bound of 4, and only for S3 is the level below its 7 excluded by SAT alone. The results with extra gates rest on the SAT search alone, and for decryption on one restriction of it (appendix D.5).

S-box	min gates	first solution	lowest at min gates	chosen gates	chosen objective

S0	7	9	8	8	7
S1	7	9	7	7	7
S2	7	9	7	7	7
S3	8	8	7	8	7
S4	7	11	8	8	7
S5	7	10	8	8	7
S6	7	10	8	8	7
S7	8	9	8	9	7
SI0	7	6	5	8	4
SI1	7	6	6	8	4
SI2	7	8	5	8	4
SI3	8	8	4	8	4
SI4	7	7	7	8	4
SI5	7	8	6	8	4
SI6	7	8	5	9	4
SI7	8	6	5	9	4

"First solution" is the objective of the program the gate-count search returned first, with no depth objective. These programs are the "fewest gates" set of section 3.4 and are listed in appendix A.3.

The depth-optimised programs, with one or two extra gates where that lowers the objective, are in appendix A.2.

3.4 Performance

The depth-optimised S-boxes of appendix A.2 have been posted as a patch to libgcrypt's AVX-512 Serpent implementation (Kivilinna, 2026), replacing S-boxes derived from Osvik's. ECB throughput of Serpent-128 in cycles per byte, before and after. The percentages are the change in cycles per byte; the patch message gives the same results as throughput gains (27% and 14% on Zen 5). The patch also lists an Intel Core i3-1115G4 (Tiger Lake) measured with turbo; the Tiger Lake figures used here are the base-clock runs in the next table.

CPU	encryption	decryption
AMD Ryzen 9 9950X3D (Zen 5)	2.10 → 1.65 (−21%)	1.77 → 1.55 (−12%)
AMD Ryzen 9 7900X (Zen 4)	2.66 → 2.23 (−16%)	2.55 → 2.30 (−10%)

A second set of runs compares all four S-box sets on the same machines: Osvik's, the fewest gates with no depth objective (appendix A.3), the fewest gates with the lowest depth at that count (appendix A.1), and the depth-optimised set (appendix A.2). All four are written like the patch, with `^` for XOR, which matters to

GCC (see below). ECB cycles per byte of libgcrypt's AVX-512 Serpent-128 (`tests/bench-slope`, GCC 15.2, boost or turbo off), encryption / decryption:

CPU	Osvik	fewest gates	fewest gates, lowest depth (A.1)	depth-optimised (A.2)
Zen 5 (4.32 GHz)	2.10 / 1.77	1.94 / 1.84	1.69 / 1.64	1.65 / 1.55
Zen 4 (4.65 GHz)	2.67 / 2.56	2.36 / 2.40	2.28 / 2.28	2.23 / 2.30
Tiger Lake (2.99 GHz)	1.92 / 1.97	1.69 / 1.82	1.66 / 1.80	1.72 / 1.81

Cycles per byte are nanoseconds per byte times the measured clock. The same statically linked binaries ran on all three machines. On Zen 5 and Zen 4 the four sets ran at the same time on separate physical cores, in two passes with the sets rotated between cores; the figures are the mean, and the passes differ by at most 0.005 cycles per byte. Tiger Lake has two cores, so there the sets ran one at a time on one core, six runs each. Its decryption lands on one of two levels from run to run of the same binary, 0.1 to 0.2 cycles per byte apart, and in ECB and CBC mode independently within a run, which points to memory layout rather than the S-boxes. The Tiger Lake figures are the fastest of the six runs. Within the faster level the runs of a set differ by at most 0.02 (one Osvik encryption run was 0.05 slower), and every Tiger Lake comparison below holds for all runs at that level. Apart from A.2's slower encryption (0.06), though, the figures of the ternlog sets differ there by 0.01 to 0.03, little more than that spread, and the nearest runs of two sets are less than 0.01 apart, so their order on Tiger Lake is only weakly established.

On Zen 5 the gate count alone made decryption slower than Osvik's S-boxes (1.84 against 1.77 cycles per byte), although it has 58 logic instructions per set of eight S-boxes instead of 92. Lowering the depth at the same gate count gained 13% for encryption and 11% for decryption, and the extra gates of A.2 another 2% and 5%. On Zen 4 and Tiger Lake, where these operations take one cycle, lowering the depth at the same gate count gained 3% and 5% on Zen 4 and 2% and 1% on Tiger Lake. The extra gates of A.2 do not clearly pay off on these two CPUs: on Zen 4 they gain 2% for encryption and cost 1% for decryption, and on Tiger Lake they cost 4% for encryption and 1% for decryption. A.2 is clearly the fastest set only on Zen 5.

Register copies do not decide the ranking. Instructions in libgcrypt's encryption and decryption functions compiled with GCC 15 (one loop iteration runs eight rounds on two interleaved groups of 16 blocks), with the zmm register copies among them in parentheses:

S-boxes	encryption	decryption
Osvik	714 (152)	749 (186)
fewest gates	607 (108)	611 (94)
fewest gates, lowest depth (A.1)	645 (141)	625 (104)
depth-optimised (A.2)	664 (149)	651 (120)

Written with `_mm512_xor_si512` instead of `^`, the same programs compile to 622 (119) and 613 (86) for the fewest-gates set, 643 (140) and 626 (99) for A.1 and 667 (152) and 655 (110) for A.2. Like the timings, these counts depend on the compiler and on how the source is written.

The lowest-depth set has more instructions and more copies than the fewest-gates set without a depth objective, and is still faster on all three CPUs.

The isolated counts of section 3.1 overstate the saving in decryption. Without copies, one loop iteration of Osvik's S-boxes has 562 instructions for encryption and 563 for decryption, and the fewest-gates set 499 and 517. That is 63 and 46 fewer, where the isolated counts predict 60 and 68. In decryption the round-key XOR follows the S-box, and GCC can merge it into a final two-input operation, the effect P models in section 3.3. Compiled alone with the key XOR, the eight inverse S-boxes take 113 logic instructions for Osvik's set (11 of the 32 XORs merged), 86 for the fewest-gates set (4 merged: its native XOR outputs that feed nothing else) and 93 for A.2 (5 merged). That predicts 54 fewer per loop iteration, so the merging accounts for 14 of the 22 instructions missing from the saving; the rest is not explained here. In encryption the key XOR comes before the S-box, and the in-context difference is close to the isolated count. The merging is part of why the fewest-gates set loses to Osvik's in decryption on Zen 5.

How the XORs are written matters to GCC. `_mm512_xor_si512` compiles to an XOR of 32-bit elements (`vpxord`), and GCC does not merge it with the key XOR, a `^` on 64-bit elements; written that way, the fewest-gates and A.1 sets merge none of their key XORs. The `^` versions in the table have fewer instructions but are not faster on Zen 5. An earlier run of the `_mm512_xor_si512` versions measured 1.90 / 1.82 and 1.69 / 1.62 cycles per byte, against 1.94 / 1.84 and 1.69 / 1.64 here. That run used another setup (shared libraries, one set at a time), so its differences of up to 0.04 are suggestive at most, even though Osvik's set and A.2, the same code in both runs, agreed within 0.01.

4. All 4-bit permutation classes

Two S-boxes are affine equivalent if `S'(x) = B·S(A·x ⊕ a) ⊕ b` for invertible `A`, `B`; equivalent S-boxes have the same differential and linear properties, and there are 302 classes of 4-bit permutations (De Cannière, 2007). For representatives I took the 272 given by Ullrich et al. and found the remaining 30 by computing an affine canonical form (lexicographically smallest member, by brute force over the input map with the output map chosen greedily) for random permutations until all 302 classes had appeared.

To find the cheapest member of a class, the SAT instance also chooses `A`, `a` and `B` (the output constant is free in this model): row by row it computes `B·S(A·x ⊕ a)` from the representative's table and requires the program outputs to match. Requiring the result to be a permutation forces `A` and `B` to be invertible. Each solution is checked both by simulation and by an independent affine-equivalence test.

For classes 1-8_5 and 1-8_6 the 6-gate SAT instance did not finish in 12 hours on 16 threads. The work is in finding `A` and `B`, not the program, so I removed them from the search: with `a` = 0 and the columns of `A` sorted (column order only relabels the inputs) there are 840 input maps, and for a fixed `A` the outputs must be 4 linearly independent components `u·S(A·x)`, that is 4 of 15 fixed tables. A depth-first search over programs then tries, at an output gate, only the remaining tables that are a function of 3 available values, and at a hidden gate every function of every 3 available values. A program with k gates and 4 outputs has only k − 4 hidden gates, so the exhaustive 6-gate search takes 4 seconds per class on 32 threads. It agrees with SAT on all 243 other classes without affine components (a program at the class minimum, none with one gate fewer), and finds no 6-gate program for 1-8_5 or 1-8_6, also with its gate-ordering pruning

switched off. With `A` fixed to the identity and each output fixed to the corresponding S-box bit, the same search confirms the Serpent gate counts of section 3.1.

All 302 classes are solved. The cheapest member needs at most 7 gates, and every minimum is proven (one gate fewer is impossible for the whole class). For 97 of the 272 representatives of Ullrich et al., which are the cheapest members of their classes in two-input instructions, the class has a cheaper ternlog member.
Number of classes by two-input cost (Ullrich et al.) and ternlog cost:

two-input cost	0 gates	1 gate	2 gates	3 gates	4 gates	5 gates	6 gates	7 gates	classes
0	1								1
3		1							1
4			1						1
5			3						3
6			1	1					2
7				10					10
8				4	6				10
9					20				20
10					15	13			28
11					1	66			67
12						53	27		80
13						8	41		49
13+ (not found)						5	23	2	30

Classes of the Serpent S-boxes, with the cheapest ternlog member of the class and the cost of the S-box itself (fixed input and output bits):

S-boxes	class	two-input cost	class minimum	these S-boxes
$S0$, $S1^{-1}$	15	10	4	7
$S1$, $S0^{-1}$	14	10	4	7
$S2$, $S6$, $S2^{-1}$, $S6^{-1}$	16	11	5	7
$S3$, $S7$, $S3^{-1}$, $S7^{-1}$	12	13+	6	8
$S4$, $S5$	9	11	5	7
$S4^{-1}$, $S5^{-1}$	10	12	5	7

The Serpent classes agree with Ullrich et al.; S3 and S7, which they list as not found, are in class 12. Every Serpent S-box needs two or three gates more than the cheapest member of its class: the class minimum assumes that the affine maps are absorbed elsewhere, and Serpent's linear layer is fixed.

Appendix B classifies the 4-bit S-boxes of other ciphers: lightweight designs such as PRESENT, GIFT, SKINNY-64, PRINCE, Midori and LBlock, older ciphers and hash functions such as Lucifer, JH and Luffa, Magma and the other GOST 28147 parameter sets, the permutations inside Twofish, the 4-bit mini-boxes of Whirlpool, Khazad and CLEFIA, the 4-bit components of the Streebog/Kuznyechik S-box, and the rows of the DES S-boxes.

Ternlog cost is roughly half the two-input cost. The 30 classes beyond Ullrich et al.'s search include the eight classes 1–8. Classes 1–16 are the optimal ones (maximum differential probability 1/4, maximum linear correlation 1/2), and classes 1–8 also have the fewest differentials and linear approximations that reach those maxima. Class 13, the cheapest optimal class in two-input instructions (9, the class of the Noekeon S-box), needs 4 ternlog gates. Among the optimal classes, the cheapest in ternlog gates (classes 13, 14, 15, 4 gates) have the most differentials and approximations at those maxima, while classes 1–8, with the fewest, need 5–7 gates.

References

- D. A. Osvik, Speeding up Serpent, Third AES Candidate Conference, 2000.
- M. Ullrich, C. De Cannière, S. Indesteege, Ö. Küçük, N. Mouha, B. Preneel, Finding Optimal Bitsliced Implementations of 4×4-bit S-boxes, SKEW 2011.
- K. Stoffelen, Optimizing S-box Implementations for Several Criteria using SAT Solvers, FSE 2016.
- M. Dansarie, sboxgates: A program for finding low gate count implementations of S-boxes, Journal of Open Source Software 6(62), 2946, 2021.
- Y. Sovyn, V. Khoma, I. Opirskyy, V. Kozachok, Minimization of Bitsliced Representation of 4×4 S-Boxes based on Ternary Logic Instruction, CPITS 2023, CEUR Workshop Proceedings 3421, pp. 12–24.
- C. De Cannière, Analysis and Design of Symmetric Encryption Algorithms, PhD thesis, K.U.Leuven, 2007.
- A. Biryukov, L. Perrin, A. Udovenko, Reverse-Engineering the S-Box of Streebog, Kuznyechik and STRIBOBr1, EUROCRYPT 2016.
- J. Lloyd, Speeding up Serpent: SIMD Edition, 2009.
- A. Abel, J. Reineke, uops.info: instruction latency, throughput and port usage tables.
- J. Kivilinna, [PATCH] serpent-avx512-x86: use depth-optimized vpternlogd S-boxes, gcrypt-devel, November 2026.

Appendix A: Serpent S-boxes

Notation: `T(a, b, c; imm)` is the 3-input function with truth table `imm` (hex), where bit `4a + 2b + c` of `imm` is the result for inputs `a, b, c`; this is `vpternlogd` with `a` as the destination operand. `^ & |` are XOR, AND and OR, and `~a & b` is AND-NOT. Inputs are `x0..x3` and outputs `y = (y0, y1, y2, y3)`, bit 0 first.

A.1 Fewest gates

The minimum gate count of each S-box (section 3.1), with the lowest round depth at that count (section 3.3).

```
S0, 7 gates
  t1 = T(x0, x2, x3; 96)
  t2 = T(x0, x1, x3; 8e)
  t3 = T(x0, x1, x3; b8)
  t4 = T(x2, t2, t3; 96)
  t5 = T(t1, t3, t4; 27)
  t6 = T(t1, t2, t3; c6)
  t7 = T(t1, t3, t4; 1b)
  y  = (t7, t5, t6, t4)
```

```
S1, 7 gates
  t1 = T(x0, x1, x3; d8)
  t2 = T(x0, x1, x3; 96)
  t3 = T(x0, x1, x2; 59)
  t4 = T(t1, t2, t3; 36)
  t5 = T(t1, t2, t3; 93)
  t6 = x3 ^ t3
  t7 = T(t1, t2, t3; 2b)
  y  = (t7, t4, t6, t5)
```

```
S2, 7 gates
  t1 = T(x0, x2, x3; 6a)
  t2 = T(x0, x1, x2; 96)
  t3 = T(x0, x2, x3; c6)
  t4 = T(x0, t1, t2; 96)
  t5 = T(t1, t2, t3; 8e)
  t6 = T(t1, t2, t3; b4)
  t7 = T(t2, t3, t6; 8d)
  y  = (t4, t5, t6, t7)
```

```
S3, 8 gates
  t1 = T(x0, x1, x3; 1c)
  t2 = T(x0, x2, x3; 62)
  t3 = T(x0, x1, x2; 1a)
  t4 = T(x1, x2, x3; b0)
  t5 = T(x0, t2, t4; 56)
  t6 = T(x1, t2, t3; 16)
  t7 = T(x3, t1, t3; 9a)
  t8 = T(x3, t1, t2; 96)
  y  = (t5, t8, t7, t6)
```

```
S4, 7 gates
  t1 = T(x0, x1, x3; b8)
  t2 = T(x0, x1, x3; 96)
  t3 = T(x0, x2, x3; 36)
  t4 = T(x1, t1, t3; 69)
  t5 = T(x1, t2, t4; c9)
  t6 = T(t1, t2, t4; b1)
  t7 = T(t2, t3, t6; c6)
  y  = (t4, t7, t6, t5)
```

```
S5, 7 gates
  t1 = T(x0, x1, x3; 96)
  t2 = T(x0, x1, x3; e4)
  t3 = T(x0, x1, x2; 6a)
  t4 = T(x3, t2, t3; 69)
  t5 = T(x3, t1, t4; 36)
  t6 = T(t1, t2, t4; 27)
  t7 = T(t1, t3, t6; c6)
  y  = (t4, t5, t6, t7)
```

```
S6, 7 gates
  t1 = T(x1, x2, x3; 96)
  t2 = T(x0, x2, t1; 36)
  t3 = T(x0, x3, t1; 59)
  t4 = T(x0, x3, t1; 4e)
  t5 = T(x1, t2, t4; c9)
  t6 = T(t1, t2, t4; 4b)
  t7 = T(t4, t5, t6; d2)
  y  = (t6, t3, t5, t7)
```

```
S7, 8 gates
  t1 = T(x0, x1, x3; 1a)
  t2 = T(x1, x2, x3; b6)
  t3 = T(x1, x2, x3; d2)
  t4 = T(x0, x3, t3; 70)
  t5 = T(x0, t2, t3; 39)
  t6 = T(t1, t3, t5; 39)
  t7 = T(x2, t1, t6; 36)
  t8 = T(x2, t2, t4; 46)
  y  = (t5, t7, t8, t6)
```

```
S0^-1, 7 gates
  t1 = T(x0, x1, x3; 68)
  t2 = T(x0, x1, x2; a9)
  t3 = T(x0, x1, x3; 4e)
  t4 = T(x2, t1, t3; 63)
  t5 = x3 ^ t2
  t6 = T(t2, t3, t4; 1b)
  t7 = T(t2, t3, t4; 69)
  y  = (t4, t6, t5, t7)
```

```
S1^-1, 7 gates
  t1 = T(x0, x1, x3; d2)
  t2 = T(x0, x1, x3; e8)
  t3 = T(x0, x1, x3; 4e)
  t4 = x2 ^ t1
  t5 = T(x2, t2, t3; 9a)
  t6 = T(t2, t4, t5; 69)
  t7 = T(t1, t5, t6; 4b)
  y  = (t6, t5, t7, t4)
```

```
S2^-1, 7 gates
  t1 = T(x0, x1, x2; 1e)
  t2 = T(x1, x2, x3; 39)
  t3 = T(x1, x3, t1; 6a)
  t4 = T(x1, x2, x3; 72)
  t5 = T(x0, t2, t4; 9c)
  t6 = T(x0, t4, t5; 69)
  t7 = T(t3, t4, t5; 93)
  y  = (t3, t6, t5, t7)
```

```
S3^-1, 8 gates
  t1 = x1 ^ x2
  t2 = T(x0, x1, x2; b4)
  t3 = T(x3, t1, t2; 36)
  t4 = T(x3, t1, t2; e3)
  t5 = T(x1, t2, t4; 75)
  t6 = T(x1, x3, t2; 96)
  t7 = T(x2, t4, t6; 79)
  t8 = T(t1, t3, t6; a6)
  y  = (t3, t5, t8, t7)
```

```
S4^-1, 7 gates
  t1 = T(x0, x2, x3; 86)
  t2 = T(x0, x1, x2; 56)
  t3 = T(x0, x1, t1; 6a)
  t4 = T(x0, x3, t3; 78)
  t5 = T(x3, t2, t4; 96)
  t6 = T(t2, t3, t4; 87)
  t7 = T(t4, t5, t6; 4b)
  y  = (t6, t3, t7, t5)
```

```
S5^-1, 7 gates
  t1 = T(x0, x2, x3; c6)
  t2 = T(x0, x1, x2; d9)
  t3 = T(x0, x3, t2; 6a)
  t4 = T(x0, x1, t3; 2d)
  t5 = T(x1, t1, t4; 96)
  t6 = T(t1, t3, t4; 4b)
  t7 = T(t4, t5, t6; d2)
  y  = (t5, t6, t7, t3)
```

```
S6^-1, 7 gates
  t1 = T(x0, x2, x3; a6)
  t2 = T(x0, x2, x3; b1)
  t3 = T(x0, x2, x3; 9a)
  t4 = T(x1, t2, t3; c5)
  t5 = T(x1, t2, t3; ac)
  t6 = T(x1, t1, t2; c3)
  t7 = T(t1, t4, t5; 78)
  y  = (t4, t6, t7, t5)
```

```
S7^-1, 8 gates
  t1 = T(x0, x2, x3; 92)
  t2 = T(x0, x1, x3; bc)
  t3 = T(x0, t1, t2; 8e)
  t4 = t1 ^ t2
  t5 = T(x1, x2, x3; a4)
  t6 = T(x0, t1, t5; a9)
  t7 = T(x2, t2, t5; 69)
  t8 = T(t3, t6, t7; 87)
  y  = (t7, t6, t4, t8)
```

