# Syndrome-based hash functions

Tanja Lange

Technische Universiteit Eindhoven

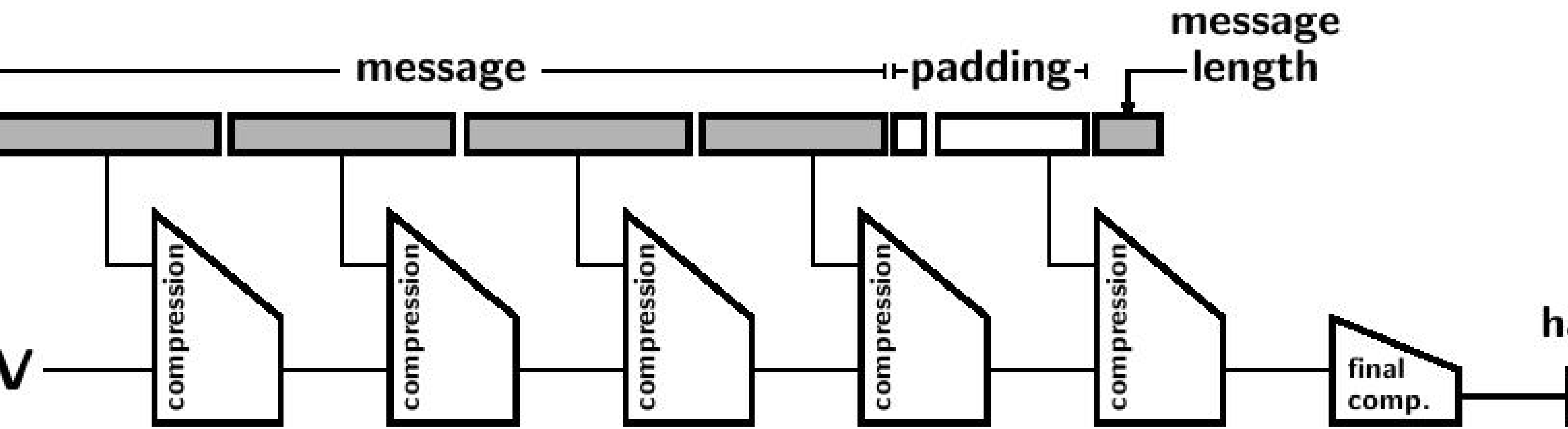
tanja@hyperelliptic.org

31 May 2011

joint work with Daniel J. Bernstein, Christiane Peters, and Peter Schwabe

# Syndrome-based hashing

- Build hash function using a compression function with Merkle-Damgård chaining and final compression.




Picture from submission document of FSB hash function to SHA-3 competition by Augot, Finiasz, Gaborit, Manuel, and Sendrier.

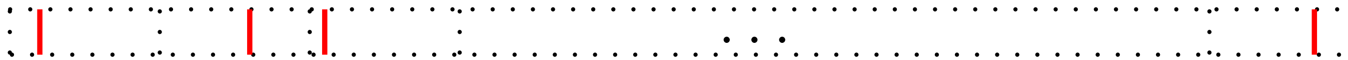
- The compression function is based on a matrix-vector multiplication.

