Generic collision attacks on hash-functions and HMAC

Chris Mitchell

Royal Holloway, University of London

Agenda

1. Hash-functions and collision attacks
2. Memoryless strategy for finding collisions
3. Properties of random functions
4. Impact on memoryless strategy
5. Attacking HMAC
6. Other observations
7. Acknowledgements

Hash-functions

- Cryptographic hash-functions are a key element of all widely used signature schemes.
- They are also used as a component of many asymmetric encryption schemes.
- Such functions map an arbitrary length bit string to a fixed length bit string (e.g. of length 128, 160, 224 or 256 bits).
- If the output length is *n* bits, then we refer to an *n*-bit hash-function.

Examples

- Examples of widely used hash-functions include:
  – RIPEMD-128 and RIPEMD-160 (standardised in ISO/IEC 10118-3);
  – MD5 (now discredited);
  – SHA-1 (a NIST and ISO/IEC standard, but now looking a little shaky);
  – SHA-256 (NIST and ISO/IEC standardised).

Necessary properties

- A cryptographic hash-function h is normally expected to have the following properties:
  - *pre-image resistance (the 'one way' property)*: given an arbitrary n-bit block y it is computationally infeasible to find an input x such that $h(x) = y$;
  - *second pre-image resistance*: given an arbitrary bit string x, it is computationally infeasible to find a distinct bit-string $x' \neq x$ such that $h(x') = h(x)$.
- In many cases (e.g. when h is used as part of a signature scheme) the following property is also required:
  - *collision resistance*: it is computationally infeasible to find two distinct bit-strings x and x' ($x \neq x'$) such that $h(x) = h(x')$.

A trivial collision attack

- The Birthday Paradox tells us that, if we choose around $\sqrt{m}$ elements from a set of size m (with replacement), then there is a good chance of choosing one element twice.
- Hence if we compute $h(x)$ for around $\sqrt{(2^n)} = 2^{n/2}$ messages, then there is a good chance of finding a collision.
- Hence, given $2^{n/2}$ computations and $2^{n/2}$ storage, we can find a collision for any n-bit hash-function.

Agenda

1. Hash-functions and collision attacks
2. Memoryless strategy for finding collisions
3. Properties of random functions
4. Impact on memoryless strategy
5. Attacking HMAC
6. Other observations
7. Acknowledgements

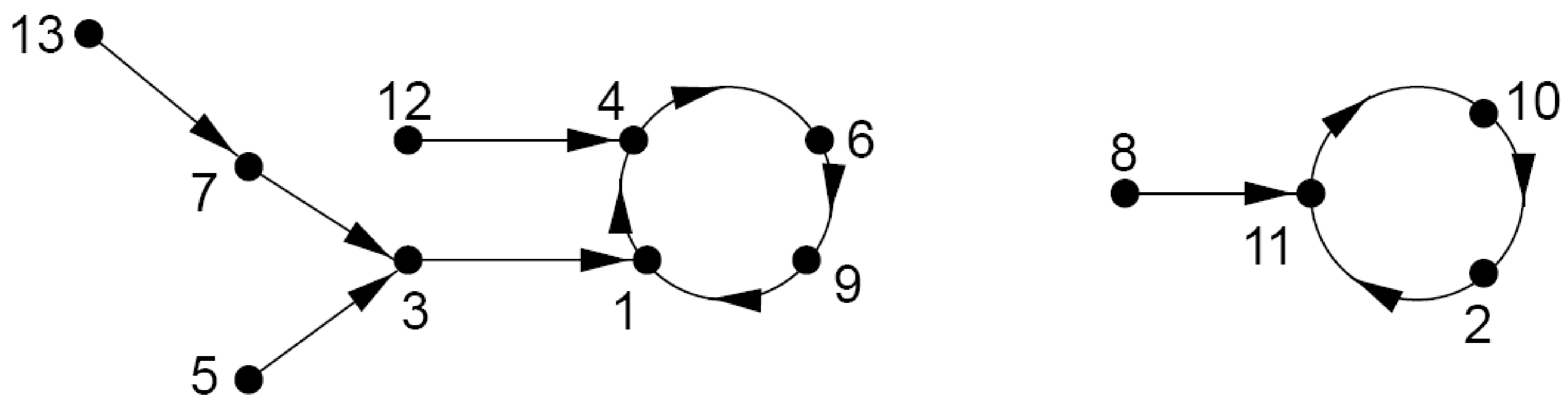
Background

- A memoryless technique for finding hash-function collisions was proposed by Quisquater and Delescaille (1989) – we refer to this as the QD attack.
- This was refined by van Oorschot and Wiener (1994) to produce a technique which is both parallelisable and produces collisions for 'meaningful' messages [see also section 9.7.1 of the HAC].
- This requires $O(2^{n/2})$ computational steps and trivial memory for an n-bit hash-function.