A.2 Depth-optimised

One or two extra gates where that lowers the round depth (section 3.3).

```
S0, 8 gates
  t1 = T(x0, x1, x3; b8)
  t2 = T(x0, x1, x3; 36)
  t3 = T(x0, x1, x3; 6c)
  t4 = T(x0, x2, x3; 96)
  t5 = x2 ^ t2
  t6 = T(t1, t2, t4; 6c)
  t7 = T(t1, t4, t5; 1b)
  t8 = T(t1, t3, t4; 4d)
  y  = (t8, t7, t6, t5)
```

```
S1, 7 gates
  t1 = T(x0, x1, x3; d8)
  t2 = T(x0, x1, x3; 96)
  t3 = T(x0, x1, x2; 59)
  t4 = T(t1, t2, t3; 36)
  t5 = T(t1, t2, t3; 93)
  t6 = x3 ^ t3
  t7 = T(t1, t2, t3; 2b)
  y  = (t7, t4, t6, t5)
```

```
S2, 7 gates
  t1 = T(x0, x2, x3; 6a)
  t2 = T(x0, x1, x2; 96)
  t3 = T(x0, x2, x3; c6)
  t4 = T(x0, t1, t2; 96)
  t5 = T(t1, t2, t3; 8e)
  t6 = T(t1, t2, t3; b4)
  t7 = T(t2, t3, t6; 8d)
  y  = (t4, t5, t6, t7)
```

```
S3, 8 gates
  t1 = T(x0, x1, x3; 1c)
  t2 = T(x0, x2, x3; 62)
  t3 = T(x0, x1, x2; 1a)
  t4 = T(x1, x2, x3; b0)
  t5 = T(x0, t2, t4; 56)
  t6 = T(x1, t2, t3; 16)
  t7 = T(x3, t1, t3; 9a)
  t8 = T(x3, t1, t2; 96)
  y  = (t5, t8, t7, t6)
```

```
S4, 8 gates
  t1 = T(x0, x2, x3; 36)
  t2 = T(x0, x1, x3; 96)
  t3 = T(x0, x2, x3; 93)
  t4 = T(x0, x1, x3; 2e)
  t5 = T(t2, t3, t4; 4b)
  t6 = t3 ^ t4
  t7 = T(x1, t2, t6; c9)
  t8 = T(t1, t2, t5; d2)
  y  = (t6, t8, t5, t7)
```

```
S5, 8 gates
  t1 = T(x0, x1, x2; 6e)
  t2 = T(x0, x1, x3; 38)
  t3 = T(x0, x1, x3; 7e)
  t4 = T(x0, x2, t3; 69)
  t5 = T(x3, t1, t3; 87)
  t6 = T(x3, t2, t5; 4d)
  t7 = T(x1, t4, t6; 96)
  t8 = T(x2, x3, t2; a9)
  y  = (t4, t8, t5, t7)
```

```
S6, 8 gates
  t1 = T(x0, x1, x3; 16)
  t2 = T(x0, x1, x3; 62)
  t3 = T(x0, x1, x3; 2e)
  t4 = T(x0, x2, x3; 93)
  t5 = T(x2, t1, t2; a3)
  t6 = T(x2, t1, t3; c9)
  t7 = x1 ^ t4
  t8 = T(t4, t5, t6; 2d)
  y  = (t5, t7, t6, t8)
```

```
S7, 9 gates
  t1 = T(x0, x2, x3; 64)
  t2 = T(x0, x1, x2; 62)
  t3 = T(x0, x2, x3; d2)
  t4 = T(x0, x1, x2; 16)
  t5 = T(x0, t1, t4; 6a)
  t6 = T(x1, t1, t3; 93)
  t7 = T(x3, t2, t4; 3a)
  t8 = T(x3, t2, t4; 26)
  t9 = T(t3, t6, t8; 59)
  y  = (t6, t9, t7, t5)
```

```
S0^-1, 8 gates
  t1 = T(x0, x1, x3; 4e)
  t2 = T(x0, x1, x2; a9)
  t3 = x3 ^ t2
  t4 = T(x0, x1, x2; 63)
  t5 = T(x0, x1, x2; 81)
  t6 = T(x3, t4, t5; c9)
  t7 = T(t1, t2, t6; 27)
  t8 = T(t1, t2, t6; 69)
  y  = (t6, t7, t3, t8)
```

```
S1^-1, 8 gates
  t1 = T(x0, x1, x2; 6a)
  t2 = T(x0, x1, x2; e4)
  t3 = T(x0, x1, x2; 7c)
  t4 = T(x3, t1, t3; 69)
  t5 = T(x0, t2, t4; 8d)
  t6 = T(x1, t1, t4; c5)
  t7 = T(x3, t2, t3; 95)
  t8 = T(x2, t4, t7; 6c)
  y  = (t7, t5, t8, t6)
```

```
S2^-1, 8 gates
  t1 = T(x0, x1, x2; 1e)
  t2 = T(x1, x3, t1; 6a)
  t3 = T(x1, x2, x3; ca)
  t4 = T(x1, x2, x3; 8d)
  t5 = T(x1, x2, x3; 49)
  t6 = T(t1, t4, t5; 56)
  t7 = T(t2, t3, t4; 93)
  t8 = T(t2, t3, t4; 3a)
  y  = (t2, t6, t7, t8)
```

```
S3^-1, 8 gates
  t1 = x1 ^ x2
  t2 = T(x0, x1, x2; b4)
  t3 = T(x3, t1, t2; 36)
  t4 = T(x3, t1, t2; e3)
  t5 = T(x1, t2, t4; 75)
  t6 = T(x1, x3, t2; 96)
  t7 = T(x2, t4, t6; 79)
  t8 = T(t1, t3, t6; a6)
  y  = (t3, t5, t8, t7)
```

```
S4^-1, 8 gates
  t1 = T(x0, x2, x3; 26)
  t2 = T(x0, x1, x3; 6e)
  t3 = T(x0, x1, x2; 3a)
  t4 = T(x0, t1, t2; 6c)
  t5 = T(x0, x1, x2; 96)
  t6 = T(x2, t2, t3; c9)
  t7 = T(x3, t3, t5; 65)
  t8 = T(t1, t3, t7; 69)
  y  = (t7, t4, t6, t8)
```

```
S5^-1, 8 gates
  t1 = T(x0, x1, x2; 5b)
  t2 = T(x0, x1, x3; 38)
  t3 = T(x0, x3, t1; c5)
  t4 = T(x1, x3, t1; ac)
  t5 = T(x1, x2, t3; 59)
  t6 = T(x2, t2, t3; 9c)
  t7 = T(x3, t1, t2; c9)
  t8 = t2 ^ t4
  y  = (t8, t7, t6, t5)
```

```
S6^-1, 9 gates
  t1 = T(x0, x2, x3; d8)
  t2 = T(x0, x1, x2; d0)
  t3 = T(x0, x2, x3; 78)
  t4 = T(x0, x1, x2; 48)
  t5 = T(x0, x1, x2; 39)
  t6 = x3 ^ t5
  t7 = T(x3, t2, t4; c9)
  t8 = T(x1, t3, t7; 93)
  t9 = T(t1, t5, t7; 69)
  y  = (t7, t6, t8, t9)
```

```
S7^-1, 9 gates
  t1 = T(x0, x1, x2; 87)
  t2 = T(x0, x1, x2; 26)
  t3 = T(x1, x3, t2; 76)
  t4 = T(x1, x3, t1; 34)
  t5 = t2 ^ t4
  t6 = T(x0, t3, t4; 49)
  t7 = T(x2, x3, t1; 2e)
  t8 = T(x2, t1, t3; 2a)
  t9 = t4 ^ t7
  y  = (t9, t6, t8, t5)
```

A.3 Fewest gates, first solution

The programs the gate-count search returned first, with no depth objective; the "fewest gates" baseline of section 3.4.

```
S0, 7 gates
  t1 = T(x0, x1, x3; 36)
  t2 = T(x0, x1, x3; 6c)
  t3 = x2 ^ t1
  t4 = T(x0, x1, x3; b8)
  t5 = T(t2, t3, t4; d1)
  t6 = T(t2, t3, t4; 47)
  t7 = T(t1, t5, t6; e1)
  y  = (t5, t6, t7, t3)
```

```
S1, 7 gates
  t1 = T(x0, x1, x3; d8)
  t2 = T(x0, x2, t1; d8)
  t3 = T(x0, x1, x2; 59)
  t4 = T(x1, x3, t2; 69)
  t5 = x3 ^ t3
  t6 = T(t1, t3, t4; 4e)
  t7 = T(t1, t3, t4; 69)
  y  = (t6, t7, t5, t4)
```

```
S2, 7 gates
  t1 = T(x0, x2, x3; 6a)
  t2 = T(x0, x1, t1; 2d)
  t3 = T(x1, x2, t1; 96)
  t4 = T(x2, t1, t2; b4)
  t5 = t2 ^ t3
  t6 = T(t1, t4, t5; e1)
  t7 = t4 ^ t6
  y  = (t3, t7, t6, t5)
```

```
S3, 8 gates
  t1 = T(x1, x2, x3; 79)
  t2 = T(x1, x2, x3; 63)
  t3 = T(x0, x1, t2; 29)
  t4 = T(x0, x3, t2; 9e)
  t5 = t1 ^ t4
  t6 = T(t3, t4, t5; 63)
  t7 = x3 ^ t3
  t8 = T(t2, t5, t7; 87)
  y  = (t5, t6, t7, t8)
```

```
S4, 7 gates
  t1 = T(x0, x2, x3; c6)
  t2 = T(x0, x1, t1; 4b)
  t3 = T(x3, t1, t2; 69)
  t4 = T(x1, t1, t3; 93)
  t5 = T(x3, t1, t3; c2)
  t6 = x1 ^ t5
  t7 = T(t2, t4, t6; 1e)
  y  = (t4, t6, t7, t3)
```

```
S5, 7 gates
  t1 = T(x0, x1, x2; a6)
  t2 = T(x1, x3, t1; 4b)
  t3 = T(x0, t1, t2; 96)
  t4 = T(x0, t1, t2; d3)
  t5 = x3 ^ t4
  t6 = T(x3, t1, t3; 63)
  t7 = T(t2, t5, t6; b4)
  y  = (t6, t3, t7, t5)
```

```
S6, 7 gates
  t1 = T(x0, x1, x3; 96)
  t2 = T(x0, x2, x3; 93)
  t3 = x1 ^ t2
  t4 = T(x0, x3, t3; e4)
  t5 = T(t1, t2, t4; 63)
  t6 = T(t1, t4, t5; b1)
  t7 = T(t1, t4, t5; 8d)
  y  = (t7, t3, t6, t5)
```

```
S7, 8 gates
  t1 = T(x1, x2, x3; a6)
  t2 = x1 ^ x2
  t3 = T(x0, t1, t2; 86)
  t4 = T(x0, t1, t2; 9a)
  t5 = T(x1, t1, t3; 56)
  t6 = x2 ^ t3
  t7 = T(t1, t4, t5; 1d)
  t8 = T(t1, t4, t5; b4)
  y  = (t7, t6, t8, t4)
```

```
S0^-1, 7 gates
  t1 = T(x0, x1, x2; a9)
  t2 = T(x0, x1, x3; 4e)
  t3 = T(x0, x1, x3; 96)
  t4 = T(t1, t2, t3; 59)
  t5 = T(t1, t2, t3; 2b)
  t6 = x3 ^ t1
  t7 = T(t1, t2, t4; 69)
  y  = (t7, t5, t6, t4)
```

```
S1^-1, 7 gates
  t1 = T(x0, x1, x3; e8)
  t2 = T(x0, x1, x3; 4e)
  t3 = T(x0, x1, x3; d2)
  t4 = x2 ^ t3
  t5 = T(x2, t1, t2; 9a)
  t6 = T(t1, t4, t5; 69)
  t7 = T(t3, t5, t6; 4b)
  y  = (t6, t5, t7, t4)
```

```
S2^-1, 7 gates
  t1 = T(x1, x2, x3; 72)
  t2 = x0 ^ x3
  t3 = T(x1, t1, t2; e1)
  t4 = x2 ^ t3
  t5 = T(x0, t1, t4; 69)
  t6 = T(t1, t2, t5; 36)
  t7 = T(t1, t4, t6; 87)
  y  = (t6, t5, t4, t7)
```

```
S3^-1, 8 gates
  t1 = T(x0, x1, x2; b4)
  t2 = T(x1, x2, x3; 96)
  t3 = T(x3, t1, t2; a6)
  t4 = T(x3, t1, t3; c2)
  t5 = T(x1, t1, t4; 9a)
  t6 = x2 ^ t4
  t7 = T(t1, t2, t5; 78)
  t8 = T(t2, t5, t7; b4)
  y  = (t3, t8, t6, t7)
```

```
S4^-1, 7 gates
  t1 = T(x0, x1, x2; 56)
  t2 = T(x0, x3, t1; 46)
  t3 = x1 ^ t2
  t4 = T(x0, x3, t3; 78)
  t5 = T(x3, t1, t4; 96)
  t6 = T(t1, t3, t4; 87)
  t7 = T(t4, t5, t6; 4b)
  y  = (t6, t3, t7, t5)
```

```
S5^-1, 7 gates
  t1 = T(x0, x2, x3; c6)
  t2 = T(x0, x1, t1; d9)
  t3 = x3 ^ t2
  t4 = T(x0, x1, t3; 2d)
  t5 = T(x1, t1, t4; 96)
  t6 = T(t1, t3, t4; 4b)
  t7 = T(t4, t5, t6; d2)
  y  = (t5, t6, t7, t3)
```

```
S6^-1, 7 gates
  t1 = T(x0, x2, x3; 59)
  t2 = T(x0, x2, t1; e4)
  t3 = T(x0, x1, t2; 1e)
  t4 = x1 ^ t1
  t5 = T(x2, t1, t3; 96)
  t6 = T(x1, t2, t5; 96)
  t7 = T(t1, t5, t6; 87)
  y  = (t6, t4, t7, t5)
```

```
S7^-1, 8 gates
  t1 = T(x0, x1, x2; 9c)
  t2 = T(x0, x1, x2; 17)
  t3 = T(x2, x3, t1; 6b)
  t4 = T(x3, t2, t3; 87)
  t5 = x0 ^ t3
  t6 = T(t1, t4, t5; 9e)
  t7 = t2 ^ t6
  t8 = T(t1, t4, t5; 87)
  y  = (t8, t5, t7, t4)
```

Appendix B: S-boxes of other ciphers

4-bit S-boxes of other ciphers, grouped by class, including the 4-bit mini-boxes from which the 8-bit S-boxes of Whirlpool, Khazad and CLEFIA are built. Each table was checked against an independent implementation, or (for CLEFIA) by rebuilding the cipher's 8-bit S-box from its mini-boxes; the class minimum is the cheapest ternlog member of the class (section 4). Sovyn et al. give heuristic ternlog counts for 225 S-boxes. None of them is below the class minimum of its class, and 9 are equal to it.

S-boxes	class	two-input cost	class minimum
SKINNY-64, Noekeon, Piccolo, PRIDE, Pyjamask-128, LBlock S0 (also LAC), LBlock S1, LBlock S2, LBlock S3, LBlock S4, LBlock S5, LBlock S6, LBlock S7, LBlock S8, LBlock S9	13	9	4
Spongent (Elephant)	14	10	4
Streebog/Kuznyechik ν_1	269	9	4

QARMA-64 σ_2	1-8_2	13+	5
KLEIN, QARMA-64 σ_1, Iceberg S0 (also Puffin), Iceberg S1, Whirlpool R, Khazad P, Khazad Q	1-8_3	13+	5
Hummingbird-2 S3	9	11	5
Twofish q0 t2, Hummingbird-2 S4, CLEFIA SS0	10	12	5
PRESENT (also LED, PHOTON), Midori Sb0 (also CRAFT, MANTIS, WARP), KNOT, RECTANGLE, Luffa, CS-Cipher G	16	11	5
GIFT	25	11	5
Streebog/Kuznyechik σ	40	13	5
Twofish q1 t2	43	12	5
Twofish q1 t1	52	11	5
Twofish q1 t0	56	12	5
Twofish q0 t0	63	13	5
PRINCE, Midori Sb1, JH S0, JH S1	1-8_1	13+	6
Hummingbird-2 S2, CLEFIA SS1	11	13	6
Magma π'_0, Magma π'_1, Magma π'_2, Magma π'_3, Magma π'_4, Magma π'_5, Magma π'_6, Magma π'_7, QARMA-64 σ_0, Hummingbird-2 S1, SC2000, CLEFIA SS2, CLEFIA SS3	12	13+	6
Lucifer S1	17	13+	6
Streebog/Kuznyechik ν_0	23-24_1	13+	6
Twofish q0 t1	39	13	6
Twofish q0 t3	72	13	6
Twofish q1 t3	130	13	6
Lucifer S0	162	13	6
Whirlpool E	1-8_5	13+	7
GF(2^4) inversion, MIBS, TWINE, Streebog/Kuznyechik inversion, mCrypton S0, mCrypton S1, mCrypton S2, mCrypton S3	1-8_6	13+	7

The lightweight designs concentrate in a few classes: SKINNY-64, Noekeon, Piccolo, PRIDE, Pyjamask and all ten LBlock S-boxes use members of class 13, the cheapest optimal class, and PRESENT, Midori Sb0, KNOT and RECTANGLE use class 16. The class of GF(2^4) inversion also contains MIBS, TWINE and all four mCrypton S-boxes. Inversion is shown for the polynomial $x^4 + x + 1$; the other two irreducible polynomials (and other bases) give the same class, since changing the representation of the field is a linear map. The inversion itself needs 7 gates. Whirlpool's mini-box E is in class 1-8_5. Class 12, the class of Serpent's S3 and S7, contains all eight Magma S-boxes, and Khazad's P and Q, Whirlpool's R and Iceberg's S-boxes share class 1-8_3 with KLEIN. The 8-bit S-box of Streebog and Kuznyechik has no published

design rationale, but Biryukov, Perrin and Udovenko decomposed it into 4-bit components; the permutations among them are ν_0, ν_1, σ and GF(2^4) inversion (their φ is not a permutation). ν_1, which has a probability-1 differential, is in class 269. The eight 4-bit permutations inside Twofish's q_0 and q_1 are in eight different classes; only q0 t2 is in an optimal class. The other GOST 28147 parameter sets (56 S-boxes) fall into 34 classes, 27 of the S-boxes in optimal classes, with class minimums of 4–6 gates.