# Compression function

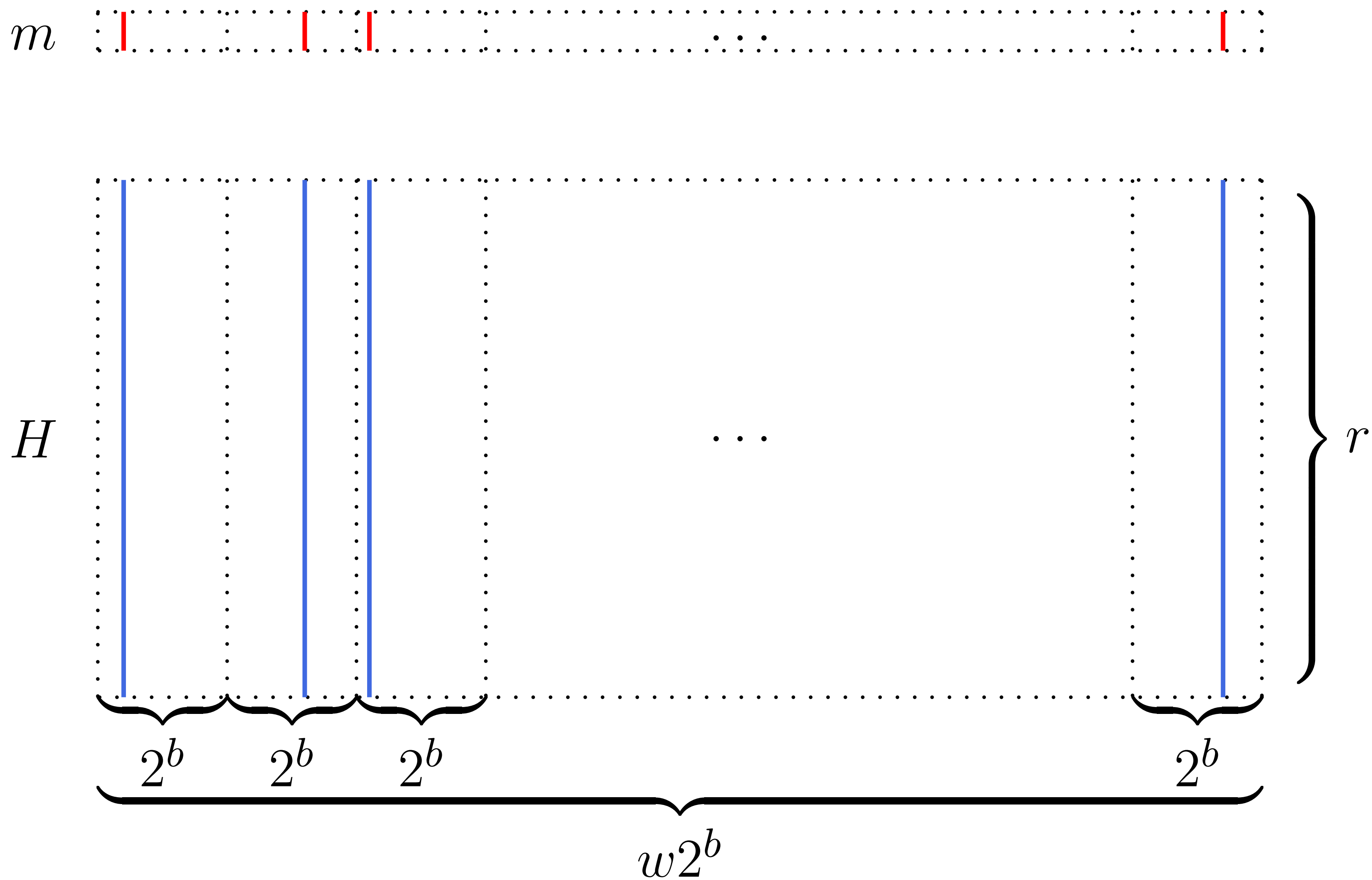
- Take random $r \times w2^b$ matrix $H$.
- Consider the matrix portioned into $w$ blocks, each containing $2^b$ columns.
- Process $wb$ bits at once as $m = (m_1, m_2, \ldots, m_w)$, considering $b$ bits as one number in $\{0, 1, \ldots, 2^b - 1\}$.
- Define the compression function as the sum of $w$ columns of $H$ by taking the $m_i$th column in the $i$th block.
- This computation can be seen as the multiplication $Hm'$, where $m'$ is a regular word (a word having exactly 1 nonzero entry in every block of length $w$) obtained from $m$ by putting a $1$ into position $m_i$ in the $i$th block. The result of the compression is the syndrome of $m'$ (considering $H$ as a parity check matrix), hence the name “syndrome-based hash functions”.

# Picture

$m$



# Picture



# Security of hash function

- Finding preimages = syndrome-decoding with extra requirement that error has weight $w$ and is regular.
- Finding collisions = finding 2-regular words of weight $2(w-i)$ with $0$ or $2$ entries for each of the $w$ blocks.
- Requirement of regularity makes both problems significantly harder – there are many words of weight $2(w-i)$ for small enough $i$ but they are unlikely to be 2-regular; vice versa, a 2-regular word of weight $2(w-i)$ is unlikely to produce syndrome 0.
- Will describe linearization attacks, generalized birthday attacks, and information-set decoding attacks soon.