Functions and functional graphs

- Suppose f: $\{1,2,\ldots,m\} \to \{1,2,\ldots,m\}$.
- Then f defines a directed graph (the *functional graph*) with:
  - m vertices (the values 1, 2, …, m); and
  - m directed edges, where $a \to b$ is an edge if and only if $f(a)=b$.
- This graph is made up of a number of connected sub-graphs (components), where each component will contain a single directed cycle and some directed trees attached to (pointing to) the cycle.

Example



The above example of a functional graph for *m*=13 is taken from the HAC.

Random functions

- Suppose $f : \{1,2,\ldots,m\} \rightarrow \{1,2,\ldots,m\}$ is randomly chosen from the class of all such functions. [We call f a random function].
- The expected length of a cycle in the functional graph is $\sqrt{(\pi m/8)}$, and the expected maximum cycle length is around $1.7\sqrt{m}$.
- The expected length of a 'tail' (i.e. a path from a point to a cycle) is also $\sqrt{(\pi m/8)}$, and the expected maximum tail length is around $0.8\sqrt{m}$.
- The largest connected component will contain $O(2m/3)$ vertices, i.e. the majority of the vertices will be in one large connected component.

The QD attack I

- The QD attack involves observing that, for an *n*-bit hash-function *h*, we can consider the action of *h* on the set of all *n*-bit blocks:
- We can model this restricted version of *h* as a random function,

  $h: \{1,2,\ldots,m\} \rightarrow \{1,2,\ldots,m\}$, where $m = 2^n$.
- Hence, if we choose a random block *x*, then the sequence:

  $x, h(x), h^2(x), h^3(x), \ldots$

  will define a path in the functional graph of (the restricted version of) *h*.

QD attack II

- This path will eventually reach a cycle, and will then repeatedly go round the cycle.
- We just need to find a way to notice when the path round the cycle has been completed.
- This can be done with *distinguished points*.
- That is, call an *n*-bit block *x* a *distinguished block* if the leftmost *n*/2-*s* bits of *x* are all zeros (for some smallish *s*, e.g. *s*=8).

  (Actually, any fixed pattern of *n*/2-*s* bits will do equally well – however, looking for zeros is usually easiest).

QD attack III

- The attack proceeds as follows:

```
choose a random block $x$;
**for** $i = 1, 2, \ldots$
    **compute** $h^i(x)$;
    **if** $h^i(x)$ is a distinguished block **then**
        **if** ($h^i(x) = h^j(x)$ for any recorded *distinguished pair*
            ($j, h^j(x)$), $j < i$) **then stop**;
        **record** the *distinguished pair* ($i, h^i(x)$)
```

QD attack IV

- The algorithm will give a pair (i, j) such that $h^i(x) = h^j(x)$ and $1 \leq j < i$.
- This means that, either:
  - $x = h^{i-j}(x)$ [not a collision but an interesting fact in itself, since we have found a pre-image for x], or
  - for some $t < j$: $h^{i-t}(x) = h^{j-t}(x)$ & $h^{i-t-1}(x) \neq h^{j-t-1}(x)$, i.e. we have a collision.
- If the first event occurs, start again. The first event will occur if and only if x is on a cycle (probability very small).

QD attack complexity

- The storage requirement is clearly trivial, i.e. $O(2^s)$.
- Each cycle will probably contain at least one distinguished block (choose s large enough to ensure this, bearing in mind the average cycle length).
- Since the expected path length and expected cycle size are both $\sqrt{(\pi m/8)}$, where $m=2^n$, the number of computational steps is clearly $O(2^{n/2})$.

Agenda

1. Hash-functions and collision attacks
2. Memoryless strategy for finding collisions
3. Properties of random functions
4. Impact on memoryless strategy
5. Attacking HMAC
6. Other observations
7. Acknowledgements

Iterated random functions

- We now need to consider some further properties of random functions.
- In particular, we are interested in the properties of the image set of functions

$$f: \{1,2,\ldots,m\} \to \{1,2,\ldots,m\}.$$

Image sets I

Theorem Suppose f_1, f_2, … are independent random functions, where

f_i: $\{1,2,\ldots,m\} \rightarrow \{1,2,\ldots,m\}$.

