FSBday:

Implementing Wagner's Generalized Birthday Attack against the round-1 SHA-3 Candidate FSB

Dan Bernstein, Tanja Lange, Ruben Niederhagen, Christiane Peters and Peter Schwabe

Eindhoven University of Technology
University of Illinois at Chicago
RWTH Aachen University

December 15, 2009

INDOCRYPT 2009

Wagner's generalized birthday attack

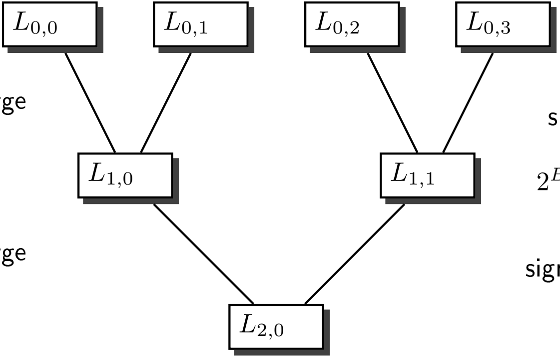
Given 2^{i-1} lists containing B-bit strings.

Generalized birthday problem:
The 2^{i-1}-sum problem consists of finding 2^{i-1} elements — exactly one per list — such that their sum equals 0 (bitwise modulo $2 \Rightarrow$ xor).

Wagner (CRYPTO '02)

We can expect a solution to the generalized birthday problem after one run of an algorithm using time $O((i-1) \cdot 2^{B/i})$ and lists of size $O(2^{B/i})$.

Wagner's tree algorithm



$2^{B/3}$ elements

1. merge

compare on least significant $B/3$ bits

$2^{B/3}$ elements of xors

2. merge

compare on least significant $2 \times B/3$ bits

Expect to get 1 match after the last merge step.

Tree algorithm for 2^{i-1} lists

The tree algorithm generalizes to 2^{i-1} lists as follows:

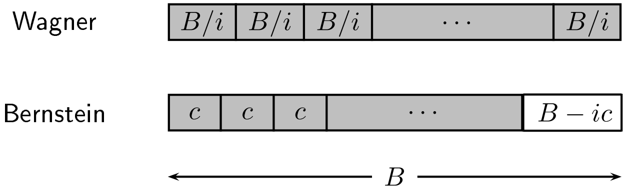
- Compare lists — always two at a time — by looking at the least significant B/i bits of elements.
- On level $i-2$ we are left with two lists whose elements need to be compared on $2B/i$ remaining bits.

Precomputation step

Suppose that there is space for lists of size only 2^c with $c < B/i$.

Bernstein (SHARCS '07):

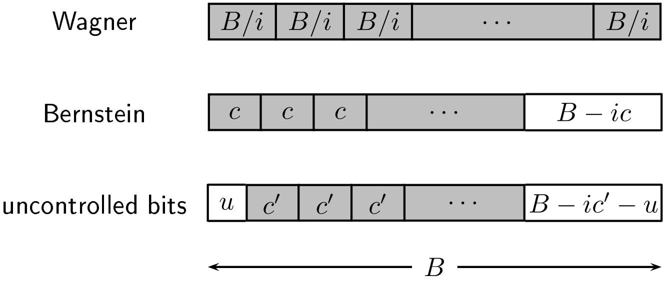
- Generate $2^{c \cdot (B-ic)}$ entries and only consider those of which the least significant $B - ic$ bits are zero.
- Then apply Wagner's algorithm with lists of size 2^c and clamp away c bits on each level.



Repeating (parts of) the tree algorithm

- When performing the algorithm with smaller lists, u bits remain "uncontrolled" at the end.

- Deal with the lower success probability by repeatedly running the attack with different clamping values.

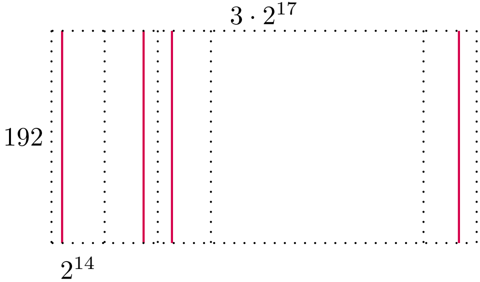


Target: the compression function of FSB_{48}

Given a binary random 192×393216 matrix H; number of blocks: $w = 24$.

Input: a regular weight-24 bit string of length 393216, i.e., there is exactly a single 1 in each interval $[(i-1) \cdot 16384, i \cdot 16834]_{1 \leq i \leq 24}$.

Output: Xor the 24 columns indicated by the input bit string.



Goal: Find a collision in FSB_{48}'s compression function; i.e., find 48 columns — exactly 2 per block — which add up to 0.

Applying Wagner to FSB_{48}

Determine the number of lists for a Wagner attack on FSB_{48}.

- We choose 16 lists to solve this particular 48-sum problem. (16 is the highest power of 2 dividing 48).
- Each list entry will be the xor of three columns coming from one and a half blocks (no overlaps!).
  $\rightarrow$ We can generat at most 2^{40} elements per list.