# History

- Zobrist in 1970 introduced compression function $(m_1, m_2, \ldots, m_w) \mapsto c_1[m_1] \oplus c_2[m_2] \oplus \cdots \oplus c_w[m_w]$, with random choices of matrix entries $c_i[j]$.
- Same compression function reintroduced and discarded in a Eurocrypt 1997 paper by Bellare and Micciancio.
- Very similar compression function with limited $w$ introduced by Damgård at Crypto 1989; he used addition rather than $\oplus$, took $w = 256$ for $128$-bit output (or more generally $w \approx 2r$ for $r$-bit output), and $m_i \in \{0, 1\}$.
- Augot, Finiasz, and Sendrier in 2003 were first to consider limited $w$ and several bits in each $m_i$ in initial FSB (Fast Syndrome-Based hashing) version.
- In 2007, Finiasz, Gaborit, and Sendrier reduced storage space for $H$ by using a quasi-cyclic matrix.

# FSB

- Fast Syndrome-Based Hashing introduced by Augot, Finiasz, and Sendrier in 2003.
- Several intermediate versions and attacks.
- Submitted to SHA-3 competition in 2008; using truncated quasi-cyclic matrix $H$. This means that $(r+\epsilon)\times 2^b$ (for small $\epsilon$) entries determine all of $H$.
- Security estimated to be exponential in $r$, matrix contains $\Theta(r)$ bits.
- Attacks focus on collision finding in compression function; thus ignore IV and chaining values. If finding collisions without any restrictions is hard, it is even harder to find collisions with fixed IV bits.

# Linearization attacks I

- Initially suggested as preimage attacks by Bellare & Micciancio; extended by Saarinen and also generalized to collision attacks. Hybrid approach due to Bernstein-L-Peters-Schwabe [BLPS11b].
- Target $h$.
- Pick randomly $m_1, m_2, \ldots, m_w, m_1', m_2', \ldots, m_w'$, and compute $\delta_1 = c_1[m_1] \oplus c_1[m_1'], \ldots, \delta_w = c_w[m_w] \oplus c_w[m_w']$, and $\Delta = h \oplus c_1[m_1] \oplus c_2[m_2] \oplus \cdots \oplus c_w[m_w]$.
- Use linear algebra to find linear relation $\epsilon_1\delta_1 \oplus \cdots \oplus \epsilon_w\delta_w = \Delta$, if such a relation exists.
- $c_1[m_1 \oplus \epsilon_1(m_1' \oplus m_1)] \oplus \ldots \oplus c_w[m_w \oplus \epsilon_w(m_w' \oplus m_w)] = h$.
- Probability of success $2^w/2^r$.

# Linearization attacks II

- Saarinen suggests to consider multiple differences $\delta'_1 = c_1[m_1] \oplus c_1[m''_1], \ldots, \delta'_w = c_w[m_w] \oplus c_w[m''_w]$ to increase chance of solvability while not massively increasing the chance of useless collisions (using $\delta_i$ and $\delta'_i$). If $w < r/2$ then probability $3^w/2^r$.
- In general, for $w < r/k$, compute $k$ differences for each $i$; probability $((k+1)/2^k)^w$.
- For finding collisions pick randomly $m_1, \ldots, m_w, n_1, \ldots, n_w, m'_1, \ldots, m'_w, n'_1, \ldots, n'_w$, compute $\Delta = c_1[m_1] \oplus c_1[n_1] \oplus c_2[m_2] \oplus c_2[n_2] \oplus \cdots \oplus c_w[m_w] \oplus c_w[n_w]$, $\delta_i = c_i[m_i] \oplus c_i[m'_i]$, and $\delta'_i = c_i[n_i] \oplus c_i[n'_i]$. Use linear algebra to find a subset of $\delta_1, \delta'_1, \ldots, \delta_w, \delta'_w$ with sum $\Delta$.
- For $w \le r/(2k)$ using $2k$ generators for each $i$ gives probability $(k+1)^{2w}/2^r$. Hybrid approach possible.