Then the expected size of the image set of the compound function $f_t \bullet f_{t-1} \bullet \ldots \bullet f_1$ (i.e. the function made up of f_1, followed by f_2, …, followed by f_t) is bounded above by $2m/t$ (for large image sets).

Corollary If one iteratively applies 2^w random functions to a set, then the size of the image set (for a large image set) is around $1/2^{w-1}$ times the size of the domain.

Image sets II

The expected sizes of the image sets for small *t* (and large *m*) are as follows:

t **(number of functions)**	**size of image set**
1	0.63*m*
2	0.47*m*
3	0.37*m*
4	0.31*m*
5	0.27*m*

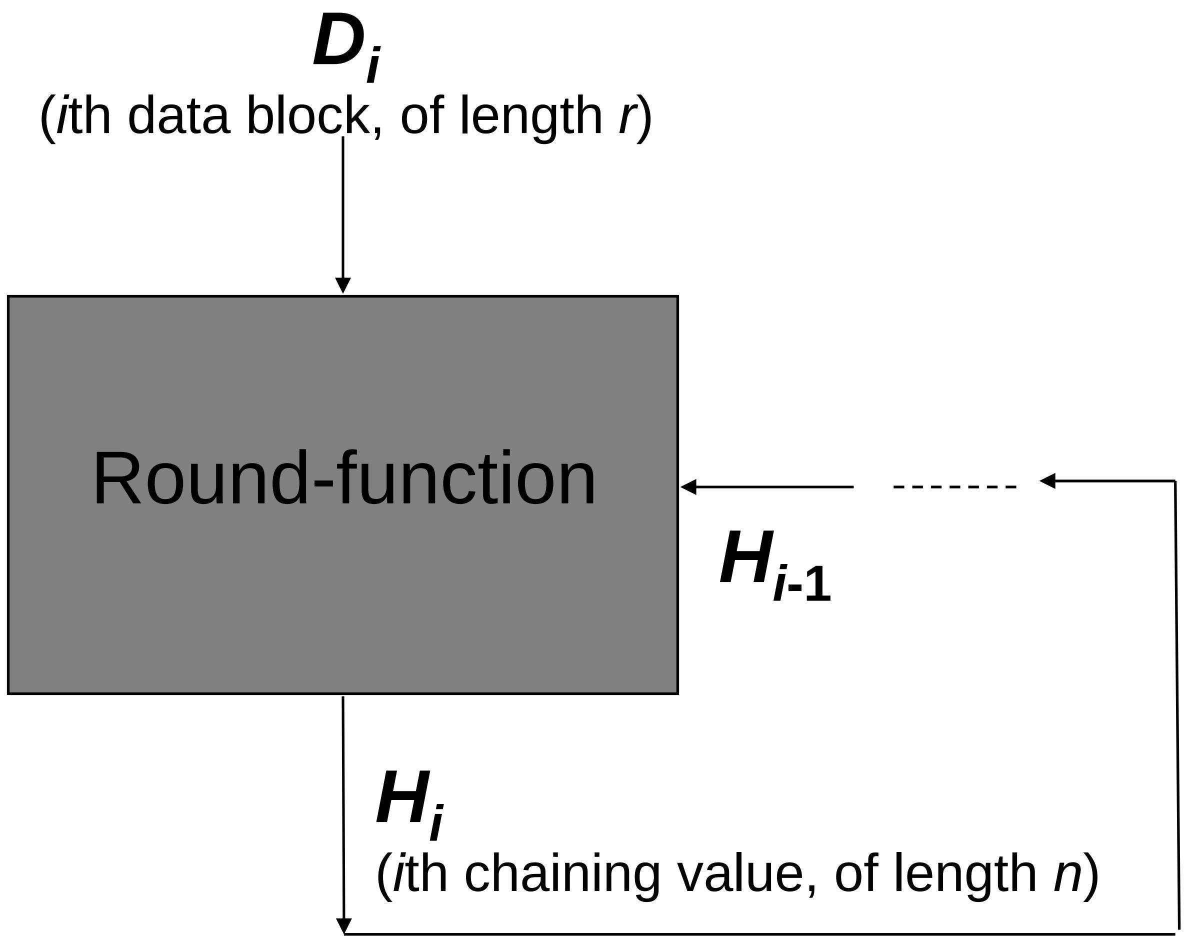
Agenda

1. Hash-functions and collision attacks
2. Memoryless strategy for finding collisions
3. Properties of random functions
4. <u>Impact on memoryless strategy</u>
5. Attacking HMAC
6. Other observations
7. Acknowledgements