DES S-boxes map 6 bits to 4, but each of their four rows is a 4-bit permutation. Class of each row, with its class minimum in parentheses:

	row 0	row 1	row 2	row 3
S1	207 (5)	207 (5)	207 (5)	148 (4)
S2	109 (5)	153 (5)	142 (4)	196 (6)
S3	190 (5)	210 (5)	142 (4)	215 (4)
S4	196 (6)	196 (6)	196 (6)	196 (6)
S5	190 (5)	170 (5)	92 (6)	163 (5)
S6	38 (5)	180 (6)	195 (6)	109 (5)
S7	210 (5)	145 (5)	196 (6)	218 (4)
S8	81 (5)	238 (5)	169 (5)	208 (5)

All four rows of S4 are in the same class, which reflects S4's known redundancy. These are only the classes of the rows: a bitsliced DES S-box computes the 6-bit function as a whole.

Appendix C: cheapest ternlog member of every class

Same notation as appendix A, one program per row. The canonical form identifies the class: it is the lexicographically smallest member, so the class of any 4-bit permutation can be looked up by computing its canonical form. The member column is the S-box the program computes; both are given as `S(0)..S(15)` in hex. Class numbers follow Ullrich et al. Their numbering cannot tell apart classes with identical differential and linear histograms, so the 30 classes they did not find are labelled by range and an arbitrary index: 1-8_3 is the third of classes 1–8.

class	canonical form	2-input cost	gates	member	program

1-8_1	`0123468b59ceda7f`	13+	6	`019cf28ae65d7b43`	`t1=T(x0,x1,x2;e6);`
					`t2=T(x0,x3,t1;9a);`
					`t3=T(x2,x3,t1;a4);`
					`t4=T(x0,t1,t3;68);`
					`t5=T(x0,x3,t4;ae);`
					`t6=T(x1,x3,t4;b6); y = (t2, t3, t5, t6)`

1-8_2	0123468b5c9de7af	13+	5	0819ef2bda653c74	t1=T(x0,x2,x3;4a);
					t2=T(x1,x2,t1;9a);
					t3=T(x2,x3,t2;b4);
					t4=T(x1,x3,t3;c6);
					t5=T(x0,x1,t4;72); y = (t2, t3,
					t4, t5)

1-8_3	0123468b5c9dea7f	13+	5	0132ce564abd879f	t1=T(x1,x2,x3;a6);
					t2=T(x0,x1,t1;9c);
					t3=T(x0,x2,t2;d2);
					t4=T(x2,x3,t3;b4);
					t5=T(x1,x3,t4;c6); y = (t2, t3,
					t4, t5)

1-8_4	0123468b5cd79fae	13+	6	0fa489e13725bdc6	t1=T(x0,x1,x2;94);
					t2=T(x1,x3,t1;9e);
					t3=T(x0,x1,t2;72);
					t4=T(x0,t1,t2;8e);
					t5=T(x0,x3,t2;e6);
					t6=T(x2,x3,t1;52); y = (t3, t4,
					t5, t6)

1-8_5	0123468b5cd7af9e	13+	7	0368719b452dcfea	t1=T(x0,x1,x3;6e);
					t2=T(x0,x2,x3;94);
					t3=T(x0,x1,t2;7a);
					t4=T(x1,t1,t2;6a);
					t5=T(x2,t1,t2;24);
					t6=T(x3,t4,t5;78);
					t7=T(t1,t5,t6;36); y = (t3, t4,
					t6, t7)

1-8_6	0123468b5cd7f9ea	13+	7	0e671cfb392a85d4	t1=T(x0,x1,x3;92);
					t2=T(x0,x2,t1;56);
					t3=T(x1,x2,t1;3e);
					t4=T(x2,x3,t3;1a);
					t5=T(x3,t2,t3;56);
					t6=T(x0,t2,t5;6a);
					t7=T(t1,t2,t3;24); y = (t2, t4,
					t6, t7)

1-8_7	0123468c59bdea7f	13+	6	01a9276843dfec5b	t1=T(x0,x1,x3;68);
					t2=T(x0,x2,t1;7a);
					t3=T(x1,x2,t1;56);
					t4=T(x1,x2,t2;34);
					t5=T(x3,t3,t4;b4);
					t6=T(x2,t4,t5;9c); y = (t2, t3,
					t5, t6)

1-8_8	0123468c59dfa7be	13+	6	03a587b2169d4ecf	t1=T(x0,x1,x3;9e);
					t2=T(x2,x3,t1;d6);
					t3=T(x0,t1,t2;d4);
					t4=T(x0,x2,t2;e6);
					t5=T(x0,x3,t2;d8);
					t6=T(x1,x2,t1;a4); y = (t3, t4,
					t5, t6)

9	0123468b59cf7dae	11	5	05cd6129a7fe8b34	t1=T(x0,x1,x3;78);
					t2=T(x2,x3,t1;9c);
					t3=T(x1,x3,t2;b4);
					t4=T(x2,t1,t3;1e);
					t5=T(t2,t3,t4;9c); y = (t1, t2,
					t4, t5)

10	0123468a5bcf7d9e	12	5	082da6fc195e7b34	t1=T(x0,x2,x3;b4);
					t2=T(x1,x3,t1;6c);
					t3=T(x1,x2,t2;9c);
					t4=T(x1,t1,t3;d2);
					t5=T(x2,t1,t3;9c); y = (t2, t3,
					t4, t5)

11	0123468c59bde7af	13	6	0b7d6f8a932145ec	t1=T(x1,x2,x3;d2); t2=T(x0,x2,t1;d6); t3=T(x0,t1,t2;68); t4=x3^t2; t5=T(x1,x2,x3;9c); t6=T(t1,t3,t4;78); y = (t3, t4, t5, t6)
12	0123468b59ced7af	13+	6	0eb83c42df9157a6	t1=T(x1,x2,x3;94); t2=T(x0,x3,t1;46); t3=T(x0,x1,x3;7a); t4=T(x0,t1,t3;6c); t5=T(x1,x2,t3;4a); t6=T(x2,t1,t3;8e); y = (t2, t4, t5, t6)
13	0123468a5bcfde79	9	4	07ce5adf4182b639	t1=T(x0,x1,x2;9a); t2=T(x0,x3,t1;78); t3=T(x1,x3,t1;36); t4=T(x1,t2,t3;b4); y = (t1, t2, t3, t4)
14	0123468a5bcf7e9d	10	4	0987b216dc34fe5a	t1=T(x0,x2,x3;1e); t2=T(x1,x2,t1;6c); t3=T(x1,x3,t2;6c); t4=T(x1,t1,t3;1e); y = (t1, t2, t3, t4)
15	0123468a5bcf79de	10	4	01ba39ce6574fd28	t1=T(x0,x1,x2;36); t2=T(x0,x3,t1;1e); t3=T(x2,x3,t1;9c); t4=T(x2,t2,t3;b4); y = (t1, t2, t3, t4)
16	0123468a5bcfed97	11	5	048c579bd3e26a1f	t1=T(x1,x2,x3;c6); t2=T(x0,x3,t1;94); t3=T(x0,x3,t1;e4); t4=T(x1,t1,t2;a6); t5=T(x1,x3,t2;b4); y = (t1, t3, t4, t5)
17	0123458a6cbde97f	13+	6	013275dc96f8a4be	t1=T(x0,x2,x3;a6); t2=T(x0,x1,t1;1e); t3=T(x2,t1,t2;64); t4=T(x1,x2,t3;96); t5=T(x2,t1,t4;d8); t6=T(x1,x3,t5;e4); y = (t2, t4, t5, t6)
18	0123458a6ce9bdf7	12	5	013c4da96782ef5b	t1=T(x0,x2,x3;c6); t2=T(x0,x1,t1;b4); t3=T(x0,x1,x3;a6); t4=T(x1,t1,t3;1e); t5=T(t1,t2,t4;d2); y = (t2, t3, t4, t5)
19	0123458a6bcf7d9e	12	5	04132ef5ca9d7b68	t1=T(x1,x2,x3;78); t2=T(x0,x3,t1;94); t3=T(x0,x2,t2;9c); t4=T(x2,t1,t2;6a); t5=T(x3,t3,t4;78); y = (t1, t3, t4, t5)
20	0123458a6cbde79f	12	5	018b2dc5f69a43e7	t1=T(x0,x2,x3;d2); t2=T(x1,x2,t1;24); t3=x3^t2; t4=T(x1,x2,t3;c6); t5=T(x1,t1,t4;78); y = (t1, t3, t4, t5)

21	`0123458a6bcf9d7e`	13	6	`051d89ea674f23bc`	`t1=T(x0,x2,x3;36); t2=T(x0,x1,t1;b4); t3=T(x1,x3,t2;9c); t4=T(x1,x3,t2;4e); t5=T(x1,t1,t4;e8); t6=T(t1,t3,t4;56); y = (t2, t3, t5, t6)`
22	`0123458a6bced79f`	13+	5	`02ad78136c4fb95e`	`t1=T(x1,x2,x3;b4); t2=T(x0,x2,t1;6c); t3=T(x0,x3,t1;96); t4=T(x3,t1,t2;d8); t5=T(x0,t3,t4;b4); y = (t2, t3, t4, t5)`
23-24_1	`0123458a6c7fdb9e`	13+	6	`042631aefc985db7`	`t1=T(x1,x2,x3;26); t2=T(x0,x3,t1;6e); t3=T(x0,x1,t1;46); t4=T(x0,x1,x3;92); t5=T(x0,t1,t4;7a); t6=T(x2,t1,t4;1c); y = (t2, t3, t5, t6)`
23-24_2	`0123458a6c7feb9d`	13+	6	`0c84532a9dfe16b7`	`t1=T(x1,x2,x3;2c); t2=T(x0,x3,t1;6e); t3=T(x0,x2,t1;e6); t4=T(x0,x2,x3;b6); t5=T(x0,t1,t4;a8); t6=T(x1,t1,t4;92); y = (t2, t3, t5, t6)`
25	`0123458a6c9dbf7e`	11	5	`068e59c24a73d1fb`	`t1=T(x1,x2,x3;ac); t2=T(x0,x1,t1;d8); t3=T(x0,x2,x3;1e); t4=T(x2,t2,t3;d2); t5=T(x1,x3,t4;96); y = (t1, t2, t3, t5)`
26	`0123458a6cbf7e9d`	11	5	`0b8f9e74d21a563c`	`t1=T(x0,x2,x3;1e); t2=T(x0,x1,x2;78); t3=T(x1,x2,x3;96); t4=T(x2,t1,t3;78); t5=T(t2,t3,t4;9c); y = (t1, t2, t4, t5)`
27	`0123458a6bcf97de`	12	5	`0f152b39ca8d6e47`	`t1=T(x0,x1,x3;d4); t2=T(x0,x1,x3;78); t3=T(x1,x2,t2;4e); t4=T(x2,x3,t2;c6); t5=T(t2,t3,t4;d2); y = (t1, t3, t4, t5)`
28	`0123458a6cbfe7d9`	12	5	`0384fdb69c17ea25`	`t1=T(x0,x2,x3;a6); t2=T(x0,x1,t1;9a); t3=T(x0,x2,t2;6c); t4=T(x0,x1,t3;d2); t5=T(x1,x3,t4;1e); y = (t2, t3, t4, t5)`
29	`0123458a6bcfde97`	12	5	`028f4ce9da1b3657`	`t1=T(x0,x1,x3;56); t2=T(x0,x1,t1;c6); t3=T(x0,x2,t1;78); t4=T(x2,t1,t2;78); t5=T(t1,t3,t4;b4); y = (t2, t3, t4, t5)`

30	`0123458a69ceb7fd`	11	5	`0ab7f2c3e891d465`	`t1=T(x0,x1,x2;c6); t2=T(x0,x2,t1;1e); t3=T(x0,x3,t1;36); t4=T(x2,t2,t3;d2); t5=T(x3,t2,t3;6c); y = (t1, t3, t4, t5)`
31	`0123458a6c7db9fe`	11	5	`026c7e39f5a84db1`	`t1=T(x0,x2,x3;6a); t2=T(x1,x2,t1;c6); t3=T(x0,x1,t2;1e); t4=T(x1,x2,x3;1e); t5=T(t1,t3,t4;d2); y = (t2, t3, t4, t5)`
32	`012345896aceb7fd`	9	4	`08fdb94c7135e2a6`	`t1=T(x1,x2,x3;36); t2=T(x0,x3,t1;4e); t3=T(x0,x1,x3;c6); t4=T(x0,x3,t1;36); y = (t1, t2, t3, t4)`
33	`0123458a67ceb9fd`	9	4	`08193f5d6ae24cb7`	`t1=T(x1,x2,x3;d4); t2=T(x1,x2,x3;a6); t3=T(x0,x2,t2;c6); t4=T(x0,x1,t2;78); y = (t1, t2, t3, t4)`
34-36_1	`0123467859cfbead`	13+	6	`021f8b7ae95346dc`	`t1=T(x1,x2,x3;c6); t2=T(x0,x1,t1;6c); t3=T(x1,x3,t2;6c); t4=T(x0,t1,t3;78); t5=T(t2,t3,t4;9c); t6=T(x3,t1,t5;c6); y = (t2, t4, t5, t6)`
34-36_2	`0123467859cfebda`	13+	6	`06b982ed7541c3af`	`t1=T(x0,x1,x3;16); t2=T(x0,x2,t1;52); t3=T(x2,x3,t1;6a); t4=T(x1,x2,t3;38); t5=T(x0,t1,t4;56); t6=T(x2,x3,t4;52); y = (t2, t3, t5, t6)`
34-36_3	`0123468b5c9dafe7`	13+	5	`04f795c832e1bad6`	`t1=T(x1,x2,x3;9e); t2=T(x0,x3,t1;6a); t3=T(x1,x3,t2;6c); t4=T(x0,x1,t3;9c); t5=T(x0,x2,t4;4e); y = (t2, t3, t4, t5)`
37	`0123467859cfeadb`	11	5	`0d25fe74ab69c138`	`t1=T(x1,x2,x3;c6); t2=T(x0,x2,t1;78); t3=T(x1,t1,t2;9c); t4=T(x3,t1,t2;1e); t5=T(x1,t3,t4;1e); y = (t2, t3, t4, t5)`
38	`0123468a59cfde7b`	12	5	`0da3fec4672b8519`	`t1=T(x0,x1,x3;a6); t2=T(x0,x2,t1;b4); t3=T(x1,x2,t1;36); t4=T(x1,t2,t3;1e); t5=T(x1,t1,t4;c6); y = (t2, t3, t4, t5)`
39	`0123458a69c7bfde`	13	6	`02459bae7fc6318d`	`t1=T(x0,x2,x3;64); t2=T(x0,x1,x3;e2); t3=T(x1,t1,t2;2e); t4=T(x0,t1,t3;d6); t5=T(x1,x2,t2;b2); t6=T(x1,x3,t1;6a); y = (t3, t4, t5, t6)`

40	`0123458a69cf7ebd`	13	5	`0e8d95123fac7b64`	`t1=T(x0,x1,x2;6a); t2=T(x1,x3,t1;2e); t3=T(x0,x3,t2;9c); t4=T(x0,x2,t3;78); t5=T(x1,x2,t4;36); y = (t2, t3, t4, t5)`
41-42_1	`0123458a6bced97f`	13+	6	`0c3df485791b2ea6`	`t1=T(x0,x1,x3;d2); t2=T(x0,x2,x3;26); t3=T(x1,x3,t2;9a); t4=T(x3,t1,t3;d2); t5=T(x0,t2,t4;38); t6=T(x0,x2,t1;74); y = (t3, t4, t5, t6)`
41-42_2	`0123458a6c9d7feb`	13+	6	`02a8137d45b6cfe9`	`t1=T(x0,x2,x3;d8); t2=T(x1,x2,x3;ec); t3=T(x0,t1,t2;92); t4=T(x0,x1,t1;6c); t5=T(x1,x3,t3;6c); t6=T(x1,t1,t5;d8); y = (t3, t4, t5, t6)`
43	`0123458a6c9dbfe7`	12	5	`0a1b654739ec2df8`	`t1=T(x1,x2,x3;5e); t2=T(x0,x2,t1;a6); t3=T(x0,x1,t1;96); t4=T(x2,x3,t2;b4); t5=T(x0,x3,t4;d8); y = (t2, t3, t4, t5)`
44	`0123458a6ceb7fd9`	12	5	`0a9765b4128edfc3`	`t1=T(x0,x1,x3;b6); t2=T(x0,x2,t1;9a); t3=T(x0,x2,x3;b4); t4=T(x1,x2,t3;6c); t5=T(x3,t1,t4;b4); y = (t2, t3, t4, t5)`
45	`0123458a6bc7de9f`	12	6	`0645a8bf9d2ec137`	`t1=T(x0,x1,x2;56); t2=T(x0,x3,t1;9e); t3=T(x1,x2,t2;d6); t4=T(x1,x3,t1;d2); t5=T(x0,t1,t4;e4); t6=T(x0,x1,t2;a6); y = (t3, t4, t5, t6)`
46	`0123458a69bc7fde`	13	6	`02efb7386c1da495`	`t1=T(x0,x1,x3;b4); t2=T(x0,x2,t1;c6); t3=T(x1,x3,t2;d2); t4=T(x3,t1,t3;1e); t5=T(t1,t2,t4;d2); t6=T(x0,t2,t5;6c); y = (t3, t4, t5, t6)`
47-50_1	`0123458a69cebfd7`	13+	6	`0f54b8dc19372ea6`	`t1=T(x1,x2,x3;dc); t2=T(x0,x3,t1;56); t3=T(x1,x3,t2;c6); t4=T(x0,x1,t3;e4); t5=T(x1,t2,t3;6c); t6=T(x0,x2,t5;68); y = (t2, t3, t4, t6)`
47-50_2	`0123458a69cefb7d`	13+	6	`08729a61ebf35d4c`	`t1=T(x1,x2,x3;9a); t2=T(x0,x2,t1;5c); t3=T(x0,x1,t2;96); t4=T(x2,t1,t3;1e); t5=T(x0,x3,t4;4e); t6=T(x1,x3,t2;46); y = (t3, t4, t5, t6)`