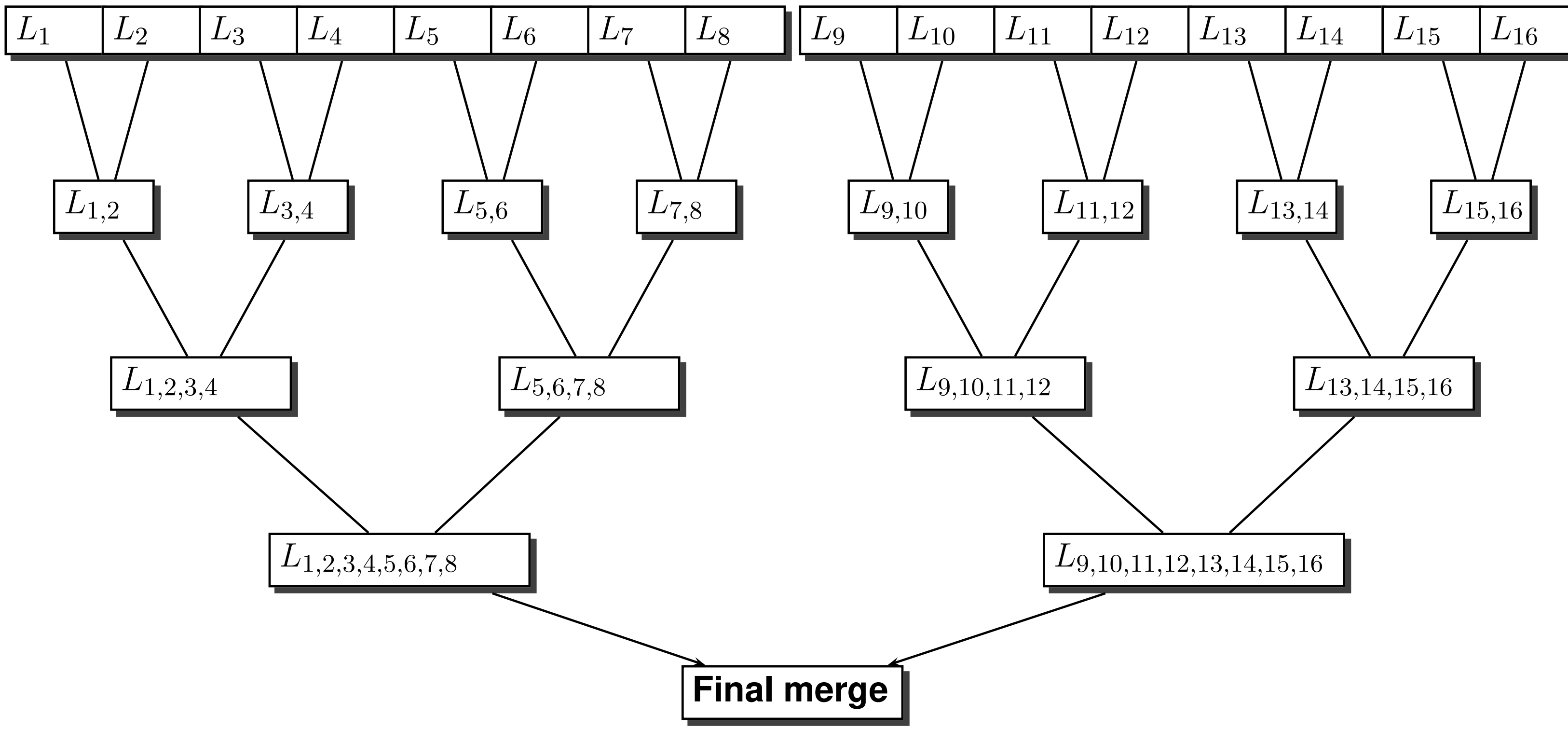
# Generalized birthday attacks I

- The $k$-sum problem is to find $x_1 \in L_1, \ldots, x_k \in L_k$ such that $x_1 \oplus x_2 \oplus \cdots \oplus x_k = 0$, given $k$ lists $L_1, \ldots, L_k$ of $r$-bit strings drawn uniformly and independently at random.
- If $k = 2^{i-1}$ and each list has $2^{r/i}$ elements then generalized birthday attacks solve this problem using $O(k \cdot 2^{r/i})$ operations.
- Merge $L_1$ and $L_2$: keep sums $u \oplus v, u \in L_1, v \in L_2$ that are $0$ on first $r/i$ bits. Store these in new list $L_{1,2}$.
- Expected number of elements in $L_{1,2}$ is again $2^{r/i}$.
- Same for $L_{3,4}$ from $(L_3, L_4)$, ..., $L_{k-1,k}$ from $(L_{k-1}, L_k)$.
- Leads to $2^{i-2}$ lists of expected length $2^{r/i}$ containing $r$-bit strings with their first $r/i$ bits zero.

# Generalized birthday attacks II

- On next level merge $L_{1,2}$ and $L_{3,4}$; elements in $L_{1,2,3,4}$ are sums $u \oplus v, u \in L_{1,2}, v \in L_{3,4}$ that are $0$ on their first $2r/i$ bits.
- Same for $L_{5,6,7,8}$ from $(L_{5,6}, L_{7,8})$, . . . $L_{k-3,k-2,k-1,k}$ from $(L_{k-3,k-2}, L_{k-1,k})$.
- Continue for $i - 2$ levels, obtain $L_{1,\dots,k/2}$ and $L_{k/2+1,\dots,k}$, each containing an expected number of $2^{r/i}$ strings that are $0$ on their first $(i-2)r/i$ bits.
- Compute all sums $u \oplus v$ with $u \in L_{1,\dots,k/2}$ and $v \in L_{k/2+1,\dots,k}$ to see, on average, one element with all $r$ bits zero $\Rightarrow$ solution!