The Merkle meta-method

- All the widely used hash-functions are based on Merkle's meta-method.
- That is, they involve:
  1. padding the data
  2. splitting the padded data into blocks of a fixed length (say *r*);
  3. setting an initial chaining value to a fixed value;
  4. iteratively applying a round function (or compression function), which takes an *n*-bit chaining value and an *r*-bit data block as input, and outputs an *n*-bit chaining value.
- The hash-code is the final chaining value (of length *n* bits)

Merkle meta-method



D_i
(*i*th data block, of length *r*)
Round-function
H_{i-1}
H_i
(*i*th chaining value, of length *n*)

Example parameters

- MD5, SHA-1 and RIPEMD-160 have r=512 and n=160.
- SHA-256 has r=512 and n=256.

The length block

- All the well-known hash-functions have a similar padding scheme.
- This involves adding up to r-1 zeros to the end of the message to make sure the length is a multiple of r.
- Then an additional r-bit block is added to the end, which encodes the original message length.

Effect of the length block

- When computing $h^i(x)$ (for i=1,2,…) in the QD attack, the input to h has a fixed length (n bits).
- Hence, in each case, the padded message being hashed will contain two r-bit blocks (assuming $n \leq r$), the first of which will contain the n-bit input followed by r-n zeros, and the second of which will be fixed (the length block).
- Since the initial chaining value is fixed, the first iteration of the round function can be treated as a random function from a set of size $m=2^n$ onto itself.
- Since the r-bit length block is fixed, the second iteration of the round function can also be thought of as a random function.
- That is, the values of $h^i(x)$ for i=1,2,…, will be drawn from a set of size 0.47×2^r (and not 0.63×2^r).

Aside: a simplying assumption

- For the previous slide (and in all subsequent discussions) we look only at the size of the image set, and (implicitly) assume that the probabilities of choosing elements in this set are equal.
- However, this is not correct. Some elements are more likely to be chosen than others.
- This will increase the probability of a collision, i.e. our simplifying assumption underestimates the attack success probability.

QD attack effects I

- This will mean that the expected size of the cycle will be smaller then the previous estimates would suggest, which means that the attack is actually more effective than would otherwise be the case.
- Indeed, suppose the attacker actually computed $g^i(x)$ instead of $h^i(x)$, where:

$$g(y) = h(y||0^{r-n}||b_1||b_2||\ldots||b_v)$$

for some v, where 0^{r-n} denotes r-n zeros, b_i is a (fixed) distinct r-bit block for every i, and || is concatenation.

- Thus computing g involves computing the round function a total of v+2 times, which can be modelled as the iterative performance of v+2 independent random functions from a set of size m=2^n onto itself.

QD attack effects II

- That is, from the theorem, the output of $g^i(x)$ will be drawn from a set of size at most $2^{n+1}/(v+2)$.
- Suppose v = 2^u-2.
- In this case (by heuristic arguments) the expected size of the cycle which the computation of $g^i(x)$ will proceed round will be $O(2^{(n+1-u)/2})$.
- That is, the complexity of the attack has now been reduced by a factor of $2^{(u-1)/2}$, i.e. a collision can be found using $O(2^{(n+1-u)/2})$ hash computations.

QD attack effects III

- However, we have cheated!
- The reduction in the number of hash computations is at the cost of hashing messages whose padded versions will contain 2^u r-bit blocks, so the overall attack complexity will actually increase as u increases.
- This means it is necessary to be careful about assessing attack complexity solely in terms of hash function computations.

Agenda

1. Hash-functions and collision attacks
2. Memoryless strategy for finding collisions
3. Properties of random functions
4. Impact on memoryless strategy
5. Attacking HMAC
6. Other observations
7. Acknowledgements

Extending the attack

- We can directly apply the observations on the effectiveness of the QD attack on a hash-function to a collision attack on HMAC.
- A collision attack on a MAC scheme requires finding two messages with the same MAC.
- A MAC can then be found for a message, without it being computed by the legitimate parties.
- Using 'standard' methods, finding a collision requires $n^{0.5}$ message/MAC pairs, for an n-bit MAC.