47-50_3	0123458a69cfd7be	13+	6	021364de5bca7f89	t1=T(x0,x1,x2;bc);
					t2=T(x0,x3,t1;96);
					t3=T(x0,x1,t2;6c);
					t4=T(x0,x2,t3;b4);
					t5=T(x2,x3,t4;b4);
					t6=T(x3,t3,t5;d8); y = (t2, t4, t5, t6)

47-50_4	0123458a69cfe7bd	13+	6	0981b7ea6d25cf34	t1=T(x0,x1,x3;36);
					t2=T(x0,x2,t1;b4);
					t3=T(x2,x3,t2;b4);
					t4=T(x3,t1,t3;d8);
					t5=T(t2,t3,t4;e6);
					t6=T(x3,t1,t5;86); y = (t2, t3, t4, t6)

51	0123458a6c7dbef9	11	5	0afd3689ecb17425	t1=T(x0,x1,x2;c6);
					t2=T(x0,x3,t1;1e);
					t3=T(x2,x3,t1;7c);
					t4=T(x1,t1,t3;46);
					t5=T(x2,t2,t4;1e); y = (t1, t2, t4, t5)

52	0123458a6c9feb7d	11	5	0cd162735a49fe8b	t1=T(x0,x1,x3;c6);
					t2=T(x0,x3,t1;b4);
					t3=T(x0,x2,t2;d4);
					t4=T(x2,t1,t2;56);
					t5=T(x2,x3,t4;c6); y = (t1, t3, t4, t5)

53	0123458a6b7c9efd	12	5	063dc25f8b9a417e	t1=T(x0,x1,x3;6c);
					t2=T(x2,x3,t1;6c);
					t3=T(x0,t1,t2;b4);
					t4=T(x0,x2,t2;9c);
					t5=T(t2,t3,t4;d2); y = (t1, t3, t4, t5)

54	012345786acf9bed	12	5	0c26d9be3f5784a1	t1=T(x0,x1,x2;6a);
					t2=T(x2,x3,t1;6c);
					t3=T(x1,t1,t2;d2);
					t4=T(x0,t2,t3;b4);
					t5=T(x0,t1,t4;6c); y = (t2, t3, t4, t5)

55	012345786acfed9b	12	5	06e94acf57d813b2	t1=T(x0,x1,x3;6a);
					t2=T(x2,x3,t1;78);
					t3=T(x0,x1,t2;b4);
					t4=T(t1,t2,t3;c6);
					t5=T(x1,t2,t4;b4); y = (t1, t3, t4, t5)

56	0123458a6b9cedf7	12	5	08d7312a95c6fbe4	t1=T(x1,x2,x3;1e);
					t2=T(x0,x3,t1;6c);
					t3=T(x0,x2,t2;6c);
					t4=T(t1,t2,t3;9c);
					t5=T(x0,t2,t4;1e); y = (t1, t3, t4, t5)

57	0123458a6bcf7e9d	12	5	018af6b723e54dc9	t1=T(x0,x1,x2;6a);
					t2=T(x0,x3,t1;d2);
					t3=T(x2,x3,t1;36);
					t4=T(x1,t1,t3;6c);
					t5=T(x1,t2,t4;78); y = (t2, t3, t4, t5)

58	`0123458a6c9fdb7e`	12	5	`042a1b3d7fe6c958`	`t1=T(x0,x1,x2;c6); t2=T(x2,x3,t1;74); t3=T(x2,x3,t1;1e); t4=T(x0,x3,t3;9c); t5=T(x0,t2,t4;d2); y = (t2, t3, t4, t5)`
59	`0123458a6cb9f7ed`	12	5	`08b72691c4f5d3ea`	`t1=T(x1,x2,x3;78); t2=T(x0,x3,t1;84); t3=T(x2,t1,t2;b4); t4=T(x0,x3,t3;6c); t5=T(x0,t1,t4;1e); y = (t1, t3, t4, t5)`
60	`0123458a6bcfd9e7`	12	5	`0fba9c58e7436d21`	`t1=T(x1,x2,x3;56); t2=T(x0,x3,t1;d2); t3=T(x2,t1,t2;1e); t4=T(x0,x1,t3;36); t5=T(t1,t2,t4;78); y = (t2, t3, t4, t5)`
61	`0123458a69cfdeb7`	12	6	`0c2ea413f9d7b586`	`t1=T(x1,x2,x3;6a); t2=T(x0,x2,t1;c2); t3=T(x1,x2,t2;96); t4=T(x1,t1,t2;e8); t5=T(x0,x3,t4;98); t6=T(x0,x2,t4;16); y = (t1, t3, t5, t6)`
62	`0123458a69cfb7de`	12	6	`0462ec7591f3a8db`	`t1=T(x1,x2,x3;e2); t2=T(x1,x2,x3;b0); t3=T(x0,x2,t2;a6); t4=T(x0,x1,t3;92); t5=T(x1,x3,t4;52); t6=T(x2,x3,t4;e4); y = (t1, t3, t5, t6)`
63	`0123458a6c7fb9de`	13	5	`018be7a9c36d52f4`	`t1=T(x0,x1,x3;98); t2=T(x0,x2,x3;78); t3=T(x2,t1,t2;96); t4=T(x1,x3,t3;c6); t5=T(x1,t2,t4;d2); y = (t2, t3, t4, t5)`
64	`0123458a69bc7efd`	13	6	`08691b3fc54de72a`	`t1=T(x0,x2,x3;36); t2=T(x1,x2,x3;b2); t3=T(x0,t1,t2;94); t4=T(x1,x2,t1;d4); t5=T(x1,x3,t1;9c); t6=T(x0,x1,t5;d2); y = (t3, t4, t5, t6)`
65	`0123458a6bc9de7f`	13	6	`05a87462fcb93ed1`	`t1=T(x1,x2,x3;5e); t2=T(x0,x1,t1;56); t3=T(x1,x2,x3;36); t4=T(x0,t1,t3;4e); t5=T(x0,x1,t3;36); t6=T(x3,t3,t5;e4); y = (t2, t4, t5, t6)`
66	`0123458a6bcfd79e`	13	6	`0e9b4682ac5d13f7`	`t1=T(x1,x2,x3;b8); t2=T(x0,x2,t1;74); t3=T(x0,x1,x2;a6); t4=T(x0,x3,t3;b4); t5=T(x1,x3,t2;ca); t6=T(x3,t2,t3;56); y = (t1, t4, t5, t6)`

67	`0123458a6b7cedf9`	13	6	`038942571ebcdfa6`	`t1=T(x1,x2,x3;6a); t2=T(x0,x2,t1;9a); t3=T(x0,x3,t1;9c); t4=T(x0,x1,t3;68); t5=T(x2,t2,t3;d2); t6=T(x1,t3,t5;d8); y = (t2, t4, t5, t6)`
68	`0123458a6c7fbe9d`	13	6	`0589c7d3abfe6421`	`t1=T(x0,x2,x3;26); t2=T(x0,x1,t1;98); t3=T(x0,x3,t2;7a); t4=T(x0,x2,x3;6a); t5=T(x1,x2,t2;ae); t6=T(x1,x2,t1;ba); y = (t3, t4, t5, t6)`
69	`0123457869cebfda`	13	6	`0da9c4f82567e1b3`	`t1=T(x0,x1,x3;5c); t2=T(x0,x2,t1;78); t3=T(x0,x1,x3;8e); t4=T(x0,x2,x3;76); t5=T(x1,t3,t4;86); t6=T(t1,t2,t5;d2); y = (t2, t3, t5, t6)`
70	`0123458a6b7c9fde`	13	5	`0ce23865bd9f41a7`	`t1=T(x0,x1,x3;b4); t2=T(x2,x3,t1;9c); t3=T(x1,t1,t2;d2); t4=T(x2,t1,t3;9c); t5=T(x2,t2,t4;1e); y = (t2, t3, t4, t5)`
71	`0123458a6c7dbfe9`	13	5	`0b891ad46ec2f735`	`t1=T(x0,x2,x3;9c); t2=T(x0,x1,t1;b4); t3=T(x1,x3,t2;4e); t4=T(x1,x3,t2;9c); t5=T(t1,t2,t4;6c); y = (t1, t3, t4, t5)`
72	`0123458a6c7ebdf9`	13	6	`08d4c2a17e95f6b3`	`t1=T(x0,x2,x3;b4); t2=T(x1,x3,t1;1e); t3=T(x1,x3,t1;d4); t4=T(x2,x3,t2;d8); t5=T(x0,t2,t4;9c); t6=T(x0,x1,x2;1e); y = (t3, t4, t5, t6)`
73	`0123458a6c7ebfd9`	13	6	`0ca456b187293fde`	`t1=T(x1,x2,x3;56); t2=T(x0,x1,t1;6c); t3=T(x0,x2,t2;6c); t4=T(t1,t2,t3;c6); t5=T(x0,t3,t4;b4); t6=T(x0,t1,t5;36); y = (t3, t4, t5, t6)`
74-76_1	`0123458a69ced7fb`	13+	6	`0246df53ac791e8b`	`t1=T(x0,x1,x2;6c); t2=T(x2,x3,t1;78); t3=T(x1,x3,t2;c6); t4=T(x0,x2,t3;d2); t5=T(t1,t3,t4;b4); t6=T(x1,t3,t5;4c); y = (t2, t4, t5, t6)`
74-76_2	`0123458a6b7cdfe9`	13+	6	`049bf5e6372c1d8a`	`t1=T(x0,x2,x3;36); t2=T(x1,x2,x3;1e); t3=T(x0,t1,t2;9c); t4=T(x0,x2,x3;6a); t5=T(x0,t3,t4;b4); t6=T(t2,t3,t4;56); y = (t2, t3, t5, t6)`

74-76_3	`0123458a6c7fbd9e`	13+	6	`09f234a1876bc5ed`	`t1=T(x1,x2,x3;36); t2=T(x0,x3,t1;d2); t3=T(x1,t1,t2;d8); t4=T(x1,t1,t2;4a); t5=T(x3,t1,t4;b4); t6=T(x3,t3,t4;56); y = (t2, t3, t5, t6)`
77	`0123458a69c7befd`	13	5	`0a7824651dc93bfe`	`t1=T(x0,x1,x2;b4); t2=T(x1,x3,t1;6c); t3=T(x2,x3,t1;d2); t4=T(x1,t1,t3;e4); t5=T(x0,x3,t4;d8); y = (t2, t3, t4, t5)`
78	`0123458a69ce7fdb`	13+	5	`07e123d6ab89f4c5`	`t1=T(x1,x2,x3;a6); t2=T(x0,x2,t1;b4); t3=T(x1,t1,t2;1e); t4=T(x2,t1,t3;72); t5=T(x0,x3,t4;4e); y = (t2, t3, t4, t5)`
79	`012345786acfb9ed`	11	5	`02463d719b8eca5f`	`t1=T(x0,x1,x3;c6); t2=T(x1,x2,x3;c6); t3=T(x0,x3,t2;d2); t4=T(t1,t2,t3;b4); t5=T(x3,t1,t4;d2); y = (t2, t3, t4, t5)`
80	`012345786aceb9df`	11	5	`023579861b4adfce`	`t1=T(x1,x2,x3;1e); t2=T(x0,x3,t1;d2); t3=T(x1,x2,t2;98); t4=T(x2,x3,t3;ea); t5=T(x2,x3,t2;d8); y = (t1, t2, t4, t5)`
81	`0123458a6bcf97ed`	11	5	`0a397f5d1bec2648`	`t1=T(x1,x2,x3;56); t2=T(x0,x2,t1;9c); t3=T(x0,x1,t2;36); t4=T(t1,t2,t3;c6); t5=T(x0,t2,t4;d2); y = (t1, t3, t4, t5)`
82	`0123458a6bcedf97`	11	5	`09a7dc81bf362e54`	`t1=T(x0,x1,x3;b6); t2=T(x1,x2,t1;56); t3=T(x1,x2,x3;3a); t4=T(x0,t2,t3;b4); t5=T(x0,x1,t2;36); y = (t2, t3, t4, t5)`
83	`0123458a6bce79fd`	12	5	`0bc73afe456d2189`	`t1=T(x0,x1,x3;58); t2=T(x0,x2,x3;b4); t3=T(x0,x2,t1;a6); t4=T(x1,x2,x3;72); t5=T(x1,x2,t1;da); y = (t2, t3, t4, t5)`
84	`012345687aceb9fd`	10	5	`095c2b6e7da1834f`	`t1=T(x0,x1,x3;b6); t2=T(x0,x2,t1;a2); t3=T(x0,x2,x3;c6); t4=T(x1,x2,x3;d2); t5=T(x0,x3,t4;b4); y = (t2, t3, t4, t5)`

85	`0123458a69cde7fb`	12	6	`0e71465f3b892cda`	`t1=T(x0,x1,x2;6a); t2=T(x0,x3,t1;d2); t3=T(x1,x3,t1;b4); t4=T(t1,t2,t3;1e); t5=T(x2,x3,t4;72); t6=T(x3,t2,t4;d8); y = (t3, t4, t5, t6)`
86	`012345786acde9bf`	13	5	`029b71685fc43dea`	`t1=T(x1,x2,x3;1e); t2=T(x0,x3,t1;b4); t3=T(x0,x2,t2;6c); t4=T(x3,t2,t3;d2); t5=T(x1,t2,t4;d8); y = (t1, t3, t4, t5)`
87	`0123458a69bcf7ed`	13	6	`0f5c8a6b13249ed7`	`t1=T(x0,x1,x2;94); t2=T(x0,x3,t1;a6); t3=T(x0,x1,t2;b4); t4=T(x3,t2,t3;96); t5=T(x1,t3,t4;b4); t6=T(x2,t4,t5;72); y = (t2, t3, t5, t6)`
88	`0123458a697cbfde`	13	6	`0142da3b86f957ce`	`t1=T(x0,x1,x3;36); t2=T(x1,x2,t1;6c); t3=T(x3,t1,t2;96); t4=T(x0,t2,t3;d2); t5=T(x3,t2,t4;e4); t6=T(t1,t3,t5;1e); y = (t3, t4, t5, t6)`
89	`0123458a697cbefd`	13	6	`04c91b3ae65df728`	`t1=T(x0,x1,x3;36); t2=T(x0,x2,x3;a6); t3=T(x1,x2,t1;9c); t4=T(x2,t1,t2;d8); t5=T(x3,t1,t3;e4); t6=T(x0,t1,t5;78); y = (t3, t4, t5, t6)`
90	`0123458a69c7debf`	13	6	`0b3c12458a96dfe7`	`t1=T(x0,x1,x3;d0); t2=T(x1,x2,t1;96); t3=T(x2,x3,t1;36); t4=T(x0,x1,t3;74); t5=T(x1,x3,t3;64); t6=T(x2,x3,t1;46); y = (t2, t4, t5, t6)`
91	`0123458a69cb7efd`	13	6	`083af927641bd5ce`	`t1=T(x0,x1,x3;9c); t2=T(x1,x2,t1;6c); t3=T(x0,t1,t2;3e); t4=x3^t3; t5=T(x3,t1,t2;72); t6=T(x0,t3,t5;78); y = (t2, t4, t5, t6)`
92	`0123457869ceafdb`	13+	6	`0581adc9b36742fe`	`t1=T(x0,x1,x2;56); t2=T(x0,x3,t1;b4); t3=T(x2,x3,t1;6c); t4=T(t1,t2,t3;a4); t5=T(x1,t1,t4;96); t6=T(t1,t2,t5;1e); y = (t2, t3, t5, t6)`
93	`0123458a6cbd7e9f`	11	5	`07fc6b3e5498da12`	`t1=T(x0,x1,x3;1e); t2=T(x2,x3,t1;72); t3=T(x2,x3,t1;ec); t4=T(x1,t2,t3;56); t5=T(x1,x2,t3;78); y = (t1, t2, t4, t5)`

94	0123458a6c9fdbe7	11	5	0b9254761f8edc3a	t1=T(x0,x1,x3;36);
					t2=T(x0,x2,t1;2e);
					t3=T(x0,x1,x2;d8);
					t4=T(x2,x3,t3;78);
					t5=T(x3,t2,t4;b4); y = (t2, t3,
					t4, t5)

95	0123458a6c9fbd7e	12	5	0d2b3a84efc95761	t1=T(x0,x2,x3;4c);
					t2=T(x0,x1,t1;52);
					t3=T(x1,x3,t1;96);
					t4=T(x0,x3,t3;9c);
					t5=T(x2,t2,t4;1e); y = (t2, t3,
					t4, t5)

96	0123458a69cb7fde	13	6	026ce41573d98bfa	t1=T(x0,x2,x3;a2);
					t2=T(x1,x2,t1;6a);
					t3=T(x1,x2,x3;74);
					t4=T(x0,t2,t3;56);
					t5=T(x0,t1,t4;1e);
					t6=T(x2,t3,t4;b4); y = (t2, t4,
					t5, t6)

97	0123458a69cfbed7	13	5	08c46972135bfdea	t1=T(x0,x1,x3;9c);
					t2=T(x2,x3,t1;6c);
					t3=T(x0,x2,t2;6c);
					t4=T(x0,t1,t3;1e);
					t5=T(t1,t2,t4;78); y = (t2, t3,
					t4, t5)

98	0123458a69cfeb7d	13	6	0fea67843c1d295b	t1=T(x0,x2,x3;78);
					t2=T(x1,x3,t1;c6);
					t3=T(x2,t1,t2;70);
					t4=T(x1,t2,t3;96);
					t5=T(t1,t2,t4;d2);
					t6=T(x2,t1,t5;1e); y = (t2, t4,
					t5, t6)

99	0123458a6cf7be9d	13	6	0a84d952fb17ec36	t1=T(x0,x1,x3;78);
					t2=T(x1,x2,t1;7c);
					t3=T(x2,x3,t2;6c);
					t4=T(x3,t1,t2;74);
					t5=T(t1,t3,t4;1e);
					t6=T(x1,t1,t5;1e); y = (t3, t4,
					t5, t6)

100	012345786acfbd9e	13	6	0fb74de318296ac5	t1=T(x1,x2,x3;d2);
					t2=T(x0,x2,t1;92);
					t3=T(x0,x3,t2;ba);
					t4=T(x3,t1,t2;1e);
					t5=T(x0,x2,t4;6c);
					t6=T(x0,x1,t1;78); y = (t3, t4,
					t5, t6)