# Tree with $k$ =16




Each list needs to contain $2^{r/5}$ entries to get one 0-string of length $r$.

# Application to hash functions

- Initially ruled out as attack since attack needs $2^{r/(1+\lfloor \lg k \rfloor)}$ entries per list and $2b-1$ is usually much smaller than $r/(1+\lfloor \lg k \rfloor)$, so $k=w$ does not work. (The $2b-1$ takes care of all combinations of 2 columns per block.)
- Coron & Joux take $k$ smaller than $w$ and combine columns from multiple blocks, 2 at a time.
- Submission to SHA-3 competition put a lot of armor against generalized birthday attacks.
- Attack against compression function of FSB$_{48}$, a toy version of FSB with $r=192, b=14, w=24$, used sums of 3 columns from 1.5 blocks.

# Attack on $\mathrm{FSB}_{48}$

- Can make lists smaller and run attack several times (clamp away some bits when building lists).
- In [BLNPS09] we describe our implementation of this attack on our university cluster. Lots of cool tricks, careful study of communication involved, new lean file system – lists did fit but attack took 5 days, 13 hours, and 20 minutes using essentially the entire 8·750GB hard disks of 8 computers.
- Attacking $\mathrm{FSB}_{48}$ with a generic Pollard-rho attack took 1 minute and 20 seconds and close to no memory.
- Conclusion: The designers overshot – taking into account the memory and communication requirements of the attack much smaller parameters would suffice to provide security.

# Information-set decoding

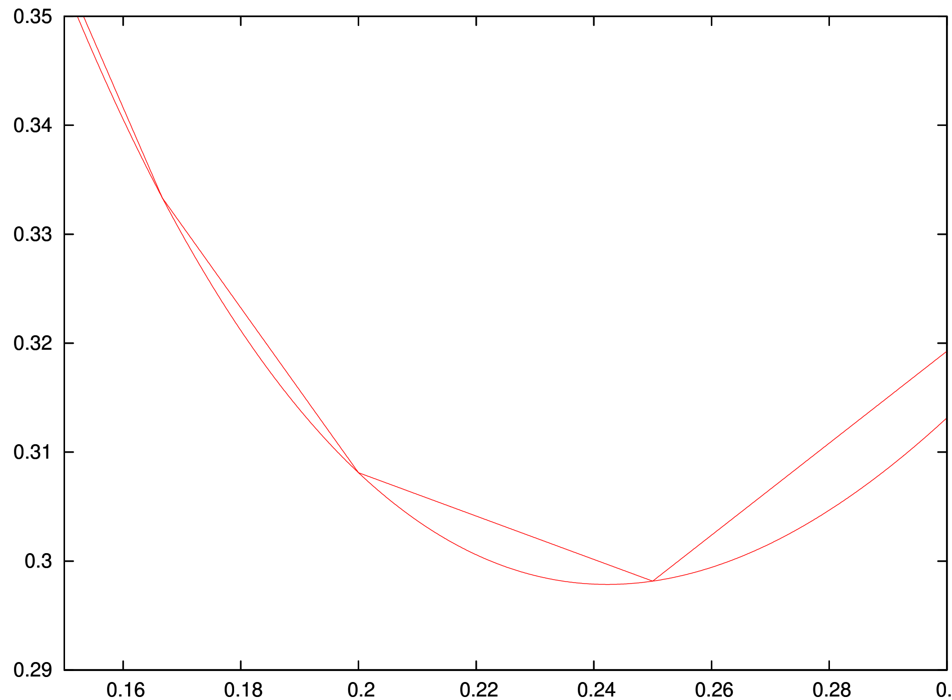
- Previous talk showed how to find error vectors of small weight using information-set decoding.
- We are looking for a vector of weight $2(w-i)$ and syndrome 0.
- In addition the vector should be 2-regular.
- Weight $2(w-i)$ is not actually small, so likely to find vectors of this weight quickly – but unlikely that they are regular.
- Will first show and analyze plain information-set decoding attack (a la Prange), then show our improvements.

