Better price-performance ratios for generalized birthday attacks

D. J. Bernstein
University of Illinois at Chicago

Motivation

A hashing structure proposed by Bellare/Micciancio, 1996:

Standardize functions $f_1, f_2, \ldots$ from, e.g., 48 bytes to 64 bytes.

Compress message $(m_1, m_2, \ldots)$ to $f_1(m_1) \oplus f_2(m_2) \oplus \cdots$.

Bellare/Micciancio advertise "incrementality" of this hash: e.g., updating m_9 to m_9' adds $f_9(m_9') \oplus f_9(m_9)$ to hash. Much faster than recomputation.

Another advantage of this hash:
extreme parallelizability.

Related stream-cipher anecdote:
Salsa20 is one of the world's
fastest unbroken stream ciphers.
Many operations per block
but always 4 parallel operations.

Intel Core 2 Duo software for
8 rounds, 20 rounds of Salsa20
took 3.21, 7.15 cycles per byte
... until Wei Dai suggested
handling 4 *blocks* in parallel.
Now 1.88, 3.91 cycles per byte.

Design hashes for parallelism!

But is this structure secure?

Let's focus on difficulty
of finding collisions in
$f_1(m_1) \oplus f_2(m_2) \oplus \cdots$.

Bellare/Micciancio evaluation:

Easy for long inputs.
Say B blocks/input, B bits/block;
find linear dependency between
$f_1(1) \oplus f_1(0), \ldots, f_B(1) \oplus f_B(0)$;
immediately write down collision.

Not so easy if $\oplus$ is replaced by
$+$, vector $+$, modular $\cdot$, etc.

Much harder for shorter inputs.

van Oorschot/Wiener, 1999, exploiting an idea of Rivest: Parallel collision search against generic B-bit hash function H.

Use 2^c parallel cells; $c \ge 1$. On cell i, generate hashes $H(i), H(H(i)), H(H(H(i))), \ldots$ until a "distinguished" hash h: last $B/2 - c$ bits of h are 0.

Sort the distinguished hashes. Good chance to find H collision.

Total time $2^{B/2-c}$.

. . . assuming some limit on c; no analysis; my guess: $c < B/3$.

Wagner, 2002, "generalized birthday attack": impressively fast collisions for $\oplus$, $+$, vector $+$ for medium-length inputs.

Speed not so impressive for short inputs.
Also, heavy memory use.

Open questions from Wagner:
Smaller memory use?
Parallelization "without enormous communication complexity"?

Bernstein, 2007, this talk:
smaller and much smaller T.

Generalized birthday attack has many other applications.

Some examples from Section 4 of Wagner's paper: LFSR-based stream ciphers (via low-weight parity checks); code-based encryption systems; the GHR signature system; blind-signature systems.

Understanding attack cost is critical for choosing cryptosystem parameters.

Review of Wagner's attack

Example: $f_1(m_1) \oplus \cdots \oplus f_4(m_4)$.

Wagner says:
Choose $2^{B/4}$ values of m_1
and $2^{B/4}$ values of m_2.

Sort all pairs $(f_1(m_1), m_1)$
into lexicographic order.
Sort all pairs $(f_2(m_2), m_2)$
into lexicographic order.

Merge sorted lists to find
$\approx 2^{B/4}$ pairs (m_1, m_2)
such that first $B/4$ bits
of $f_1(m_1) \oplus f_2(m_2)$ are 0.

Compute $\approx 2^{B/4}$ vectors $(f_1(m_1) \oplus f_2(m_2), m_1, m_2)$ where first $B/4$ bits are 0.

Sort into lexicographic order.

Similarly $f_3(m_3) \oplus f_4(m_4)$.

Merge to find $\approx 2^{B/4}$ vectors (m_1, m_2, m_3, m_4) such that first $2B/4$ bits of $f_1(m_1) \oplus f_2(m_2) \oplus f_3(m_3) \oplus f_4(m_4)$ are 0.