101	0123458a6cf7bd9e	13	6	048cd79a25ef136b	t1=T(x0,x2,x3;68);
					t2=T(x1,x2,t1;6e);
					t3=T(x1,x3,t1;e6);
					t4=T(x1,x3,t2;42);
					t5=T(x0,x2,t4;9a);
					t6=T(x1,t1,t4;72); y = (t2, t3,
					t5, t6)

102	012345786abcf9ed	13	6	0c7e34d251f68b9a	t1=T(x0,x1,x2;f6);
					t2=T(x0,x2,x3;56);
					t3=T(x0,x1,t2;1e);
					t4=T(x1,x2,t3;78);
					t5=T(x2,t2,t4;4e);
					t6=T(x2,t1,t2;78); y = (t3, t4,
					t5, t6)

103	`012345786ac9debf`	13	6	`04b51dec69f73a82`	`t1=T(x0,x2,x3;d6); t2=T(x0,x1,x3;2c); t3=T(x1,x2,t2;36); t4=T(x3,t1,t2;da); t5=T(x0,t1,t4;68); t6=T(x2,t2,t5;ec); y = (t3, t4, t5, t6)`
104	`012345786ac9edfb`	13	6	`018b47d923fae56c`	`t1=T(x0,x2,x3;a6); t2=T(x0,x1,t1;78); t3=T(x1,x2,t2;68); t4=T(x0,x3,t3;6c); t5=T(x0,x2,t3;8e); t6=T(x1,x3,t1;b4); y = (t2, t4, t5, t6)`
105	`0123458a69cef7db`	13+	6	`04c85bea79fd6312`	`t1=T(x0,x1,x3;6c); t2=T(x2,x3,t1;9c); t3=T(x0,x2,t2;c6); t4=T(x0,t1,t3;b4); t5=T(x2,t3,t4;56); t6=T(t1,t2,t4;78); y = (t2, t3, t5, t6)`
106	`0123458a6c9d7fbe`	11	5	`04f32e9d87b6a15c`	`t1=T(x0,x1,x3;6c); t2=T(x1,x2,t1;36); t3=T(x3,t1,t2;b4); t4=T(x0,t1,t3;b4); t5=T(t2,t3,t4;6c); y = (t1, t2, t4, t5)`
107	`0123458a6b9cde7f`	11	5	`0c78d945ea12f36b`	`t1=T(x0,x1,x2;a6); t2=T(x2,x3,t1;c6); t3=T(x1,t1,t2;96); t4=T(x0,t2,t3;1e); t5=T(x1,t3,t4;4e); y = (t1, t2, t4, t5)`
108	`012345786acfb9de`	11	5	`085b4c9fa3d26e71`	`t1=T(x0,x2,x3;d2); t2=T(x0,x1,t1;9c); t3=T(x1,x3,t1;6c); t4=T(x1,x2,t3;9c); t5=T(x2,t1,t4;9c); y = (t2, t3, t4, t5)`
109	`0123458a6c9d7efb`	11	5	`0e97fd6c284a3b51`	`t1=T(x1,x2,x3;9c); t2=T(x0,x3,t1;52); t3=T(x1,x3,t2;96); t4=T(x0,x3,t3;36); t5=T(x0,x1,t2;ba); y = (t1, t3, t4, t5)`
110	`012345786a9cdfeb`	11	5	`0b74a8fd9625e13c`	`t1=T(x0,x2,x3;62); t2=T(x0,x1,t1;96); t3=T(x0,x1,x2;1e); t4=T(x1,x3,t3;78); t5=T(x2,t2,t4;b4); y = (t2, t3, t4, t5)`
111	`012345786acfdeb9`	12	5	`0278ce341fa965bd`	`t1=T(x1,x2,x3;78); t2=T(x0,x3,t1;c6); t3=T(x0,x1,t1;1e); t4=T(x2,t2,t3;78); t5=T(x2,t1,t4;36); y = (t2, t3, t4, t5)`

112	012345786a9cedfb	12	5	016d98524fe7c3ab	t1=T(x0,x2,x3;b4);
					t2=T(x1,x3,t1;d8);
					t3=T(x1,x2,x3;96);
					t4=T(x2,t2,t3;1e);
					t5=T(t1,t3,t4;c6); y = (t1, t2, t4, t5)

113	012345786acb9efd	12	5	0bd24c6ea8f19735	t1=T(x0,x1,x3;36);
					t2=T(x2,x3,t1;c6);
					t3=T(x0,t1,t2;96);
					t4=T(x0,x2,t2;c6);
					t5=T(t1,t3,t4;78); y = (t2, t3, t4, t5)

114	012345786acf9dbe	12	5	0169ec4ab872fd35	t1=T(x0,x2,x3;9a);
					t2=T(x0,x1,x2;36);
					t3=T(x0,t1,t2;1e);
					t4=T(x2,t2,t3;d8);
					t5=T(x1,x3,t4;1e); y = (t1, t3, t4, t5)

115	0123458a6bc7f9ed	12	5	081b45fc6e97a3d2	t1=T(x0,x1,x2;6c);
					t2=T(x0,x2,x3;1e);
					t3=T(x3,t1,t2;78);
					t4=T(x2,x3,t3;78);
					t5=T(t1,t2,t4;c6); y = (t1, t3, t4, t5)

116	012345786acb9fed	13	6	01438257f9db6eca	t1=T(x0,x2,x3;ba);
					t2=T(x1,x2,t1;62);
					t3=T(x0,x1,t2;dc);
					t4=T(x0,t1,t3;a4);
					t5=T(x1,t1,t4;78);
					t6=T(x2,x3,t3;9c); y = (t2, t4, t5, t6)

117-118_1	012345786a9cedbf	13+	6	0a4eb1df863295c7	t1=T(x0,x1,x3;c6);
					t2=T(x1,x2,t1;ca);
					t3=T(x1,x2,t1;9c);
					t4=T(x0,t1,t3;d2);
					t5=T(x1,x3,t2;b4);
					t6=T(x3,t2,t4;4e); y = (t3, t4, t5, t6)

117-118_2	0123458a6c7fbed9	13+	6	0ac6f97d8543b2e1	t1=T(x0,x1,x2;78);
					t2=T(x0,x1,x3;56);
					t3=T(x2,t1,t2;b4);
					t4=T(x1,t1,t3;c6);
					t5=T(x1,t2,t3;d2);
					t6=T(t2,t4,t5;78); y = (t3, t4, t5, t6)

119	012345786ac9edbf	12	5	0a8e9c641bdf2357	t1=T(x0,x1,x2;e8);
					t2=T(x2,x3,t1;9c);
					t3=T(x0,x2,t2;b4);
					t4=T(x1,x3,t1;e2);
					t5=T(x1,x2,t3;36); y = (t2, t3, t4, t5)

120	012345786abc9fed	12	5	05623f1c8b749dae	t1=T(x1,x2,x3;9c);
					t2=T(x0,x1,t1;36);
					t3=T(x3,t1,t2;6c);
					t4=T(x0,x2,t3;d2);
					t5=T(x2,x3,t4;e4); y = (t2, t3, t4, t5)

121	`012345786acbe9df`	12	6	`086fc4e739db152a`	`t1=T(x0,x1,x3;c8); t2=T(x2,x3,t1;6e); t3=T(x0,x2,t2;c6); t4=T(x1,x2,t3;92); t5=T(x3,t3,t4;4e); t6=T(x0,t3,t5;78); y = (t2, t4, t5, t6)`
122	`0123458a69bce7df`	12	6	`08175dc2af96e34b`	`t1=T(x1,x2,x3;34); t2=T(x0,x3,t1;6a); t3=T(x0,x1,x3;e2); t4=T(x0,x2,t3;6c); t5=T(x0,x1,x3;36); t6=T(t2,t4,t5;9a); y = (t2, t3, t4, t6)`
123	`0123458a69c7fdbe`	12	5	`0ac271395f8e46bd`	`t1=T(x1,x2,x3;c6); t2=T(x0,x2,t1;78); t3=T(x1,x3,t2;da); t4=T(x0,x3,t3;86); t5=T(x1,x2,t3;a2); y = (t1, t2, t4, t5)`
124	`012345786acfbde9`	12	6	`04a7e36f2b18c95d`	`t1=T(x0,x1,x3;98); t2=T(x0,x2,t1;da); t3=T(x1,x2,x3;56); t4=T(x1,x2,t1;1e); t5=T(x0,x2,t4;6c); t6=T(x0,x3,t4;da); y = (t2, t3, t5, t6)`
125	`012345786acb9fde`	12	5	`0b31d24acf65e987`	`t1=T(x0,x1,x2;a6); t2=T(x0,x3,t1;d2); t3=T(x1,t1,t2;c6); t4=T(x2,x3,t3;9c); t5=T(x1,t2,t4;1e); y = (t2, t3, t4, t5)`
126	`0123457869ceafbd`	12	5	`076f8129ca3b54ed`	`t1=T(x0,x1,x2;36); t2=T(x0,x3,t1;78); t3=T(x2,t1,t2;1e); t4=T(x2,x3,t3;c6); t5=T(t1,t2,t4;1e); y = (t2, t3, t4, t5)`
127	`012345786ace9dfb`	12	6	`0126e7af8cd9435b`	`t1=T(x0,x1,x3;68); t2=T(x0,x2,t1;da); t3=T(x0,x2,x3;e6); t4=T(x1,x2,t3;98); t5=T(x1,x2,t1;a6); t6=T(x1,t2,t3;a2); y = (t2, t4, t5, t6)`
128	`0123458a69c7dfeb`	13	6	`04bf95c2e637a81d`	`t1=T(x0,x2,x3;a6); t2=T(x1,x2,x3;b4); t3=T(x2,t1,t2;1e); t4=T(x0,t1,t3;6c); t5=T(x0,t2,t4;d2); t6=T(t3,t4,t5;9c); y = (t2, t3, t5, t6)`
129	`0123458a69cebdf7`	13	6	`0f4b51893762edac`	`t1=T(x1,x2,x3;a6); t2=T(x0,x1,t1;72); t3=T(x0,x2,x3;3a); t4=T(x0,x2,t1;7c); t5=x1^t4; t6=T(x2,t1,t2;72); y = (t2, t3, t5, t6)`

130	`012345786acd9feb`	13	6	`03651ba947e2fdc8`	`t1=T(x0,x1,x3;bc); t2=T(x1,x2,t1;5e); t3=T(x2,x3,t1;6a); t4=T(x0,x2,t3;16); t5=T(x1,x3,t4;a4); t6=T(x2,x3,t4;d8); y = (t2, t3, t5, t6)`
131	`0123458a69bcedf7`	12	6	`07b31e59c8d426fa`	`t1=T(x0,x2,x3;b4); t2=T(x1,x3,t1;72); t3=T(x1,x2,t1;4a); t4=T(x2,x3,t2;d2); t5=T(x2,x3,t3;96); t6=T(t1,t2,t5;1e); y = (t2, t4, t5, t6)`
132	`012345786acd9fbe`	12	5	`0cf7e65134bad298`	`t1=T(x0,x1,x3;4e); t2=T(x0,x1,x3;9c); t3=T(x1,x2,t1;36); t4=T(x3,t2,t3;1e); t5=T(x2,t2,t4;6c); y = (t1, t3, t4, t5)`
133	`012345786acd9efb`	12	5	`041d6278e59acbf3`	`t1=T(x1,x2,x3;6a); t2=T(x0,x1,t1;6c); t3=T(x2,t1,t2;b4); t4=T(x0,x2,t3;1e); t5=T(t1,t2,t4;78); y = (t2, t3, t4, t5)`
134	`0123458a67bcedf9`	11	5	`0c376d12eabf4598`	`t1=T(x0,x1,x2;36); t2=T(x0,x1,x2;6c); t3=T(x2,x3,t2;1e); t4=T(x2,t1,t3;c6); t5=T(x3,t3,t4;d2); y = (t2, t3, t4, t5)`
135	`012345896abcedf7`	11	5	`07652fed41a93cb8`	`t1=T(x0,x2,x3;ac); t2=T(x0,x2,x3;78); t3=T(x0,x1,t1;1e); t4=T(x1,x3,t2;36); t5=T(x1,x2,t4;d8); y = (t2, t3, t4, t5)`
136	`012345786acbf9ed`	12	5	`01edab87945263cf`	`t1=T(x1,x2,x3;6a); t2=T(x0,x2,t1;d2); t3=T(x1,x2,t2;9c); t4=T(x1,t1,t2;b4); t5=T(t1,t3,t4;1e); y = (t2, t3, t4, t5)`
137	`012345786acebd9f`	12	5	`0d76c914e38baf25`	`t1=T(x0,x2,x3;6c); t2=T(x0,x1,x3;b4); t3=T(x1,x3,t1;9c); t4=T(t1,t2,t3;1e); t5=T(x1,x2,t4;1e); y = (t2, t3, t4, t5)`
138	`012345786acdf9eb`	12	6	`05dab23cf917e648`	`t1=T(x0,x1,x2;b4); t2=T(x0,x1,x2;d2); t3=T(x2,x3,t1;1e); t4=T(t1,t2,t3;1e); t5=T(x3,t1,t4;b4); t6=T(t2,t3,t5;78); y = (t3, t4, t5, t6)`

139	0123458a6bcde9f7	12	5	0edb1acf39428576	t1=T(x0,x1,x2;6c); t2=T(x2,x3,t1;96); t3=T(x0,x3,t2;78); t4=T(x0,t1,t3;6c); t5=T(x3,t3,t4;1e); y = (t2, t3, t4, t5)
140	0123458a67bc9fde	12	5	023e7d614cb8f5a9	t1=T(x0,x2,x3;2c); t2=T(x0,x1,x2;a6); t3=T(x0,t1,t2;1e); t4=T(x1,x3,t2;1e); t5=T(x3,t1,t4;d2); y = (t2, t3, t4, t5)
141	012345896abc7fde	12	5	0e972f18ca645bd3	t1=T(x0,x1,x2;c6); t2=T(x2,x3,t1;d2); t3=T(x0,t1,t2;b4); t4=T(x0,x3,t3;6c); t5=T(x3,t1,t3;36); y = (t2, t3, t4, t5)
142	012345786ace9bfd	9	4	08d43bcf6e7a1925	t1=T(x0,x1,x2;a6); t2=T(x2,x3,t1;6c); t3=T(x1,t1,t2;d2); t4=T(x0,t2,t3;d2); y = (t1, t2, t3, t4)
143	012345786acf9bde	9	4	041526f38b7cad9e	t1=T(x0,x1,x3;6c); t2=T(x2,x3,t1;78); t3=T(x0,t1,t2;78); t4=T(x0,x3,t3;c6); y = (t1, t2, t3, t4)
144	012345786acfde9b	10	5	0967542acd81bfe3	t1=T(x0,x2,x3;4c); t2=T(x0,x1,t1;52); t3=T(x1,x2,x3;d8); t4=T(x1,x3,t1;96); t5=T(x0,x3,t4;9c); y = (t2, t3, t4, t5)
145	012345786acedfb9	10	5	05c9ef671a8bd243	t1=T(x0,x1,x3;d2); t2=T(x0,x2,x3;e4); t3=T(x1,x3,t2;98); t4=T(x0,x2,t3;56); t5=T(x1,x2,t3;36); y = (t1, t2, t4, t5)
146	012345896ace7bfd	10	4	0246b58e31fdac79	t1=T(x1,x2,x3;a6); t2=T(x0,x3,t1;1e); t3=T(x1,x2,t2;b4); t4=T(x2,t1,t3;78); y = (t1, t2, t3, t4)
147	0123458a67cf9bde	10	4	0c1dea956f3287b4	t1=T(x0,x1,x3;6c); t2=T(x2,x3,t1;9c); t3=T(x0,t1,t2;d2); t4=T(x0,x2,t3;6c); y = (t1, t2, t3, t4)
148	0123458a67cf9bed	10	4	012d6f3489e5ab7c	t1=T(x0,x1,x2;78); t2=T(x0,x1,x2;2e); t3=T(x1,x3,t2;d2); t4=T(x3,t1,t3;78); y = (t1, t2, t3, t4)
149	0123457869acefdb	10	5	094d67fec38b251a	t1=T(x0,x1,x2;78); t2=T(x2,x3,t1;78); t3=T(x1,x3,t2;36); t4=T(x1,t1,t2;4e); t5=T(x3,t2,t4;56); y = (t1, t2, t3, t5)

150	`012345896acf7bed`	10	4	`018d23a7efc954b6`	`t1=T(x0,x2,x3;78); t2=T(x1,x2,x3;c6); t3=T(x1,x3,t1;6c); t4=T(x1,x2,t3;72); y = (t1, t2, t3, t4)`
151	`0123458a67bc9efd`	11	5	`089fbd714c532e6a`	`t1=T(x1,x2,x3;74); t2=T(x0,x2,t1;6c); t3=T(x0,x1,x2;c2); t4=T(x2,x3,t3;56); t5=T(x0,t1,t4;b4); y = (t1, t2, t4, t5)`
152	`0123457869acfebd`	11	5	`01834c57eb69afd2`	`t1=T(x0,x1,x3;9c); t2=T(x0,x2,t1;78); t3=T(x3,t1,t2;72); t4=T(x2,t2,t3;d2); t5=T(x3,t1,t2;b4); y = (t2, t3, t4, t5)`
153	`012345896abce7df`	11	5	`02695e31c8afd4b7`	`t1=T(x0,x1,x2;ca); t2=T(x0,x1,x3;9c); t3=T(x0,x3,t1;2e); t4=T(x1,x2,t3;5e); t5=T(x2,x3,t3;9e); y = (t1, t2, t4, t5)`
154	`0123458a6bcfed79`	11	5	`0a3b2cd9768f54e1`	`t1=T(x0,x1,x3;c6); t2=T(x0,x3,t1;96); t3=T(x0,x2,t1;36); t4=T(x3,t2,t3;b4); t5=T(t1,t2,t4;9c); y = (t1, t3, t4, t5)`
155	`012345896abc7efd`	10	4	`09a1678f54edb2c3`	`t1=T(x0,x1,x3;d2); t2=T(x1,x2,t1;9c); t3=T(x2,x3,t2;6c); t4=T(x1,t1,t3;b4); y = (t1, t2, t3, t4)`
156	`012345786abcedf9`	12	6	`0c86fb957314ea2d`	`t1=T(x0,x1,x3;6a); t2=T(x2,x3,t1;78); t3=T(x1,t1,t2;4e); t4=T(x0,x1,t3;d2); t5=T(x2,t3,t4;c2); t6=T(x1,t1,t5;1a); y = (t2, t3, t4, t6)`
157	`0123457869cefbda`	13	6	`091487dc2ba5ef36`	`t1=T(x1,x2,x3;24); t2=T(x0,x1,t1;b4); t3=T(x0,x3,t1;6c); t4=T(x1,x2,x3;b4); t5=T(x0,x2,t4;e4); t6=T(t2,t4,t5;36); y = (t2, t3, t5, t6)`
158	`0123458a69ce7fbd`	13	6	`07ba2f9614583ecd`	`t1=T(x1,x2,x3;b6); t2=T(x0,x2,t1;96); t3=T(x0,x3,t1;76); t4=T(x0,x2,t3;b2); t5=T(x1,t1,t4;6a); t6=T(x1,x2,t4;68); y = (t2, t3, t5, t6)`