# Augot-Finiasz-Sendrier 2003

- ISD against syndrome-based hash functions:
  1. Pick $w_0 \leq w$. Choose $r$ columns of the $r \times w2^b$ matrix $H$, by choosing evenly among $w_0$ blocks of $H$.
  2. Perform Gaussian elimination on the $r \times r$ submatrix; check each kernel element for 2 regularity.
  3. If no 2-regular kernel element was found, return to step 1.
- Only $\sum_{1 \leq j \leq w_0} \binom{r}{2j}$ vectors have weight $2(w-i) \leq 2w_0$; only $\left(\binom{r/w_0}{2} + 1\right)^{w_0} - 1$ are nonzero $2$-regular vectors.
- Probability $1/2^r$ of being a codeword.
- Expected number of codewords per iteration: $\left(\left(\binom{r/w_0}{2} + 1\right)^{w_0} - 1\right) / 2^r$.

# Some nitpicks

- What if $w_0$ does not divide $r$?
- Estimates and recommendations ignore divisibility issues and integer issues (floor, ceiling).
- Top curve: repaired algorithm, bottom: estimate in [AFS03].



# Faster ISD algorithms?

- AFS do not allow any error in the $r$ selected positions; algorithm similar to Prange's; can hope to improve chance per iteration using Lee-Brickell, Leon, or Stern.
- Stern is actually exponentially faster than Prange.
- But: in AFS Gaussian elimination is applied to $r \times r$ matrix – while all other algorithms need to apply transformations also to $r \times (w2^b - r)$ part.
- But: asymptotic estimates made for finding words of very small weight where at most 1 such word exists, here lots of words have required weight.
- Stop guessing and try! Search for 2-regular codewords with exactly $2\text{Ш}$ errors in non-selected positions.
- Simplify notation: here $B = 2^b, n = wB, k = n - r$.

