Recent advances on iterated hash functions

Antoine Joux
DGA/SPOTI and
University de Versailles St-Quentin-en-Yvelines
France

State of the art
End of 2003

Security of Hash Functions

- Several properties are usually required:
  - One-wayness
  - Preimage resistance
  - Second Preimage resistance
  - Collision freeness
- Another property is also sometime used:
  - k–collision freeness

One-wayness

H is a hash function on n bits.

- Given a set S
- Given $M \in S$
- Let $h = H(M)$.

It should be hard from h to recover M.

- The generic attack is exhaustive search
- It runs in time proportional to the cardinality of S

Preimage resistance

H is a hash function on n bits.

- Given a set S
- Given $M \in S$
- Let $h = H(M)$.

It should be hard from h to find $M' \in S$ such that $H(M') = h$, possibly with $M = M'$.

- The generic attack is exhaustive search
- It runs in time cardinality of S or 2^n (whichever is the smallest)

Second Preimage resistance

H is a hash function

- Given a set S
- Given $M \in S$

It should be hard to find $M' \neq M$ with $H(M') = H(M)$.

- The generic attack is exhaustive search
- It runs in time 2^n

Collision freeness

H is a hash function

- Given a set S

It should be hard to find distinct M and M' with $H(M) = H(M')$.

- The generic attack is birthday paradox
- It runs in time $2^{n/2}$

k-Collision freeness

- Given a set S

It should be hard to find M_1, . . . , M_k with

$$H(M_1) = \cdots = H(M_k).$$

- The generic attack is generalized birthday paradox
- It runs in time $2^{n \cdot (k-1)/k}$

Iterated Hash functions

- Many practical hash functions are iterated hash functions
  - Examples: SHA, MD4, MD5, Tiger, ...
- They are based on a compression function
- The compression function f takes as input a state and a message block
- It outputs a new state

Iterated Hash functions

After an initial transform (padding) the message is split into blocks $B_1, \ldots, B_t$.

- The computation starts from h_0 an initial state
- Then it loops from $i = 1$ to $i = t$, computing

$$h_{i+1} = f(h_i, B_i).$$

- h_t is the final hash value

Example of a security problem

- Assume that :

$$f(h, B) = \pi_B(h)$$

- Then, there is a $2^{n/2}$ "only" preimage attack.
- On two blocks, we want to find $f(f(h_0, B), B') = h_F)$:
  - Compute $2^{n/2}$ values $f(h_0, B_i)$
  - Compute $2^{n/2}$ values $f^{-1}(h_F, B'_j)$
  - A collision between the two sets yields the expected preimage.

- Usually fixed by

$$f(h, B) = \pi_B(h) + h$$

Padding

- The most commonly encountered padding is as follows:
- Complete the message by a bitstring :

$$1000 \cdots 000$$

- Then add the binary representation of the initial message length
- The number of '0' is the minimal number that yields an integral number of blocks.

- This *padding* is redundant, why ?
- Can we choose a simpler padding ?

Second preimage on long messages

- Let M be a long message (say $N \approx 2^{n/2}$ blocks)
- Let $h_1, \ldots, h_N$ denote the intermediate hash values.
- Choose M' another random long message, with hash values h'_j
- Any collision $h_i = h'_j$ yields a second preimage of $H(M)$
- The redundant *padding* avoids this attack.

A case example: `SHA`

$\texttt{SHA}$ compression function

Initialize $\left\langle A^{(0)}, B^{(0)}, C^{(0)}, D^{(0)}, E^{(0)} \right\rangle$

For $i = 0$ to 79

$$A^{(i+1)} =$$
$$ADD\left(W^{(i)}, ROL_5\left(A^{(i)}\right), f^{(i)}\left(B^{(i)}, C^{(i)}, D^{(i)}\right), E^{(i)}, K^{(i)}\right)$$

$$B^{(i+1)} = A^{(i)}$$

$$C^{(i+1)} = ROL_{30}\left(B^{(i)}\right)$$

$$D^{(i+1)} = C^{(i)}$$

$$E^{(i+1)} = D^{(i)}$$

Output

$$\left\langle A^{(0)} + A^{(80)}, B^{(0)} + B^{(80)}, C^{(0)} + C^{(80)}, D^{(0)} + D^{(80)}, E^{(0)} + E^{(80)} \right\rangle$$

Functions $f^{(i)}(X, Y, Z)$, and Constants $K^{(i)}$

Tour i	Function $f^{(i)}$		Constant $K^{(i)}$
	Nom	Dfinition	
0 –19	IF	$(X \wedge Y) \vee (\neg X \wedge Z)$	`0x5A827999`
20–39	XOR	$(X \oplus Y \oplus Z)$	`0x6ED9EBA1`
40–59	MAJ	$(X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$	`0x8F1BBCDC`
60–79	XOR	$(X \oplus Y \oplus Z)$	`0xCA62C1D6`

Expansion in `SHA-0`

- Input: $\left\langle W^{(0)}, \ldots, W^{(15)} \right\rangle$

$$W^{(i)} = W^{(i-3)} \oplus W^{(i-8)} \oplus W^{(i-14)} \oplus W^{(i-16)} \quad . \tag{1}$$

- Output: $\left\langle W^{(0)}, \ldots, W^{(79)} \right\rangle$

Expansion in SHA-1

- Slight difference

$$W^{(i)} = ROL_1 \left(W^{(i-3)} \oplus W^{(i-8)} \oplus W^{(i-14)} \oplus W^{(i-16)} \right) \quad . \tag{2}$$

- $E_0 = (e_0)^{32}$ parallel expansion in SHA-0.
- E_1 more complex expansion in SHA-1.