159	`012345786abc9fde`	13	6	`0e164a978cf3bd52`	`t1=T(x1,x2,x3;68); t2=T(x0,x1,t1;ae); t3=T(x0,x1,t1;3a); t4=T(x0,x3,t3;78); t5=T(x0,x2,t1;96); t6=T(x1,x3,t3;ae); y = (t2, t4, t5, t6)`
160	`012345786acfd9be`	13	6	`012d6b45fa893e7c`	`t1=T(x1,x2,x3;9e); t2=T(x0,x3,t1;78); t3=T(x0,x1,t1;2a); t4=T(x0,x3,t1;28); t5=T(x1,x2,t4;e6); t6=T(x2,x3,t4;6e); y = (t2, t3, t5, t6)`
161	`012345786acf9ebd`	13	6	`0f4c185327ea69bd`	`t1=T(x0,x2,x3;78); t2=T(x1,x2,t1;c6); t3=T(x3,t1,t2;d8); t4=T(x1,t1,t3;78); t5=T(t1,t3,t4;1e); t6=T(x3,t4,t5;96); y = (t2, t3, t4, t6)`
162	`012345786acfeb9d`	13	6	`09c72ba3f8451ed6`	`t1=T(x1,x2,x3;9a); t2=T(x0,x3,t1;78); t3=T(x2,t1,t2;78); t4=T(x1,t1,t3;d8); t5=T(t2,t3,t4;72); t6=T(x1,t1,t5;96); y = (t2, t3, t4, t6)`
163	`012345786abce9fd`	11	5	`0a39b1fd46578ec2`	`t1=T(x1,x2,x3;74); t2=T(x0,x1,t1;9c); t3=T(x0,x3,t1;d2); t4=T(x2,x3,t2;ac); t5=T(x0,x2,t4;9c); y = (t1, t3, t4, t5)`
164	`012345786abced9f`	11	5	`027d31b94e865fca`	`t1=T(x1,x2,x3;5c); t2=T(x0,x3,t1;d2); t3=T(x0,x2,t1;36); t4=T(x1,x3,t3;ac); t5=T(x1,t2,t4;78); y = (t1, t2, t4, t5)`
165	`012345786ac9defb`	11	5	`01e93876cfab4d25`	`t1=T(x0,x2,x3;2e); t2=T(x0,x2,x3;b4); t3=T(x1,t1,t2;d8); t4=T(x1,x3,t3;6c); t5=T(x1,x2,t1;36); y = (t2, t3, t4, t5)`
166	`012345786abcfde9`	11	5	`016b72548fcda9e3`	`t1=T(x0,x2,x3;b4); t2=T(x0,x1,x3;a8); t3=T(x1,x2,t2;96); t4=T(x1,t1,t3;78); t5=T(x1,x3,t4;9c); y = (t1, t3, t4, t5)`
167	`0123457869acbefd`	11	5	`0a1b52478639fecd`	`t1=T(x0,x1,x2;c6); t2=T(x0,x3,t1;78); t3=T(x2,t1,t2;d2); t4=T(x2,x3,t3;e8); t5=T(x2,x3,t2;c6); y = (t1, t2, t4, t5)`

168	`012345786abce9df`	12	5	`02138ad9ef764b5c`	`t1=T(x0,x1,x3;6c); t2=T(x2,x3,t1;6c); t3=T(x0,x2,t2;72); t4=T(x0,x3,t2;8e); t5=T(x1,x2,t2;86); y = (t1, t3, t4, t5)`
169	`0123457869acbfde`	12	5	`02a8d65417efc39b`	`t1=T(x0,x1,x3;52); t2=T(x0,x2,t1;56); t3=T(x0,x1,t2;b4); t4=T(x2,x3,t1;b4); t5=T(x1,t3,t4;d2); y = (t2, t3, t4, t5)`
170	`0123458a69bcfd7e`	12	5	`02f9867531bd4eac`	`t1=T(x1,x2,x3;72); t2=T(x0,x1,t1;1e); t3=T(x1,x3,t2;36); t4=T(x0,x2,t3;da); t5=T(x1,x2,t3;74); y = (t1, t2, t4, t5)`
171	`012345786acbed9f`	12	6	`0d8c43951fae7b62`	`t1=T(x0,x1,x2;74); t2=T(x1,x3,t1;1e); t3=T(x2,x3,t1;e8); t4=T(x0,t2,t3;a4); t5=T(x0,x2,t4;74); t6=T(x3,t1,t4;c6); y = (t2, t3, t5, t6)`
172	`012345786abc9edf`	12	5	`071456892fe3cdba`	`t1=T(x1,x2,x3;58); t2=T(x0,x2,t1;96); t3=T(x0,x3,t1;54); t4=T(x1,x2,t2;1e); t5=T(x2,x3,t4;d8); y = (t2, t3, t4, t5)`
173	`012345786acebfd9`	12	5	`0aef2c6d1b893547`	`t1=T(x0,x1,x3;e2); t2=T(x0,x1,x2;5e); t3=T(x1,x3,t2;6a); t4=T(x1,x2,t3;e4); t5=T(x2,x3,t2;1a); y = (t1, t3, t4, t5)`
174	`012345786acefb9d`	12	5	`09453b76ad81c2fe`	`t1=T(x0,x1,x3;e2); t2=T(x0,x2,t1;74); t3=T(x0,x2,t1;c6); t4=T(x1,x3,t3;b4); t5=T(x0,x3,t4;9c); y = (t2, t3, t4, t5)`
175	`012345786a9cfdbe`	12	5	`05e1c9ad24f36b78`	`t1=T(x0,x1,x2;9c); t2=T(x0,x3,t1;78); t3=T(x0,x3,t1;4e); t4=T(x2,t1,t2;9c); t5=T(x2,t3,t4;78); y = (t2, t3, t4, t5)`
176	`0123457869acebdf`	12	6	`04396e1b2ad5cf78`	`t1=T(x0,x1,x3;26); t2=T(x0,x2,x3;5c); t3=T(x1,x2,t2;b4); t4=T(x2,t1,t2;64); t5=T(x3,t2,t3;e4); t6=T(x0,t4,t5;d2); y = (t3, t4, t5, t6)`

177	`012345786a9cdebf`	12	5	`027918e346df5acb`	`t1=T(x0,x1,x2;c6); t2=T(x2,x3,t1;d2); t3=T(x0,x3,t2;d2); t4=T(x3,t2,t3;78); t5=T(t1,t2,t4;6c); y = (t1, t3, t4, t5)`
178	`0123458a6bc97fde`	12	6	`0ab3c1d257648e9f`	`t1=T(x0,x2,x3;62); t2=x1^t1; t3=T(x0,x1,x2;94); t4=T(x0,t1,t3;ba); t5=T(x0,x2,x3;a6); t6=T(x2,t1,t3;32); y = (t2, t4, t5, t6)`
179	`0123458a69c7fbed`	12	6	`0b2e189374fa5cd6`	`t1=T(x0,x1,x2;6a); t2=T(x0,x3,t1;1e); t3=T(x0,x1,t2;28); t4=T(x1,t1,t2;72); t5=T(x3,t1,t4;78); t6=T(t1,t2,t3;98); y = (t2, t4, t5, t6)`
180	`012345786acdf9be`	13	6	`0bde824fc19367a5`	`t1=T(x0,x2,x3;52); t2=T(x0,x1,x2;94); t3=T(x0,t1,t2;ba); t4=T(x3,t1,t2;1c); t5=T(x1,t1,t4;96); t6=T(x2,t1,t5;9e); y = (t3, t4, t5, t6)`
181	`012345786acde9fb`	13	6	`04ad3b2c15968e7f`	`t1=T(x0,x1,x2;62); t2=T(x0,x1,x2;c6); t3=x3^t1; t4=T(x2,t2,t3;b4); t5=T(x0,x3,t4;d8); t6=T(t1,t2,t4;6c); y = (t3, t4, t5, t6)`
182	`012345786acd9ebf`	13	5	`01c58f4723ba9d6e`	`t1=T(x0,x1,x2;c6); t2=T(x0,x3,t1;78); t3=T(x2,x3,t2;6c); t4=T(x2,t1,t3;b4); t5=T(t1,t2,t4;78); y = (t2, t3, t4, t5)`
183	`012345786acbde9f`	11	5	`0ef9c2db71358a46`	`t1=T(x1,x2,x3;72); t2=T(x0,x2,t1;d2); t3=T(x2,x3,t2;16); t4=T(x1,t1,t3;96); t5=T(x3,t1,t4;1e); y = (t1, t2, t4, t5)`
184	`012345786acbdfe9`	11	5	`01a7be562f89cd43`	`t1=T(x0,x2,x3;b4); t2=T(x1,x2,t1;b0); t3=T(x2,x3,t2;96); t4=T(x2,t1,t3;78); t5=T(x1,x2,t4;1e); y = (t1, t3, t4, t5)`
185	`0123458a6bc7edf9`	11	5	`0c85fb432e679da1`	`t1=T(x0,x1,x2;e2); t2=T(x2,x3,t1;6c); t3=T(x1,x3,t2;84); t4=T(x0,x2,t3;b6); t5=T(x1,t1,t4;1e); y = (t1, t2, t4, t5)`
186	`0123458a6b9cf7ed`	11	5	`0462d1f3cbe5987a`	`t1=T(x0,x2,x3;6c); t2=T(x1,x2,t1;d2); t3=x3^t1; t4=T(x0,t2,t3;1e); t5=T(x3,t3,t4;d8); y = (t1, t2, t4, t5)`

187	`012345786a9cdbef`	12	5	`0f594c2a18e67db3`	`t1=T(x0,x1,x2;56); t2=T(x2,x3,t1;c6); t3=T(x1,t1,t2;78); t4=T(x0,t1,t3;9c); t5=T(x0,x1,x3;78); y = (t2, t3, t4, t5)`
188	`0123458a69cd7feb`	12	6	`0d9ac61572b8f43e`	`t1=T(x0,x1,x2;6c); t2=T(x0,x3,t1;1e); t3=T(x0,x2,t2;92); t4=T(x0,x3,t1;6c); t5=T(x2,t1,t2;72); t6=T(t1,t3,t5;42); y = (t2, t4, t5, t6)`
189	`0123456879cebfda`	12	6	`0be1ad8f2c479356`	`t1=T(x0,x1,x2;9a); t2=T(x0,x3,t1;78); t3=T(x1,x3,t2;36); t4=T(t1,t2,t3;96); t5=T(x1,t3,t4;b4); t6=T(t1,t3,t5;d8); y = (t2, t4, t5, t6)`
190	`0123456879bcaefd`	11	5	`06754293aefb1dc8`	`t1=T(x1,x2,x3;78); t2=T(x0,x2,t1;1e); t3=T(x2,x3,t2;1e); t4=T(t1,t2,t3;9c); t5=T(x3,t2,t4;b4); y = (t1, t3, t4, t5)`
191	`0123456879ceafdb`	11	5	`0154736ab8fcde92`	`t1=T(x0,x1,x2;c6); t2=T(x0,x3,t1;1e); t3=T(x2,x3,t2;78); t4=T(x2,t1,t2;9c); t5=T(x3,t1,t4;b4); y = (t2, t3, t4, t5)`
192	`0123456879bcfead`	10	5	`0a9857dc32461bfe`	`t1=T(x1,x2,x3;d2); t2=T(x0,x2,t1;4e); t3=T(x0,x3,t1;d8); t4=T(x1,x2,x3;e4); t5=T(t1,t3,t4;b4); y = (t2, t3, t4, t5)`
193	`0123456879bcefda`	10	5	`0ac64e38d5f7192b`	`t1=T(x0,x1,x2;78); t2=T(x0,x3,t1;c6); t3=T(x1,t1,t2;e4); t4=T(x1,x2,x3;36); t5=T(x0,x2,t4;d2); y = (t2, t3, t4, t5)`
194	`012345786abcde9f`	12	6	`08bf1a2d5e49c736`	`t1=T(x0,x2,x3;86); t2=x1^t1; t3=T(x3,t1,t2;d6); t4=T(x2,t1,t3;92); t5=T(x0,t3,t4;c4); t6=T(x0,t1,t3;32); y = (t2, t4, t5, t6)`
195	`0123457869acfbde`	13	6	`04578c9e6dbfa213`	`t1=T(x0,x1,x3;ac); t2=T(x1,x2,t1;b2); t3=T(x2,x3,t2;e6); t4=T(x0,x1,t3;e6); t5=T(x0,x2,t3;72); t6=T(x2,x3,t1;78); y = (t2, t4, t5, t6)`

196	`0123458a6bc7dfe9`	13	6	`023ac68de475b1f9`	`t1=T(x0,x1,x2;a6); t2=T(x1,x3,t1;e8); t3=T(x0,x3,t2;1e); t4=T(x0,t1,t3;98); t5=T(x2,x3,t1;6c); t6=T(x1,t4,t5;36); y = (t2, t3, t5, t6)`
197	`012345786ac9bfde`	12	6	`0a84fec391d576b2`	`t1=T(x0,x1,x3;1e); t2=T(x2,x3,t1;9c); t3=T(x0,x1,t2;d8); t4=T(x1,x2,t3;8e); t5=T(x1,x2,t3;6c); t6=T(t1,t3,t4;d2); y = (t2, t4, t5, t6)`
198	`012345786a9cbfed`	12	6	`0c972a58e6f4b3d1`	`t1=T(x0,x1,x3;6a); t2=T(x1,x2,x3;98); t3=T(x0,x1,t2;ae); t4=T(x1,x2,t1;2e); t5=T(x0,x2,t4;36); t6=T(x0,x3,t2;1e); y = (t3, t4, t5, t6)`
199	`012345786ac9fbed`	12	6	`0147cdf2635a98eb`	`t1=T(x1,x2,x3;c2); t2=T(x0,x3,t1;96); t3=T(x0,x1,t1;ea); t4=T(x1,x2,x3;74); t5=T(t2,t3,t4;a6); t6=T(x2,t4,t5;b4); y = (t2, t3, t5, t6)`
200	`0123457869acebfd`	12	6	`02f871e3db95ca64`	`t1=T(x0,x1,x3;a6); t2=T(x0,x1,t1;96); t3=T(x2,x3,t1;1e); t4=T(t1,t2,t3;36); t5=T(x0,t1,t4;4e); t6=T(x1,t1,t4;9c); y = (t3, t4, t5, t6)`
201	`0123457869bceafd`	11	5	`08da253be7f64c19`	`t1=T(x1,x2,x3;36); t2=T(x0,x1,t1;6c); t3=T(x3,t1,t2;c6); t4=T(x3,t1,t2;d8); t5=T(x0,t1,t4;78); y = (t2, t3, t4, t5)`
202	`012345786aced9fb`	11	5	`0c1b7dea4862f953`	`t1=T(x1,x2,x3;9c); t2=T(x0,x3,t1;36); t3=T(x0,x1,t1;c6); t4=T(x0,x3,t3;1e); t5=T(x0,t2,t3;d2); y = (t1, t3, t4, t5)`
203	`012345786abc9efd`	11	5	`03651cf2487adbe9`	`t1=T(x0,x1,x3;f2); t2=T(x2,x3,t1;96); t3=T(x0,x1,t2;6c); t4=T(x0,x3,t3;1e); t5=T(x2,x3,t4;e4); y = (t2, t3, t4, t5)`
204	`012345786a9cbdfe`	11	5	`0cfd136248ea5b79`	`t1=T(x1,x2,x3;b8); t2=T(x1,x2,x3;9c); t3=T(x0,x1,t2;6c); t4=T(x0,x3,t3;1e); t5=T(x0,x2,t1;b2); y = (t2, t3, t4, t5)`

205	012345786ac9bfed	11	5	0cd126734a8ebf95	t1=T(x1,x2,x3;d8);
					t2=T(x1,x3,t1;c4);
					t3=T(x0,x2,t2;64);
					t4=T(x0,x1,t2;96);
					t5=T(x3,t3,t4;d2); y = (t1, t3,
					t4, t5)

206	012345786abcf9de	11	5	03c7abef25164d98	t1=T(x0,x1,x3;78);
					t2=T(x1,x3,t1;e4);
					t3=T(x2,x3,t1;36);
					t4=T(x1,x3,t3;b4);
					t5=T(x1,x2,t2;d4); y = (t1, t3,
					t4, t5)

207	012345786a9cebdf	11	5	0e7f586d4cb193a2	t1=T(x0,x2,x3;9c);
					t2=T(x1,x2,t1;78);
					t3=T(x0,t1,t2;c6);
					t4=T(x3,t2,t3;1e);
					t5=T(x0,x3,t4;b4); y = (t2, t3,
					t4, t5)

208	012345786a9cfbed	11	5	0298135bc6e4d7af	t1=~x0&x1; t2=T(x2,x3,t1;72);
					t3=T(x0,x1,t2;b4);
					t4=T(x2,x3,t1;6c);
					t5=T(x1,t3,t4;d2); y = (t2, t3,
					t4, t5)

209	012345786a9cfdeb	12	5	0435d7269b1fca8e	t1=T(x1,x2,x3;36);
					t2=T(x0,x3,t1;c6);
					t3=T(x0,x2,t2;e6);
					t4=T(x0,x1,t2;d2);
					t5=T(x1,x3,t2;8e); y = (t1, t3,
					t4, t5)

210	0123457869acdebf	12	5	01cfab2967e354d8	t1=T(x0,x2,x3;78);
					t2=T(x1,x3,t1;6c);
					t3=T(x2,x3,t2;1e);
					t4=T(x1,t2,t3;9c);
					t5=T(x1,t3,t4;b4); y = (t1, t3,
					t4, t5)

211	0123457869acdfeb	12	5	0f3c47b9e8ad6521	t1=T(x0,x1,x3;64);
					t2=T(x0,x2,t1;da);
					t3=T(x0,x1,x3;1e);
					t4=T(x1,x2,t3;1e);
					t5=T(x2,x3,t4;1e); y = (t2, t3,
					t4, t5)

212	012345687abc9fde	12	6	09c7146b2afed538	t1=T(x0,x1,x3;58);
					t2=T(x1,x2,t1;a6);
					t3=T(x2,x3,t1;14);
					t4=T(x1,t2,t3;62);
					t5=T(x1,x2,t3;b4);
					t6=T(x0,t3,t5;d2); y = (t2, t4,
					t5, t6)