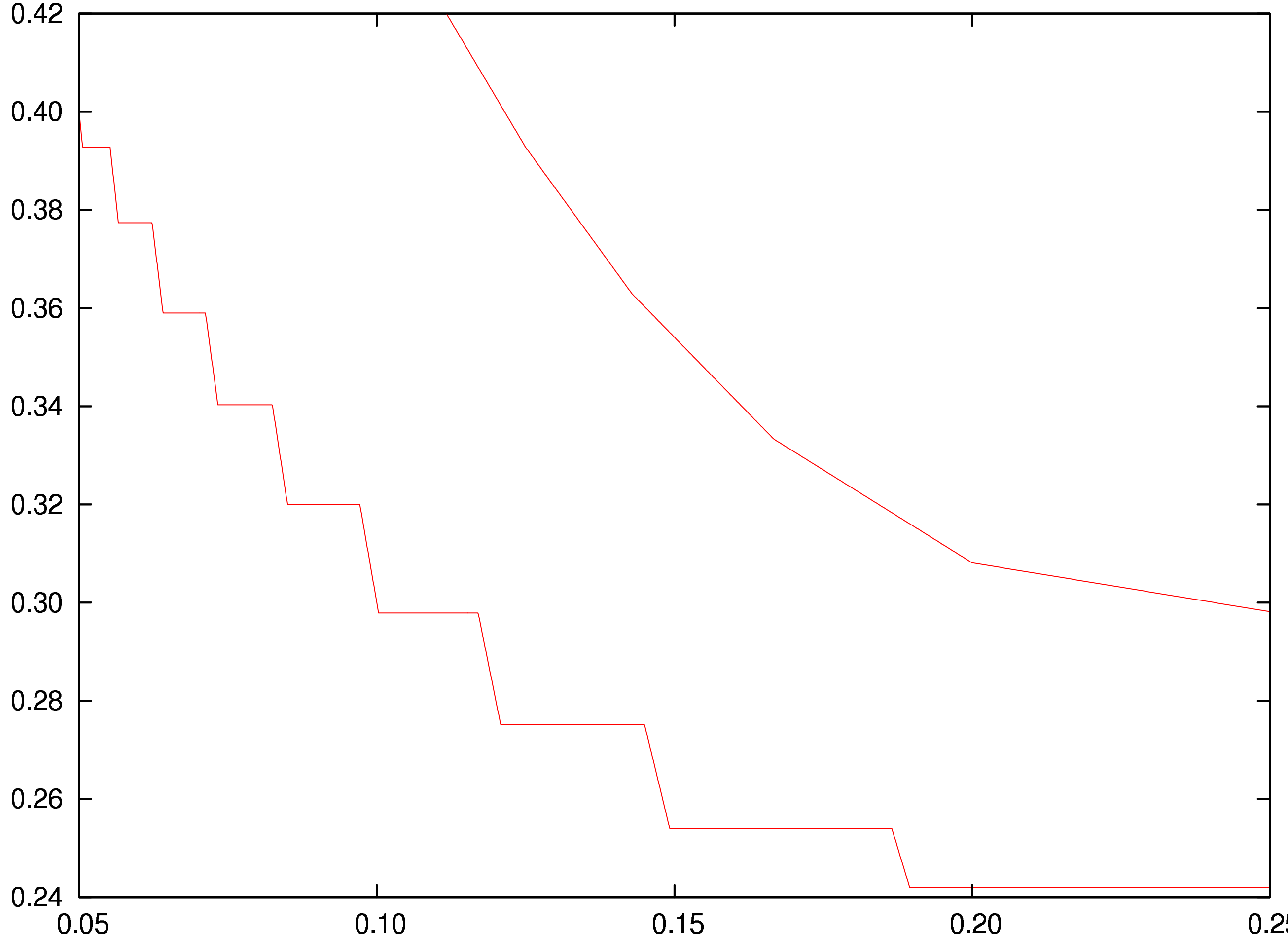
# Our ISD attack

1. Let $w_0 \in \{1, 2, 3, \ldots, w\}$ with $r \leq Bw_0$, $\ell < r$, integer $N$.
2. In $w_0$ blocks select total of $r$ columns from $H$.
3. If the $r$ vectors are linearly dependent go back to 1.
4. Apply transformation on entire matrix $H$.
5. Partition the $w_0$ selected blocks of $H$ into $w_0/2$ "left" blocks and $w_0/2$ "right" blocks.
6. Choose set $S$ of $\ell$ out of the $r$ row indices.
7. Until $L$ has $N$ elements: choose $\text{Ш}$ left blocks, choose one information-set position in each block, and add the $S$-indexed bits. Store resulting $\ell$-bits in $L$ along with the chosen positions. Same for right blocks and list $R$.
8. For each collision of $L$ and $R$, compute entire sum. Output the codeword if it is 2-regular.

# Analysis, improvements, asymptotics

- This description hides choices for $ш, \ell, N$;
- does not state early abort strategies;
- see paper for details.
- Asymptotically good choices:
  - Fix integer $B > 0$, $\omega > 0$.
  - Assume $w/r \to \omega$ as $r \to \infty$.
  - If algorithm parameters $w_0, ш, N, \ell$ are chosen to have $w_0/r \to \omega_0$, $ш/r \to \sigma$, $(\log_2 N)/r \to \tau$, and $\ell/r \to \tau$ for positive real numbers $\omega_0, \sigma, \tau$:
  - Then the algorithm run time also satisfies $(\log_2 \text{time})/r \to \tau$; furthermore, $\tau$ is as small as possible.

# Comparison




Bottom segments: Asymptotic exponent $\tau$ for our algorithm.
Top segments: Asymptotic exponent for AFS algorithm.

# RFSB family [BLPS11b]

- Abbreviation stands for "Really Fast Syndrome-Based".
- Choose $r$ prime and so that $(x^r - 1)/(x - 1)$ is irreducible in $\mathbb{F}_2[x]$. The latter avoids reducibility attacks.
- All elements will be interpreted as binary polynomials modulo $(x^r - 1)$.
- Let the bit string $(s_0, s_1, \ldots, s_{r-1})$ represent the polynomial $a_0 + s_1 x + \cdots + s_{r-1}x^{r-1}$ in $\mathbb{F}_2[x]/(x^r - 1)$.
- Let $c[0], c[1], \ldots, c[2^b - 1]$ be bit strings of length $r$ each.
- Define $c_i[j] = c[j]x^{128(w-i)}$ in $\mathbb{F}_2[x]/(x^r - 1)$, i.e. $c_i[j]$ is a $128(w - i)$-bit rotation of $c[j]$.
- Define the RFSB compression function as $(m_1, m_2, \ldots, m_w) \mapsto c_1[m_1] \oplus c_2[m_2] \oplus \cdots \oplus c_w[m_w] = x^{128(w-1)}c[m_1] \oplus x^{128(w-2)}c[m_2] \oplus \cdots \oplus x^{128}c[m_{w-1}] \oplus c[m_w]$.