Straightforward Wagner

- Applying Wagner's attack with 16 lists in a straightforward way means that we need to have at least $2^{\lceil 192/5 \rceil}$ entries per list.
- By clamping away 39 bits in each step we expect to get at least one collision after one run of the tree algorithm.

List entries

- Reduce amount of data by clamping away 2 bits $\Rightarrow 2^{38}$ entries per list (clamp 38 bits on each level)
- Ultimately we are not interested in the value of the entry; but in the column positions in the matrix that lead to this all-zero value.
  - Value-only representation
  - Positions-only representation: keep full positions; if we we need the value (or parts of it) it can be dynamically recomputed from the positions.
- Note: Unlike storage requirements for values the number of bytes for positions increases with increasing levels.

Storing positions

- Encode column positions of each entry in 40 bits (5 bytes) for the first level.
- The expected number of entries per list remains the same but the number of lists halves; so the total amount of data is the same on each level when using dynamic recomputation.
- Storing 16 lists with 2^{38} entries, each entry encoded in 5 bytes requires 20480 GB of storage space.
- The Coding and Cryptography Computer Cluster at Eindhoven University of Technology only has a total hard disk space of about 5440 GB, so we have to adapt our attack strategy to this limitation.

Adapt attack strategy

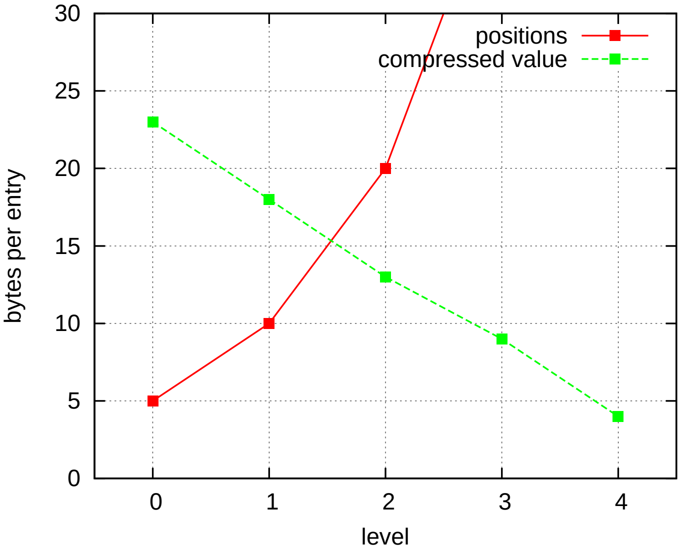
- Can handle at most $5\ TB/16\ lists/5 = 2^{36}$ entries per list.
- A straightforward implementation would use lists of size 2^{36}: clamp 4 bits during list generation; this leads to 2^{36} values for each of the 16 lists (as we can generate at most 2^{40} elements per list).
- We expect to run the attack 256.5 times until we find a collision.

Attack in two phases

Idea

- First phase: Figure out which clamping constants yield collision
- Second phase: Compute matrix positions yielding collision
- During phase one we do not have to store positions of entries
- On each level compress entries to shortest possible representation

Attack in two phases

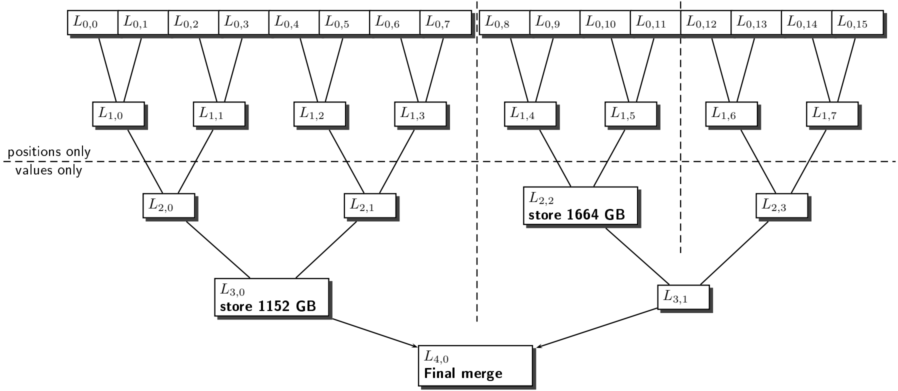


Attack in two phases

Idea

- First phase: Figure out which clamping constants yield collision
- Second phase: Compute matrix positions yielding collision
- During phase one we do not have to store positions of entries
- On each level compress entries to shortest possible representation
  - Level 0: 5 bytes (positions only)
  - Level 1: 10 bytes (positions only)
  - Level 2: 13 bytes (values only)
  - Level 3: 9 bytes (values only)
- Use lists of size 2^{37}
- Clamp 3 bits through precomputation
- This leaves 4 bits "uncontrolled" $\rightarrow$ 16.5 repetitions expected