213	012345687abce9fd	11	5	0cf26a78d5e31b49	t1=T(x0,x1,x3;a6);
					t2=T(x0,x2,t1;c6);
					t3=T(x1,x2,t2;78);
					t4=T(x0,t2,t3;1e);
					t5=T(t1,t2,t4;c6); y = (t1, t3,
					t4, t5)

214	012345687abcefd9	11	5	017a63bef49c52d8	t1=T(x0,x2,x3;6c);
					t2=T(x0,x1,x3;1e);
					t3=T(x1,x3,t1;36);
					t4=T(t1,t2,t3;78);
					t5=T(x1,t1,t4;d2); y = (t2, t3,
					t4, t5)

215	`012345896abcfd7e`	9	4	`01b4897cefad2365`	`t1=T(x0,x1,x3;b4); t2=T(x1,x3,t1;6c); t3=T(x1,x2,t2;d2); t4=T(x2,x3,t2;1e); y = (t1, t2, t3, t4)`
216	`012345896acf7bde`	9	4	`023dba85ce9f1647`	`t1=T(x0,x1,x2;c6); t2=T(x0,x3,t1;d2); t3=T(x0,x3,t1;e4); t4=T(x2,x3,t3;1e); y = (t1, t2, t3, t4)`
217	`0123457869acbedf`	12	5	`0d129c5a7f3b6e48`	`t1=T(x0,x1,x2;b4); t2=T(x2,x3,t1;1e); t3=T(x1,x3,t2;9c); t4=T(x1,t1,t3;1e); t5=T(t1,t2,t4;b4); y = (t2, t3, t4, t5)`
218	`012345786abcdef9`	9	4	`08c97f346e2b5d1a`	`t1=T(x0,x1,x2;6a); t2=T(x2,x3,t1;6c); t3=T(x1,t1,t2;1e); t4=T(x0,x1,t3;78); y = (t1, t2, t3, t4)`
219	`012345786a9cbefd`	10	4	`0891be2f4c735a6d`	`t1=T(x0,x1,x2;c6); t2=T(x2,x3,t1;78); t3=T(x0,x3,t2;6c); t4=T(x0,t1,t3;b4); y = (t1, t2, t3, t4)`
220	`0123456879acbefd`	12	6	`02579edcb4138af6`	`t1=T(x1,x2,x3;c6); t2=T(x0,x1,t1;4e); t3=T(x1,x3,t2;36); t4=T(x0,t1,t3;74); t5=T(x1,x2,t3;e6); t6=T(x3,t1,t2;9c); y = (t2, t4, t5, t6)`
221	`0123456879acfbde`	11	5	`0bc4d5382f7a19e6`	`t1=T(x1,x2,x3;9c); t2=T(x0,x1,t1;36); t3=T(x3,t1,t2;72); t4=T(x3,t1,t2;6c); t5=T(x0,t1,t4;78); y = (t2, t3, t4, t5)`
222	`0123456879acdefb`	11	5	`0127c9af38de456b`	`t1=T(x0,x2,x3;d2); t2=T(x0,x1,t1;c6); t3=T(x3,t1,t2;86); t4=T(x1,x2,t3;54); t5=T(x2,x3,t3;b4); y = (t1, t2, t4, t5)`
223	`012345786a9cfbde`	12	6	`0462f5138cad97eb`	`t1=T(x1,x2,x3;6c); t2=T(x0,x1,t1;c6); t3=T(x0,x2,t1;e8); t4=T(x3,t1,t2;6a); t5=T(x0,t2,t4;78); t6=T(x1,x3,t4;c6); y = (t3, t4, t5, t6)`
224	`0123457869acedbf`	13	6	`0891d6e7c5b4f2a3`	`t1=T(x0,x2,x3;86); t2=T(x1,x3,t1;96); t3=T(x1,x2,t2;e4); t4=T(x2,t1,t3;b8); t5=T(x2,x3,t3;74); t6=T(x0,t3,t5;1e); y = (t2, t4, t5, t6)`

225	`0123456879bceafd`	12	6	`05278da4c96f1e3b`	`t1=T(x0,x1,x2;9a); t2=T(x0,x3,t1;96); t3=T(x0,x2,t2;b4); t4=T(x0,x1,t3;9c); t5=T(x3,t1,t2;36); t6=T(t2,t4,t5;78); y = (t3, t4, t5, t6)`
226	`0123456879ceafbd`	12	6	`0ed1a632bfc89745`	`t1=T(x0,x1,x2;bc); t2=T(x0,x3,t1;96); t3=T(x2,x3,t1;9e); t4=T(x1,t1,t3;4e); t5=T(x1,t1,t4;48); t6=T(x2,t1,t3;3a); y = (t2, t4, t5, t6)`
227	`012345786a9cefdb`	10	4	`0c1d362784fb9ea5`	`t1=T(x0,x1,x2;c6); t2=T(x2,x3,t1;78); t3=T(x0,x2,t2;d2); t4=T(x3,t2,t3;d2); y = (t1, t2, t3, t4)`
228	`0123456879bcdefa`	13	6	`02468afe1b3795cd`	`t1=T(x0,x1,x2;5c); t2=T(x2,x3,t1;6c); t3=T(x0,t1,t2;d8); t4=T(x0,x1,t2;28); t5=T(x1,x2,t4;d8); t6=T(x1,x2,t4;c6); y = (t2, t3, t5, t6)`
229	`0123456879bcefad`	11	5	`0b29daf46781c3e5`	`t1=T(x0,x2,x3;e2); t2=T(x0,x2,x3;b4); t3=T(x0,x1,t1;36); t4=T(x1,x2,t1;c6); t5=T(t1,t2,t4;1e); y = (t2, t3, t4, t5)`
230	`0123456879acdebf`	11	5	`0dfe1a3c957b6284`	`t1=T(x0,x1,x2;b4); t2=T(x2,x3,t1;1e); t3=T(x1,x2,t2;b4); t4=T(t1,t2,t3;d2); t5=T(x3,t1,t4;1e); y = (t2, t3, t4, t5)`
231	`012345687a9cfbde`	11	5	`0a2e78d53b916fc4`	`t1=T(x0,x1,x2;9a); t2=T(x2,x3,t1;6c); t3=T(x1,t1,t2;1e); t4=T(x0,t1,t3;6c); t5=T(x0,x1,t3;b4); y = (t2, t3, t4, t5)`
232	`012345786acdb9ef`	11	5	`0d27ac8e15f3694b`	`t1=T(x0,x2,x3;b2); t2=T(x0,x1,x2;c6); t3=T(x1,x2,x3;56); t4=T(x0,x3,t3;b4); t5=T(x2,t3,t4;d2); y = (t1, t2, t4, t5)`
233	`0123458967ceafdb`	11	5	`09a3feb278165cd4`	`t1=T(x0,x2,x3;1e); t2=T(x1,x3,t1;78); t3=T(x2,x3,t2;3a); t4=T(x2,x3,t2;d8); t5=T(x0,t2,t4;36); y = (t1, t3, t4, t5)`

234	`0123458967cebfda`	12	5	`0a92d64e57f83b1c`	`t1=T(x0,x1,x2;c6); t2=T(x0,x3,t1;4e); t3=T(x0,x3,t1;78); t4=T(x2,x3,t2;78); t5=T(t1,t3,t4;b4); y = (t2, t3, t4, t5)`
235	`012345687a9cdfbe`	10	5	`0145c73869daef2b`	`t1=T(x1,x2,x3;6c); t2=T(x0,x1,t1;78); t3=T(x2,x3,t2;e4); t4=T(x1,x2,t3;1e); t5=T(x3,t1,t2;e4); y = (t2, t3, t4, t5)`
236	`012345687abc9efd`	10	5	`014dabc958fe2367`	`t1=T(x0,x2,x3;d2); t2=T(x1,x2,x3;ac); t3=T(x1,x3,t1;28); t4=T(x1,x2,t3;72); t5=T(x2,x3,t3;36); y = (t1, t2, t4, t5)`
237	`012345786abcfd9e`	11	5	`038b2d9a47fec156`	`t1=T(x1,x2,x3;e4); t2=T(x0,x1,t1;78); t3=T(x2,t1,t2;4e); t4=T(x0,x3,t3;9c); t5=T(x1,x2,t4;78); y = (t2, t3, t4, t5)`
238	`0123458967ceafbd`	12	5	`0efd572431b96c8a`	`t1=T(x1,x2,x3;36); t2=T(x1,x2,x3;b6); t3=T(x0,x2,t2;96); t4=T(x0,x3,t2;76); t5=T(x0,x1,t2;98); y = (t1, t3, t4, t5)`
239	`0123457869bcdefa`	11	5	`021357e4dfac9b68`	`t1=T(x1,x2,x3;1e); t2=T(x0,x1,t1;b4); t3=T(x0,t1,t2;94); t4=T(x2,x3,t3;78); t5=T(x3,t3,t4;d2); y = (t1, t2, t4, t5)`
240	`0123456879acdfbe`	11	5	`0a31fd75ec24b968`	`t1=T(x1,x2,x3;5c); t2=T(x0,x3,t1;1e); t3=T(x1,x3,t2;4e); t4=T(x2,x3,t3;78); t5=T(x1,t3,t4;4e); y = (t1, t2, t4, t5)`
241	`012345687abcdef9`	11	5	`027546a839f1cebd`	`t1=T(x1,x2,x3;b2); t2=T(x0,x1,t1;1e); t3=T(x1,x3,t2;b4); t4=T(x1,x2,t3;6c); t5=T(x3,t3,t4;e4); y = (t1, t2, t4, t5)`
242	`012345687a9cbefd`	11	5	`06b8179e4cfa53d2`	`t1=T(x0,x1,x2;2e); t2=T(x0,x1,x3;6c); t3=T(x0,x2,t2;d2); t4=T(x0,x3,t3;6c); t5=T(x1,x2,t2;b2); y = (t1, t3, t4, t5)`
243	`0123457869cdefba`	9	4	`0f57de68439b12ac`	`t1=T(x0,x1,x2;56); t2=T(x0,x1,t1;b4); t3=T(x3,t1,t2;1e); t4=T(x1,x3,t3;d2); y = (t1, t2, t3, t4)`

244	0123456879cdefba	10	5	04739dfb8ce26a51	t1=T(x0,x1,x2;60); t2=T(x1,x2,x3;d4); t3=T(x1,x2,x3;78); t4=T(x0,x1,t3;1e); t5=T(x2,x3,t1;b4); y = (t2, t3, t4, t5)
245	0123458a69cefbd7	12	5	061fae9dc27584b3	t1=T(x1,x2,x3;a4); t2=T(x0,x2,t1;9a); t3=T(x0,x1,x2;6a); t4=T(x0,x3,t3;b4); t5=T(t2,t3,t4;c6); y = (x1, t2, t4, t5)
246	0123458a69b7cefd	10	5	0eac26845b317f9d	t1=T(x0,x1,x3;52); t2=T(x0,x1,x2;36); t3=T(x0,x2,t1;ea); t4=T(x0,x2,x3;4e); t5=T(t2,t3,t4;56); y = (x3, t2, t3, t5)
247	0123458a69b7cfde	11	5	02a6e84c13d597bf	t1=T(x1,x2,x3;94); t2=T(x0,x1,t1;9a); t3=T(x0,x1,x2;2c); t4=T(x1,t1,t3;76); t5=T(x2,t3,t4;7c); y = (x3, t2, t4, t5)
248	0123458a67b9cfde	9	4	042351fe8ca97bd6	t1=T(x0,x1,x2;6a); t2=T(x1,x3,t1;78); t3=x0^t1; t4=T(x3,t2,t3;78); y = (t1, t2, t3, t4)
249	012345768acf9bed	9	4	085b4c9fa2d36e71	t1=T(x0,x2,x3;d2); t2=T(x1,x3,t1;6c); t3=T(x1,x2,t2;9c); t4=T(x2,t1,t3;9c); y = (x1, t2, t3, t4)
250	0123458a6b97cfde	10	4	0cd1aebf24938675	t1=T(x0,x1,x3;78); t2=T(x2,x3,t1;b4); t3=T(x1,t1,t2;9c); t4=T(x3,t2,t3;1e); y = (x1, t2, t3, t4)
251	012345896ab7cfde	10	4	02684acedb75139f	t1=T(x0,x1,x2;b4); t2=T(x1,x2,x3;96); t3=T(x1,t1,t2;d2); t4=T(x3,t2,t3;c6); y = (x3, t1, t3, t4)
252	0123458a6c7dfeb9	10	4	075a6b1493cedf28	t1=T(x0,x1,x3;36); t2=T(x0,x2,t1;b4); t3=T(x2,x3,t1;96); t4=T(x3,t2,t3;b4); y = (t1, t2, t3, t4)
253	012345896acefdb7	7	3	0ac4db15286e793f	t1=T(x0,x1,x3;9a); t2=T(x0,x1,x2;c6); t3=T(x0,t1,t2;d2); y = (x2, t1, t2, t3)
254	012345768acf9bde	7	3	04c8a6e253db79f1	t1=T(x0,x2,x3;6c); t2=T(x0,x1,x3;1e); t3=T(x1,x2,t2;b4); y = (x3, t1, t2, t3)
255	012345768ace9bfd	8	3	018bc54fe7a9236d	t1=T(x0,x1,x3;6a); t2=T(x1,x2,x3;c6); t3=T(x0,x1,t2;c6); y = (x0, t1, t2, t3)

256	`012345896ab7cefd`	8	3	`0ce6a8421d7f395b`	`t1=T(x0,x1,x2;c6); t2=T(x0,x2,t1;36); t3=T(x0,x3,t1;d2); y = (x3, t1, t2, t3)`
257	`012345896ace7bdf`	8	4	`02ce3175894d6fab`	`t1=T(x0,x2,x3;e4); t2=T(x0,x2,x3;9c); t3=T(x1,x2,x3;78); t4=T(x3,t1,t3;d2); y = (t1, t2, t3, t4)`
258	`012345678aceb9fd`	7	3	`0b812fa543c967ed`	`t1=T(x0,x1,x2;3a); t2=T(x0,x2,x3;ca); t3=T(x0,x1,x3;9c); y = (x0, t1, t2, t3)`
259	`012345786ab9cfde`	11	5	`0e938a1b4c5762fd`	`t1=T(x0,x1,x2;9a); t2=T(x0,x3,t1;78); t3=T(x2,x3,t1;8e); t4=T(x0,x1,t3;16); t5=T(x3,t1,t4;46); y = (x1, t2, t3, t5)`
260	`012345768a9cdefb`	11	5	`0615e2fda89bc473`	`t1=T(x0,x1,x3;92); t2=T(x0,x2,t1;a6); t3=T(x2,x3,t1;5c); t4=T(x0,x2,t3;9c); t5=T(x3,t3,t4;68); y = (x1, t2, t4, t5)`
261	`012345768a9cdebf`	9	4	`0b23c169ad87ef45`	`t1=T(x0,x1,x3;26); t2=T(x0,x2,t1;9a); t3=T(x0,x2,x3;ac); t4=T(x1,t2,t3;1e); y = (x0, t2, t3, t4)`
262	`012345786a9bcefd`	9	4	`0c42bf73ea68d159`	`t1=T(x0,x1,x2;ea); t2=x3^t1; t3=T(x0,x1,x3;1e); t4=T(x1,t2,t3;1e); y = (x2, t2, t3, t4)`
263	`012345768a9cdfeb`	9	4	`08e6ca2495f7d3b1`	`t1=T(x0,x1,x3;a8); t2=T(x0,x1,x2;6c); t3=T(x2,t1,t2;1e); t4=T(x0,x3,t3;1e); y = (x3, t2, t3, t4)`
264	`0123456879cefbda`	11	5	`06ea13d942c8f57b`	`t1=T(x0,x2,x3;6a); t2=T(x1,x2,x3;68); t3=T(x0,x1,t2;56); t4=T(x0,x1,t1;1e); t5=T(x1,t1,t2;b8); y = (x2, t3, t4, t5)`
265	`012345768a9cbfed`	10	5	`0e28c6a4d1379fb5`	`t1=T(x0,x2,x3;1c); t2=T(x1,x3,t1;26); t3=T(x1,x2,t1;96); t4=T(x0,t2,t3;b4); t5=T(t1,t2,t4;d8); y = (x3, t3, t4, t5)`
266	`012345768a9cbfde`	7	3	`08396e1fc4752adb`	`t1=T(x0,x1,x2;a6); t2=T(x2,x3,t1;6c); t3=T(x0,t1,t2;d2); y = (x1, t1, t2, t3)`
267	`012345768a9cbefd`	8	3	`082ae4c6197f35db`	`t1=T(x0,x1,x2;c6); t2=T(x2,x3,t1;78); t3=T(x0,x2,t2;78); y = (x3, t1, t2, t3)`

268	012345768a9cdfbe	8	3	01c723e5894b6fad	t1=T(x0,x1,x2;6a); t2=T(x1,x3,t1;78); t3=T(x0,x3,t2;c6); y = (x0, t1, t2, t3)
269	0123458967cdefba	9	4	08432a61cf97edb5	t1=T(x0,x1,x3;e8); t2=T(x0,x2,t1;6c); t3=T(x0,x1,x3;a6); t4=T(x0,x1,x3;3a); y = (t1, t2, t3, t4)
270	0123458967cefbda	11	5	0923cf61547adeb8	t1=x0^x3; t2=T(x0,x1,x2;6c); t3=T(x1,x2,x3;9e); t4=T(x0,x1,t3;26); t5=T(x0,x2,t3;98); y = (t1, t2, t4, t5)
271	0123457689ceafbd	11	4	096b2741a3ef85cd	t1=T(x0,x1,x3;70); t2=T(x1,x2,x3;36); t3=T(x1,x2,t1;58); t4=T(x2,x3,t1;46); y = (x0, t2, t3, t4)
272	0123457689ceafdb	11	5	0dc92b814fa567e3	t1=T(x0,x1,x2;9c); t2=T(x2,x3,t1;d8); t3=T(x0,x2,t2;c6); t4=T(x3,t1,t3;74); t5=T(x0,t1,t4;9c); y = (x0, t2, t4, t5)
273	0123456879acdbfe	9	4	0e681f7d2ac43b59	t1=T(x0,x1,x3;36); t2=T(x2,x3,t1;36); t3=T(x0,x1,t2;ac); t4=T(x0,x3,t2;78); y = (x2, t1, t3, t4)
274	0123456879acfedb	9	4	01ef8549ab6327cd	t1=T(x1,x2,x3;3a); t2=T(x1,x2,x3;6c); t3=T(x0,x1,t2;6c); t4=T(x3,t2,t3;36); y = (x0, t1, t3, t4)
275	0123456879acbdfe	10	4	08d724f5ca196e3b	t1=T(x1,x2,x3;56); t2=T(x0,x2,t1;6c); t3=T(x1,t1,t2;c6); t4=T(x0,t1,t3;78); y = (x1, t2, t3, t4)
276	0123456879abcefd	10	4	0c95841be2f7a63d	t1=T(x0,x1,x2;6a); t2=T(x2,x3,t1;9c); t3=T(x0,t1,t2;d2); t4=T(x1,t1,t3;36); y = (x1, t2, t3, t4)
277	0123456789cebfda	11	5	01a7832d6fc5eb49	t1=T(x0,x1,x3;e0); t2=T(x0,x1,x2;6c); t3=T(x1,x3,t2;2e); t4=T(x2,x3,t1;6e); t5=T(x2,t1,t2;d6); y = (x0, t3, t4, t5)
278	0123456789bcaefd	9	4	0c1d2efb8479a635	t1=T(x0,x1,x3;78); t2=T(x0,x2,t1;8e); t3=T(x1,x2,t1;6a); t4=T(x3,t1,t3;1e); y = (x1, t2, t3, t4)