# Example: RFSB-509

- Use $r = 509, b = 8, w = 112$.
- Rotations of $509 \times 2^8$ matrix define uncompressed matrix.
- Produce $c[j]$ by encrypting four $16$-byte strings $(0, j, 0, \ldots, 0, 0)$, $(1, j, 0, \ldots, 0, 0)$, $(2, j, 0, \ldots, 0, 0)$, $(3, j, 0, \ldots, 0, 0)$ with AES, concatenating the $128$-bit outputs into a $512$-bit string, and reducing modulo $x^{509} - 1$.
- Use all-0 key for AES (or make keyed hash function).
- Can save space by computing matrix on the fly.
- Parameters offer $2^{128}$ security against all attacks known.
- FSB uses digits of $\pi$ to generate matrix – this is slow and big – but can easily be replaced by using AES.

# FSB vs. RFSB I

- A bigger difference in speed stems from the way the uncompressed matrix entries are generated.
- Example FSB-256 has $r = 1061, b = 14, w = 128$. For $1 \le i \le 128$ generate $r$-bit strings $c_i[0], c_i[1024], c_i[2048], \ldots, c_i[2^{14} - 1024]$ e.g. from $\pi$.
- The $i$-block of the uncompressed FSB-256 matrix consists of the following 16384 column vectors

$$\begin{array}{cccc}
c_i[0], & c_i[0]x, & c_i[0]x^2, \ldots, & c_i[0]x^{1023}, \\
c_i[1024], & c_i[1024]x, & c_i[1024]x^2, \ldots, & c_i[1024]x^{1023}, \\
\vdots & & & \\
c_i[15360], & c_i[15360]x, & c_i[15360]x^2, \ldots, & c_i[15360]x^{1023},
\end{array}$$

considered modulo $x^{1061} - 1$, i.e. of length $r$.

# FSB vs. RFSB II

- The $i$-block of the uncompressed FSB-256 matrix consists of the following 16384 column vectors

$$\begin{array}{cccc} c_i[0], & c_i[0]x, & c_i[0]x^2, \ldots, & c_i[0]x^{1023}, \\ c_i[1024], & c_i[1024]x, & c_i[1024]x^2, \ldots, & c_i[1024]x^{1023}, \\ \vdots & & & \\ \end{array}$$

$$c_i[15360], c_i[15360]x, c_i[15360]x^2, \ldots, c_i[15360]x^{1023},$$

  considered modulo $x^{1061} - 1$, i.e. of length $r$.
- For wide range of $j$ FSB has $c_i[j]$ defined as $c_i[0]x^j$ while RFSB has $c_i[j]$ defined as $c[j]x^i$.
- In FSB the rotation distances are input dependent while they are fixed in RFSB $\Rightarrow$ RFSB better to optimize.
- Want to have $r2^b$ bits in memory anyway.

# Implementations

- FSB-256 and RFSB-509 aim for 128-bit security.
- To help FSB survive round one of the SHA-3 competition, Schwabe did assembly implementation on Core 2 Q9550 (with precomputed digits of pi). Nevertheless FSB-256 takes 95.53 cycles/byte.

# Implementations

- FSB-256 and RFSB-509 aim for 128-bit security.
- To help FSB survive round one of the SHA-3 competition, Schwabe did assembly implementation on Core 2 Q9550 (with precomputed digits of pi). Nevertheless FSB-256 takes 95.53 cycles/byte.
- FSB did not make it into round 2.

# Implementations

- FSB-256 and RFSB-509 aim for 128-bit security.
- To help FSB survive round one of the SHA-3 competition, Schwabe did assembly implementation on Core 2 Q9550 (with precomputed digits of pi). Nevertheless FSB-256 takes 95.53 cycles/byte.
- FSB did not make it into round 2.
- Assembly implementation on same Core 2 Q9550, RFSB-509 takes 10.67 cycles/byte. This is
  - Faster than Grøstl (22.32 cycles/byte);
  - faster than JH (17.20 cycles/byte);
  - faster than SHA-256 (15.26 cycles/byte);
  - faster than Keccak (11.69 cycles/byte).
- Can gain extra speed from incremental hashing, and fast batch verification.

# References

BLNPS09 : Bernstein, L., Niederhagen, Peters, Schwabe: “FSBday: Implementing Wagner’s generalized birthday attack against the SHA-3 round-1 candidate FSB”, Indocrypt 2009.

BLPS11a : Bernstein, L., Peters, Schwabe: “Faster 2-regular information-set decoding”, this conference.

BLPS11b : Bernstein, L., Peters, Schwabe: “Really fast syndrome-based hashing”, Africacrypt 2011.

ePrint versions are generally updated versions of proceedings.