HMAC

- MAC based on hash function *h*
  - K_1 and K_2 are derived from key K
  - HMAC of message m is $h(K_2||h(K_1||m))$

$$\begin{tikzcd}
m \arrow[r] & K_1||m \arrow[r] & \boxed{h} \arrow[r] & K_2||h(K_1||m) \arrow[r] & \boxed{h} \arrow[r] & \mathrm{HMAC}_K(m) \\
 & K_1 \arrow[u] & K \arrow[l] \arrow[r] & K_2 \arrow[u] & &
\end{tikzcd}$$

HMAC Forgeries I

- Collect HMACs for messages $m_1,\ldots,m_s$
  - Same length and agree in final kr bits
  - s chosen appropriately (as for QD attack).
- Find messages m_i and m_j with same HMAC
- Obtain HMAC of $m_i||p$ for any string p
  - Forge $\mathrm{HMAC}_K(m_j||p)$ as $\mathrm{HMAC}_K(m_i||p)$

HMAC Forgeries II

- Hash function with n=128 and r=512
  - Messages agreeing in last 2^{38} bytes (v=32)
  - 2^{49} hash values should give collision
- Collect 2^{49} HMAC values
  - Find messages m_i and m_j with same HMAC
  - Obtain HMAC of $m_i||p$ for any string p
  - Forge $\text{HMAC}_K(m_j||p)$ as $\text{HMAC}_K(m_i||p)$

Security of HMAC

- The attack on HMAC works because the hash-function ‘compression function’ is not 1-1 for a fixed message block input.
- However, when computing a MAC using one of the CBC-MAC methods, the compression function is 1-1.
- Hence this attack does not apply to CBC-MACs.

Agenda

1. Hash-functions and collision attacks
2. Memoryless strategy for finding collisions
3. Properties of random functions
4. Impact on memoryless strategy
5. Attacking HMAC
6. Other observations
7. Acknowledgements

Applications of iterated hash-functions

- Multiple iterations of hash-functions have been proposed for use in a variety of applications.
- For example:
  - Lamport proposed the use of hash-chains for user authentication;
  - hash-functions are used for random bit generation, by iteratively hashing a random ‘seed’;
  - hash-functions are often used as a simple and quick method of detecting accidental errors in a data string D, by storing $h(D)$ with D (for a hash-function h).

Multiple iterations

- If an *m*-bit hash-function is iterated a large number of times, then we know that the size of the domain will be $O(\sqrt{m})$.
- This means that there is a potential danger of loss of up to half of the entropy.
- The effect will, to a large extent, depend on the hash-function length, and the way that the iterations are computed.

Example scenario I

- Suppose a series of 64-bit values are required (e.g. as pseudo-random values).
- Suppose a 128-bit hash-function is iteratively applied to a 128-bit seed.
- Could take the first 64 bits of the output, before rehashing.
- However, the full 128 bits must be rehashed, and not just 64.
- If not, then after $O(2^{32})$ hash applications, the sequence of bits output will cycle with period $O(2^{32})$.
- That is only $\approx$32 bits of entropy will remain.

Example scenario II

- However, if the full 128 bits are hashed every time, then, after 2^{32} iterations, around 96 bits of entropy will remain.
- This is likely to be satisfactory for most purposes.
- Of course, after 2^{64} iterations, ‘only’ $\approx$64 bits of entropy will remain.

Theory

- Most importantly, and this work has not been done previously, we would like to avoid the simplifying assumption, and assess the true probability of a successful attack.
- Some additional work is required to properly compute the expected sizes of the cycles and tails for the functional graph of compound random functions.
- It would also help to know the expected size of the largest component of the functional graph for a ‘compound’ random function.
- This should then yield theoretical results on the performance of variants of the QD attack.

Agenda

1. Hash-functions and collision attacks
2. Memoryless strategy for finding collisions
3. Properties of random functions
4. Impact on memoryless strategy
5. Attacking HMAC
6. Other observations
7. Acknowledgements

Acknowledgements

- This is joint work with Sean Murphy.
- Thanks also to Laurence O’Toole and Peter Wild for useful discussions.
- Most of the observations in this talk were originally made by Preneel and van Oorschot (1995/1999).