279	012345687ab9defc	9	4	014589cfe32d67ab	t1=T(x0,x2,x3;60); t2=T(x1,x3,t1;6c); t3=T(x1,x3,t1;b4); t4=T(x2,x3,t3;78); y = (x0, t2, t3, t4)
280	0123456789bceafd	10	5	08e64ca29157f3bd	t1=T(x0,x2,x3;82); t2=T(x1,x3,t1;5c); t3=T(x1,x2,t2;a8); t4=T(x1,x2,t1;b4); t5=x0^t2; y = (x3, t3, t4, t5)
281	0123456789acbefd	10	5	04b58c1deaf96237	t1=T(x0,x1,x3;2e); t2=T(x2,x3,t1;ca); t3=T(x1,x2,t2;80); t4=T(x0,x3,t3;94); t5=T(x2,x3,t1;1e); y = (x1, t2, t4, t5)
282	0123456789acdebf	10	4	0e1f867dc2b34a59	t1=T(x0,x2,x3;c6); t2=T(x0,x1,t1;78); t3=T(x0,x3,t2;36); t4=T(t1,t2,t3;78); y = (x1, t2, t3, t4)
283	0123456879cdefab	8	4	09c54da36781ef2b	t1=T(x1,x2,x3;ca); t2=T(x0,x1,t1;1c); t3=T(x1,x2,x3;1e); t4=T(x2,x3,t2;6a); y = (x0, t1, t3, t4)
284	0123456879baefdc	8	4	083b2a156e7fc4d9	t1=T(x0,x1,x2;80); t2=T(x1,x2,x3;36); t3=x3^t1; t4=T(x0,t2,t3;d2); y = (x1, t2, t3, t4)
285	0123456789bcefda	7	3	024671b5a8ecf93d	t1=T(x0,x2,x3;1e); t2=T(x1,x2,t1;78); t3=T(x1,x3,t2;9c); y = (x2, t1, t2, t3)
286	0123456879abdefc	8	4	04ae26c83759bfd1	t1=T(x1,x2,x3;1e); t2=T(x0,x1,t1;b4); t3=T(x3,t1,t2;1c); t4=x2^t3; y = (x3, t1, t2, t4)
287	0123456789acdefb	8	4	0e4a86c2135b7df9	t1=T(x0,x2,x3;78); t2=T(x0,x1,x3;2c); t3=T(x1,t1,t2;a6); t4=T(x2,t1,t2;b4); y = (x3, t1, t3, t4)
288	0123458967cdefab	7	3	042eca86153fdb97	t1=T(x0,x1,x2;e4); t2=T(x0,x1,x2;d2); t3=T(x0,x1,x2;6a); y = (x3, t1, t2, t3)
289	0123457689cdefba	7	3	028a6ce41f9753db	t1=T(x0,x2,x3;b4); t2=T(x0,x2,x3;6c); t3=T(x0,x1,t2;6c); y = (x3, t1, t2, t3)
290	0123456789cdefba	7	3	0e862ca4fb731d95	t1=T(x0,x2,x3;36); t2=T(x0,x3,t1;78); t3=T(x0,x1,t2;36); y = (x3, t1, t2, t3)

291	`012345768a9bcfed`	8	4	`01458dc923af6be7`	`t1=T(x0,x2,x3;b8); t2=T(x0,x1,x3;78); t3=T(x1,t1,t2;d6); t4=T(x2,t1,t2;b6); y = (x0, x3, t3, t4)`
292	`012345768a9bcefd`	6	2	`0185cd4923af67eb`	`t1=T(x0,x1,x2;6a); t2=T(x1,x3,t1;d2); y = (x0, x3, t1, t2)`
293	`0123456789cefbda`	9	4	`04d18c95263baef7`	`t1=T(x0,x2,x3;36); t2=T(x0,x3,t1;78); t3=T(x0,x1,t1;74); t4=T(x1,x2,t2;5c); y = (x1, x3, t3, t4)`
294	`0123456789acbdfe`	7	3	`01c98d45a76b23ef`	`t1=T(x1,x2,x3;36); t2=T(x0,x1,t1;6c); t3=T(x3,t1,t2;6c); y = (x0, x3, t2, t3)`
295	`0123456789acdbfe`	6	3	`01452b67cd89a3ef`	`t1=T(x0,x1,x2;20); t2=T(x1,x2,x3;d2); t3=x3^t1; y = (x0, x2, t2, t3)`
296	`0123456789abcefd`	7	3	`058923abc14def67`	`t1=~x2&x0; t2=T(x1,x3,t1;c6); t3=T(x1,t1,t2;d2); y = (x0, x2, t2, t3)`
297	`0123456789baefdc`	5	2	`01c9458d2be36fa7`	`t1=T(x0,x1,x2;a6); t2=T(x0,x1,x3;6c); y = (x0, x3, t1, t2)`
298	`0123456789cdefab`	5	2	`0985cd412ba7ef63`	`t1=T(x0,x1,x2;6a); t2=T(x0,x1,t1;36); y = (x0, x3, t1, t2)`
299	`0123456789abdefc`	5	2	`01e34527896bcdaf`	`t1=T(x0,x1,x2;a6); t2=T(x2,x3,t1;c6); y = (x0, x1, t1, t2)`
300	`0123456789abcdfe`	4	2	`01234def89abc567`	`t1=T(x0,x1,x2;bc); t2=T(x2,x3,t1;6c); y = (x0, x1, x2, t2)`
301	`0123456789abdcfe`	3	1	`08192a3b4cd56ef7`	`t1=T(x0,x1,x3;78); y = (x1, x2, x3, t1)`
302	`0123456789abcdef`	0	0	`0123456789abcdef`	`y = (x0, x1, x2, x3)`

Appendix D: search details

This appendix specifies the searches in enough detail to reimplement them. Everything works on truth tables, so nothing depends on an instruction set beyond the definitions in D.1.

D.1 Conventions

- **Truth tables.** A function of the input x = x0 + 2x1 + 4x2 + 8x3 is a 16-bit integer whose bit x is its value. The inputs are x0 = 0xaaaa, x1 = 0xcccc, x2 = 0xf0f0, x3 = 0xff00. An S-box is given as S(0)..S(15), and output j is the table of bit j of S(x).
- **Gates.** Bit t of `T(a, b, c; imm)` is bit $4a_t + 2b_t + c_t$ of imm, as in `vpternlogd` with a as its first operand. The gate depends on a if flipping bit 2 of the index changes imm for some index, and

likewise on b (bit 1) and c (bit 0).

- **Normalisation.** norm(v) is v if bit 0 of v is 0, else ~v. Complementing a gate's result is `imm ^= 0xff`, and complementing an operand permutes imm (flips one index bit), so programs can be searched on normalised values and the polarities fixed afterwards.
- **Atoms.** For values a, b, c the eight atoms are m_p = (a or ~a) & (b or ~b) & (c or ~c), taking a when bit 2 of p is set, b for bit 1 and c for bit 0. A table t is a function of (a, b, c) iff every atom is inside t or disjoint from it. The normalised functions of a triple are the unions of non-empty sets of the non-empty atoms that do not contain row 0. Such a function depends on a iff for some p without bit 2 both m_p and $m_{p|4}$ are non-empty and it contains exactly one of them; likewise for b and c.
- **Native gate.** A gate that depends on exactly two operands p, q and computes, on actual (not normalised) values, a function F(p, q) with F(0, 0) = 0: and, or, xor, ~p&q or p&~q.
- **Depth and objective.** Input i has depth A[i], and a gate is one deeper than the deepest operand it depends on. The objective is $\max_j(\text{depth}(y_j) + T[j] + P[j])$ with P[j] = 1 if folding is on and y_j is a gate that is not native, else 0. Encryption: A = (3,0,3,1), T = (2,1,2,0), no folding. Decryption: A = (0,0,0,1), T = (1,0,0,0), folding.

D.2 SAT instance for k gates

Nodes 0..3 are the inputs and node n = 4 + g is gate g. Variables $v_{n,t}$ hold bit t of the normalised value of node n (constants for the inputs). For gate g:

- selectors a_x (x = 0..n−3), b_x (1..n−2) and c_x (2..n−1), exactly one true per slot (one clause and pairwise exclusions), with $\neg a_x \vee \neg b_y$ and $\neg b_x \vee \neg c_y$ for y ≤ x, so the operands are distinct and increasing;
- operand values $o_{s,t}$ per slot s: $\neg s_x \vee \neg v_{x,t} \vee o_{s,t}$ and $\neg s_x \vee v_{x,t} \vee \neg o_{s,t}$;
- eight variables $i_{0..7}$ for imm, and for each row t and index p two clauses: if ($o_{a,t}$, $o_{b,t}$, $o_{c,t}$) are the bits of p, then $v_{n,t} = i_p$ (16 clauses per row);
- $v_{n,0}$ false and $v_{n,1} \vee \ldots \vee v_{n,15}$; for each input, one clause over rows 1..15 saying that the gate differs from it; for each earlier gate m, variables d_t with $d_t \rightarrow v_{n,t} \neq v_{m,t}$ and $d_1 \vee \ldots \vee d_{15}$.

Across gates: for g ≥ 1 and y < x, the clause $\neg a^{g-1}_x \vee \neg a^g_y \vee c^g_{n-1}$: if gate g does not take gate g−1 as an operand, its first operand is not below that of gate g−1. Every gate is selected by a later gate or is an output. Output j has one indicator per candidate node, exactly one of them true. The indicator of a gate forces its value to norm(target j); an input is a candidate only if it equals target j exactly.

Decoding reads the operands and imm from the model. A gate that is output j holds the complement of target j when bit 0 of the target is set, so its imm is inverted, and every gate that reads it flips the matching index bit. Other gates may take either polarity, and the decoder picks the polarities with the most native gates (all choices are tried for up to 16 such gates). The imm bits of operand combinations that never occur are free; they are set to make the gate native if possible, else to depend on fewer operands, and never so that a dependency is added. The decoded program is simulated and checked.

D.3 Gate count of an S-box and of a class

For an S-box, k runs 1, 2, … and the first satisfiable k is the minimum. This needs a minimal program to satisfy the instance after rewriting it. In a minimal program no gate is unused, constant, equal to an input or equal to another gate up to complement, since such a gate could be removed. Normalising the values and sorting each gate's operands (a permutation of imm; a gate with fewer than three dependencies takes other earlier nodes as dummy operands) keeps it a program, and D.5 reorders the gates to meet the ordering clauses.

For a class, the instance also has variables for A and B (4×4 bits each). For each row t it computes $y = A \cdot x$ (XOR chains), $z = S(y)$ (for each v and output bit j, a clause: if $y = v$ then z_j is bit j of $S(v)$) and $w = B \cdot z$. All 16 values w(t) must differ, which makes A and B invertible. The columns of A, read as 4-bit numbers, increase strictly; the input constant is 0, and the output constant is free, so target j at row t is $w_j(t) \oplus w_j(0)$. Inputs may be outputs, and output $j + 1$ is at a larger node than output j. These restrictions lose nothing: column order relabels the inputs, an input constant complements inputs (absorbed by imm, or by the output constant for an output that is an input), and output order is a row permutation of B. The lower bound is $L = 4 - d$, where $2^d - 1$ of the 15 components $u \cdot S$ are affine (algebraic degree at most 1). Only those can become inputs under some A, and every other output needs its own gate.

The class search started at the gate count of the representative itself and went down while the instance was satisfiable. Every count from L to one below the minimum is unsatisfiable (where the descent stopped above L, the remaining counts were run separately) or, for the 245 classes without affine components, excluded by the exhaustive search of D.6. A solution is checked by simulation, and by brute force over A and the input constant, with B and the output constant following from the values.

D.4 Depth bound in the SAT instance

For an objective bound D, each gate has variables $e_{0..D}$ meaning "depth ≤ d", with e_0 false, e_D true and $e_d \rightarrow e_{d+1}$. A variable q_s per slot must be true when imm depends on the slot: for each pair of indices that differ only in the slot's bit, unequal imm bits imply q_s. For the node x selected in slot s and $d \geq 1$, $e_d \wedge q_s \wedge s_x$ implies $\text{depth}(x) \leq d - 1$; for an input this is false when $d - 1 < A[x]$. Output j at a gate requires $e_{D-T[j]}$ of that gate (and cannot be a gate if $D - T[j] < 1$).

With folding, each node has a polarity bit φ (0 for inputs), and its actual value is the normalised value XOR φ. Each gate has a native indicator together with a choice of operand pair, (a,b), (a,c) or (b,c), and three free bits F(0,1), F(1,0), F(1,1). If the indicator is set, every row must satisfy actual(gate) = F(actual(p), actual(q)) with F(0,0) = 0. Values are distinct and not constant, so this is exactly a native gate. Output j at gate n requires φ_n to be bit 0 of target j, and either the native indicator or $e_{D-T[j]-1}$.

At a fixed k, D starts one below the objective of a program already found and decreases until the instance is unsatisfiable. At the final D the number of native gates is then raised the same way (at least m native indicators, with a sequential counter); this only chooses between programs with the same objective.

D.5 Gate order, and what the restrictions keep

Each search keeps adjacent gates g − 1 and g in order when gate g can be computed without gate g − 1. The SAT instance compares their first operands. The exhaustive searches compare their values, and in D.7 apply the rule only when gate g reaches the same depth without gate g − 1 and, for an output, keeps all its native options. Any program can be brought into this order. Wherever the rule is violated, swap the two gates and compute gate g from operands before gate g − 1. Values and outputs stay the same, no depth or native option gets worse, and later gates are only renumbered. The keys before the pair are unchanged and the key at position g − 1 decreases, so the sequence of keys decreases lexicographically and the swapping stops.

The other restrictions lose nothing at the minimum gate count (D.3). Above it, the SAT instance for exactly k gates and the depth search of D.7 still require distinct values. Without folding, a repeated value is never needed, because a program with fewer gates does as well. With folding, a program could compute a value in both polarities so that two outputs become native. Such programs are not searched, so the decryption results with extra gates (section 3.3 and appendix A.2) hold among programs with distinct values.

D.6 Exhaustive search over a class

```
for each A with columns c0 < c1 < c2 < c3 in 1..15, linearly independent   # 840
    T[u] = norm(table of parity(u & S(A x)))   for u = 1..15
    search(vals = [x0, x1, x2, x3], nout = 0, span = [0], ng = 0)

search(vals, nout, span, ng):
    if nout == 4: report the program; return
    if K - ng < 4 - nout: return
    last = vals[-1]
    for u in 1..15 with u not in span and T[u] not in vals:          # output gate
        if T[u] is a function of some triple of vals:
            if ng > 0 and T[u] < last and T[u] is a function of a triple without last:
                continue
            search(vals + [T[u]], nout + 1, span + [s ^ u for s in span], ng + 1)
    if K - ng - 1 < 4 - nout: return                                  # hidden gate
    for v in the normalised functions of all triples of vals, each once, v not in vals:
        if ng > 0 and v < last and v is a function of a triple without last:
            continue
        search(vals + [v], nout, span, ng + 1)
```

A triple is three distinct entries of vals, and values are compared as unsigned 16-bit integers. The program computes the member B·S(A·x) ⊕ b, where the rows of B are the chosen u and b is free. Gates need not be used, so finding nothing proves that no member has a program of K gates or fewer. A gate whose value is a component is tried both as an output and as a hidden gate. A class in which some u·S is affine is not handled, since that output would be an input; checking this at A = I suffices, because a component stays affine under any A. With A fixed to the identity and u restricted to 1, 2, 4, 8 (output j is bit j of S, up to complement), the same search gives the gate count of a single S-box. Each program found is simulated, and the canonical form (D.8) of its member is compared with that of the class.

D.7 Exhaustive depth search for one S-box

This has the same structure as D.6, with A the identity and the four outputs fixed: the value of output j is norm(output j's table), and its actual polarity is kept for the native test. Hidden gates may take either polarity. Each value carries a depth: input i has A[i], and any other value has the minimum, over all triples that compute it, of one plus the largest depth among the operands it depends on (computed from the atoms, D.1).

```
output gate j (not placed yet, its value not in vals):
    dall = min depth over all triples computing it;  dold = the same without last
    natd = min of 1 + max(depth p, depth q) over pairs p < q of vals for which
           target j = and/or/xor/andnot of actual(p), actual(q),
           for some polarity of the hidden gates among p, q
    cost = folding ? min(dall + 1, natd) : dall
    skip if cost + T[j] > D
    skip if ng > 0 and dold == dall and value < last
            and not (folding and one of those pairs contains last)

hidden gate v (a normalised function of a triple, with depth <= D - 1 - min(T)):
    da = min depth over all triples;  do = the same without last
    skip if v in vals, or if ng > 0 and do == da and v < last

complete program (4 outputs placed):
    for each polarity choice of the hidden gates (a single choice without folding):
        objective = max over outputs j at node o of
            T[j] + (folding ? min(depth(o) + 1, min over pairs p < q < o with
                                    target j = native op of actual(p), actual(q)
                                    of 1 + max(depth p, depth q))
                            : depth(o))
        accept if objective <= D
```

A hidden gate deeper than D − 1 − min(T) cannot feed an output within the bound, and since unused gates are allowed it can be left out. Before the leaf the pruning uses only lower bounds of the final cost; the leaf computes the cost exactly. For parallel runs, the states after two gates are collected first (about 218,000 for the 8-gate searches) and searched independently. Each program found is checked against the objective of D.1 on its actual polarities.

D.8 Canonical form and representatives

The canonical form of S is the lexicographically smallest B·S(A·x ⊕ a) ⊕ b over all 20160 invertible A, all 16 constants a and all B and b. For fixed A and a, the best B and b are found greedily. XOR every value with S(a), so that x = 0 maps to 0. Then go through x = 1..15, and map each value that is not in the span of the earlier ones to the next power of two (1, 2, 4, 8), extending linearly. This gives the smallest possible value at each position in turn. The 30 representatives missing from Ullrich et al. are canonical forms of random permutations, drawn until all 302 classes had appeared.