Rationale for the change

- No official explanation in 1995.
- At Crypto 1998 [Chabaud, J.]
- Differential attack
- 2^{61} complexity collision on `SHA-0`

Recent results
2004–2005

Cascaded Hash functions

Cascade = Composition of several crypto functions in a single one

- The goal is usually to increase security
- With hash functions, the following construction is natural:
  - Given H and G form:

$$(H(M) \| G(M)).$$

- Turns two 128-bits hash functions into a 256-bits one.

Security of a Cascade

- Clearly, with random oracles, the construction is secure.
- What happens with real hash functions ?
- Folklore knowledge:
  - With two similar hash functions, this feels risky.
  - If the hash functions are "independent", it should be ok.

Here, we answer for iterated hash functions

Iterated Hash functions and k-collisions

Iterated hash functions are not k-collision free !

- Finding k-collisions with $k > 2$ is easier than expected
- Even when k is very large, it is still easy
- We show how to find 2^t-collisions in time $t \cdot 2^{n/2}$

Iterated Hash functions and k-collisions

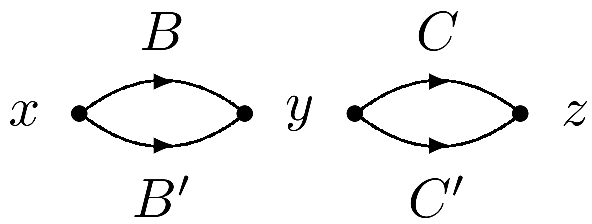
Assume (w.l.g.) that blocks are larger than internal states

- Finding a one block collision from any internal state x take times $2^{n/2}$.
- We represent it graphically by:

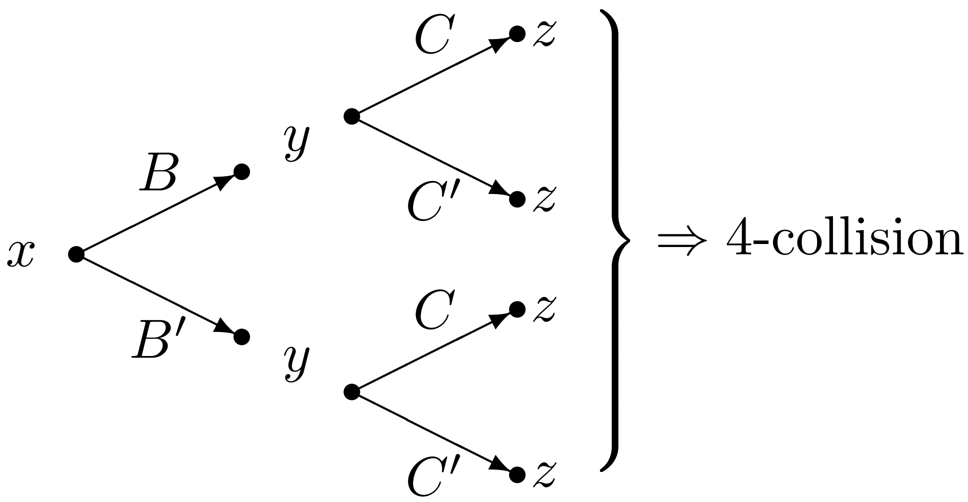
$$\begin{tikzcd} x \arrow[r, bend left, "B"] \arrow[r, bend right, "B'"'] & y \end{tikzcd}$$

4-collisions

- With two calls to the basic attack we find:



- This yields the following 4-collision



2^t-collisions

- Similarly with t calls to the basic attack we find:

$$\begin{tikzcd} h_0 \arrow[r, bend left, "B_1"] \arrow[r, bend right, "B_1'"'] & h_1 \arrow[r, bend left, "B_2"] \arrow[r, bend right, "B_2'"'] & h_2 \arrow[r, bend left, "B_3"] \arrow[r, bend right, "B_3'"'] & h_3 \cdots & h_{t-1} \arrow[r, bend left, "B_t"] \arrow[r, bend right, "B_t'"'] & h_t \end{tikzcd}$$

It yields a 2^t-collision

Application to Cascades

- Given G and H, two n-bits iterated hash functions
- Find a $2^{n/2}$-collision on G
- Among this large set, with good probability, we find M and M' with
$$H(M) = H(M')$$
- Since, M and M' also collide on G we have
$$(H(M) \| G(M)) = (H(M') \| G(M'))$$
- The runtime is $O(n \cdot 2^{n/2})$.

Application to Cascades

- **It even works when H is a random oracle**
- Thus cascading is not secure even when G and H are independent
- A similar attack applies to (second) preimage resistance

Other application

- Kelsey and Schneier (eprint 11/2004, Eurocrypt 2005)
- Renders the second preimage attack on long messages feasible

Specific attacks

- Many recent attacks on specific iterated hash algorithms
- Based on greatly improved differential attacks
  - SHA family, neutral bits, Biham and Chen, Crypto'04
  - MD4 and MD5, Wang et al., Eurocrypt'05
  - SHA-0, J. et al, Rump session Crypto'04
  - SHA-1, Wang et al., recent announce

Conclusion

Questions