Sort to find ≈ 1 collision in all B bits of $f_1(m_1) \oplus f_2(m_2) \oplus f_3(m_3) \oplus f_4(m_4)$.

Wagner says: "$O(n \log n)$ time";
$n = 2^{B/4}$; much better than $2^{B/2}$.
"A lot of memory": gigantic machine storing $2^{B/4}$ vectors.

van Oorschot/Wiener is better!

- Similar time, $\approx 2^{B/4}$, using $\approx 2^{B/4}$ parallel search units.
- Similar machine cost.
- Much more flexibility: easily use smaller machines.
- Normally want collisions in truncation(scrambling(B bits)). Truncation saves time for van Oorschot/Wiener; not Wagner.

Improving Wagner's attack

1. Allow a smaller machine, only 2^c cells.

Generate 2^c values of m_1, m_2, etc.; find collision in $4c$ bits of $f_1(m_1) \oplus f_2(m_2) \oplus \cdots$; hope it works for all B bits. Repeat 2^{B-4c} times.

2. Use parallel mesh sorting; e.g., Schimmler's algorithm.

Time only $2^{c/2}$ to sort 2^c values on 2^c cells in 2-dimensional mesh.

3. Before sorting,
spend comparable time
searching for nice m_i.

Each cell, in parallel,
generates $2^{c/2}$ values of $f_i(m_i)$,
and chooses smallest.
Typically $c/2$ bits are 0.
Reduces number of repetitions
to $2^{B-4c-c/2}$.

4. Optimize parameters,
accounting for constant factors.
Not done in my paper;
new challenge for each
generalized-birthday application.

Summary of time scalability:

- $2^{B-4c+3c/2}$ with serial sorting, non-pipelined memory access; $c \le B/4$.

- $2^{B-4c+2c/2}$ with serial sorting, pipelined memory access; $c \le B/4$.

- $2^{B-4c+c/2}$ with parallel sorting; $c \le B/4$.

- 2^{B-4c} with parallel sorting and initial searching; $c \le 2B/9$.

2^{B-4c} (new) is better than
$2^{B/2-c}$ (van Oorschot/Wiener)
if $c > B/6$. Breakeven point:
$\quad = 2^{B/6}$, $T = 2^{2B/6}$.

Without constraints on c,
minimize price-performance ratio
at $\quad = 2^{2B/9}$, $T = 2^{B/9}$.

Similar improvements for
$f_1(m_1) \oplus \cdots \oplus f_8(m_8)$
etc.

Have vague idea for
combining this attack
with van Oorschot/Wiener.

If idea works as desired:
Time $2^{B/2-7c/4}$; $c \leq 2B/9$.
No more breakeven point;
best attack for all c.

No change in best T.
Without constraints on c,
minimize price-performance ratio
at $= 2^{2B/9}$, $T = 2^{B/9}$.

A cryptanalytic challenge

Rumba20$(m_1, m_2, m_3, m_4) =$
$f_1(m_1) \oplus f_2(m_2) \oplus$
$f_3(m_3) \oplus f_4(m_4)$.
Each f_i is a tweaked Salsa20
mapping 48 bytes to 64 bytes.

Rumba20 cycles/compressed byte
$\approx 2 \cdot$ Salsa20 cycles/byte.
Generally faster than SHA-256.
Salsa20, f_i, Rumba20
have 20 internal rounds;
can reduce rounds to save time.

How cheaply can we find
collisions in Rumba20?

Status: Best $T \approx 2^{171}$
with $\approx 2^{114}$ parallel cells.

Better attack on 4-xor?
Better attack on Rumba20?
On the ChaCha20 variant?
On reduced-round variants?
Quickly generate leading 0's?

I offer $1000 prize for
the public Rumba20 cryptanalysis
that I consider most interesting.
Awarded at the end of 2007.

Send URLs of your papers to
`snuffle6@box.cr.yp.to`.