Attack Strategy



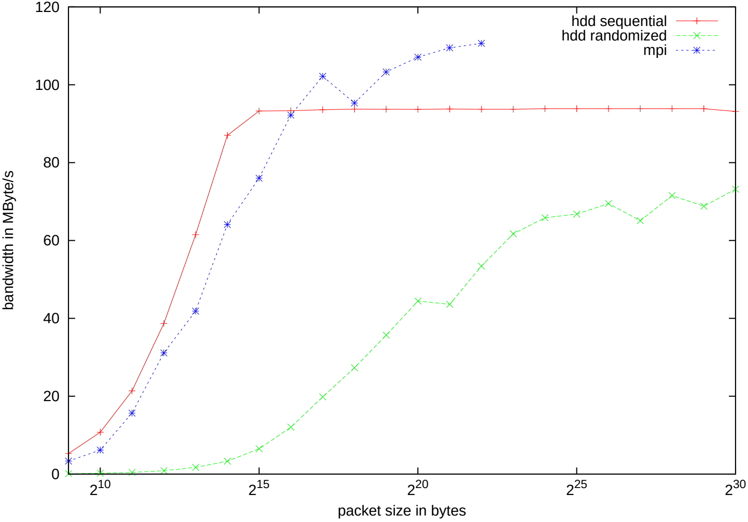
$$\Longrightarrow 1152\text{ GB} + 1664\text{ GB} + 2560\text{ GB} = 5376\text{ GB}$$

Our hardware

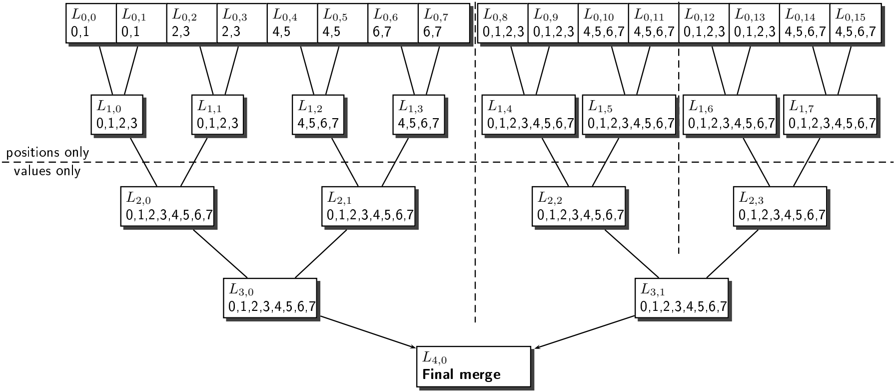
Cluster of 8 nodes:

- Intel Core 2 Quad Q6600 CPU, 2.40 GHz
- 8 GB of RAM per node
- about 680 GB accessible mass storage
- connected via switched Gigabit Ethernet

Finding the bottleneck(s)



Parallelization



Parallelization

- Split fractions further into 512 parts of 640 MB each (presort, according to 9 bits)
- Sort and merge parts independently in memory
- Pipeline
  - Loading from hard disk into memory
  - Sorting of two parts
  - Merging of previously sorted parts
- Requires 6 parts in memory at the same time (3.75 GB)

Parallelization

- Split fractions further into 512 parts of 640 MB each (presort, according to 9 bits)
- Sort and merge parts independently in memory
- Pipeline
  - Loading from hard disk into memory
  - Sorting of two parts
  - Merging of previously sorted parts
- Requires 6 parts in memory at the same time (3.75 GB)
- Two blocks of operations:
  - Load, Sort, Merge, Send
  - Receive, Presort, Store

Timing Results

- Timings for phase 1:
  - Computation of list $L_{3,0}$: ~ 32 h (once)
  - Computation of list $L_{2,2}$: ~ 14 h (once)
  - Computation of list $L_{2,3}$: ~ 14 h (exp. 16.5×)
  - Computation of list $L_{3,1}$: ~ 4 h (exp. 16.5×)
  - Check for collision in $L_{3,0}$ and $L_{3,1}$: ~ 3.5 h (exp. 16.5×)
- Expected time for phase 1: $32 + 14 + 16.5 \cdot (14 + 4 + 3.5) =$ 400.7 h or 17 days
- Time for phase 2: ∼33 h per half-tree, in total ∼66 h
- Expected time in total: ∼19.5 days.

Result

We already found a solution in step one after only five iterations!

In total the attack took 7 days, 23 hours and 53 minutes.

The result:

734, 15006, 20748, 25431, 33115, 46670, 50235, 51099, 70220, 76606, 89523, 90851, 99649, 113400, 118568, 126202, 144768, 146047, 153819, 163606, 168187, 173996, 185420, 191473 198284, 207458, 214106, 223080, 241047, 245456, 247218, 261928, 264386, 273345, 285069, 294658, 304245, 305792, 318044, 327120, 331742, 342519, 344652, 356623, 364676, 368702, 376923, 390678

Further information

Paper: http://eprint.iacr.org/2009/292

Cluster: http://www.win.tue.nl/cccc/

Code: http://www.polycephaly.org/fsbday/
(available under public domain)