Making a hash of things

Looking at cryptographic hash functions

March 2006 kw.pm

`http://kw.pm.org/`

Overview

Hash functions

- Some definitions
- Cryptographic requirements
- Some applications
- Generic attacks
  - Optimizing generic attacks
  - Finding meaningful collisions
- Iterated hash functions
  - MD5
  - SHA-1
- Signatures
  - Forging signatures

A note on the numbers

A note on the numbers

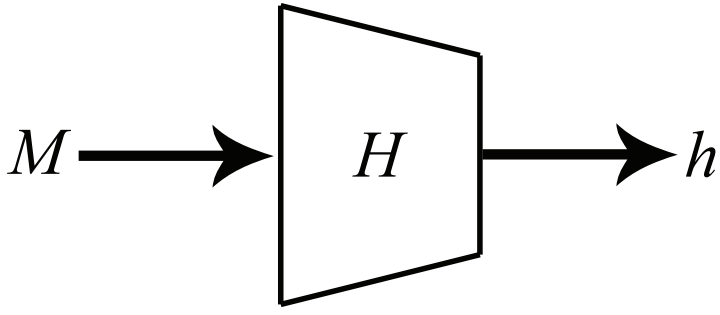
$$2^{128} = 340282366920938463463374607431768211456$$

A note on the numbers

$$2^{128} = 340282366920938463463374607431768211456$$

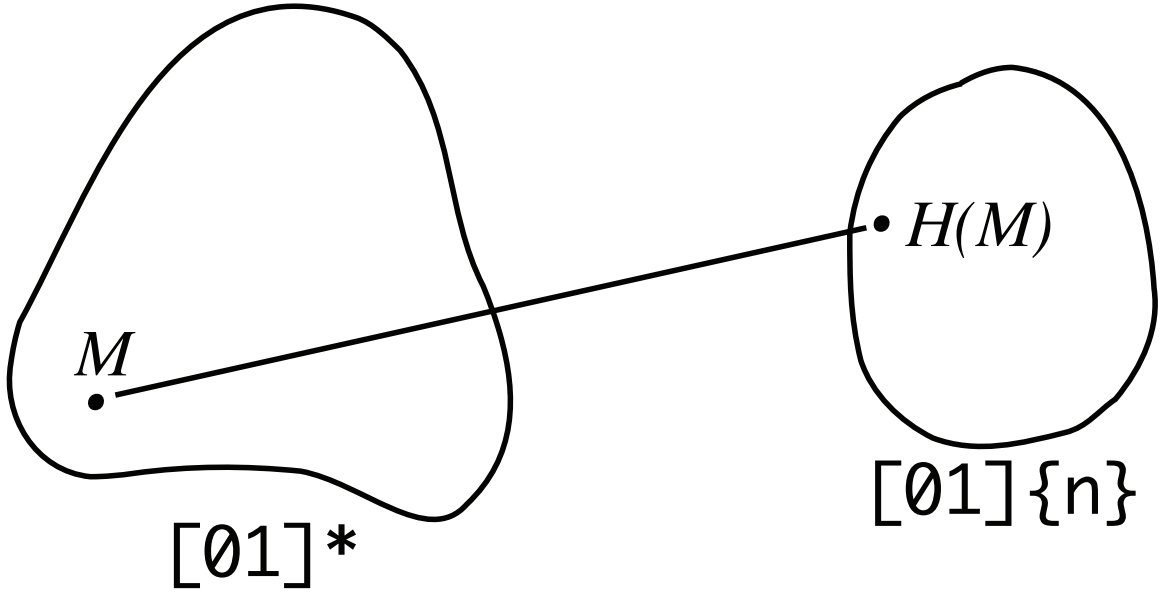
That is not a small number!

Hash functions



Simply put, a hash function takes some input of variable length, and returns a value of fixed length.

Hash space



Cryptographic requirements

What is a good hash function?

Cryptographic requirements

What is a good hash function?

- How random does it appear?

Cryptographic requirements

What is a good hash function?

- How random does it appear?
- Preimage resistance
  If you are given a hash value x, how hard is it to find a message M whose hash value $H(M) = x$.

Cryptographic requirements

What is a good hash function?

- How random does it appear?
- Preimage resistance
  If you are given a hash value x, how hard is it to find a message M whose hash value $H(M) = x$.
- 2nd preimage resistance
  Given data M_1, how hard is it to find different data $M_2, M_1 \neq M_2$ such that they have the same hash value, $H(M_1) = H(M_2)$.

Cryptographic requirements

What is a good hash function?

- How random does it appear?
- Preimage resistance
  If you are given a hash value x, how hard is it to find a message M whose hash value $H(M) = x$.
- 2nd preimage resistance
  Given data M_1, how hard is it to find different data $M_2, M_1 \neq M_2$ such that they have the same hash value, $H(M_1) = H(M_2)$.
- Collision resistance
  How hard is it to find to two arbitrary messages, $M_1, M_2, M_1 \neq M_2$ that have the samd hash value, $H(M_1) = H(M_2)$.

Some applications

Some applications

- Passwords. Requires preimage resistance.

Some applications

- Passwords. Requires preimage resistance.
- Modification Detection Codes. Requires 2nd preimage resistance.

Some applications

- Passwords. Requires preimage resistance.
- Modification Detection Codes. Requires 2nd preimage resistance.
- Message Authentication Codes.

Some applications

- Passwords. Requires preimage resistance.
- Modification Detection Codes. Requires 2nd preimage resistance.
- Message Authentication Codes.
- Pseudorandom number generation.

Some applications

- Passwords. Requires preimage resistance.
- Modification Detection Codes. Requires 2nd preimage resistance.
- Message Authentication Codes.
- Pseudorandom number generation.
- Cryptographic key derivation.

Some applications

- Passwords. Requires preimage resistance.
- Modification Detection Codes. Requires 2nd preimage resistance.
- Message Authentication Codes.
- Pseudorandom number generation.
- Cryptographic key derivation.
- Digital signatures. Requires all three resistances.

Attacking hash functions

Attacking hash functions

- Generic attacks do not exploit any specific properties of the construction of a particular hash function

Attacking hash functions

- Generic attacks do not exploit any specific properties of the construction of a particular hash function
- The more “random” a function, the more secure it is.

Attacking hash functions

- Generic attacks do not exploit any specific properties of the construction of a particular hash function
- The more “random” a function, the more secure it is.
- Finding preimages and 2nd preimages

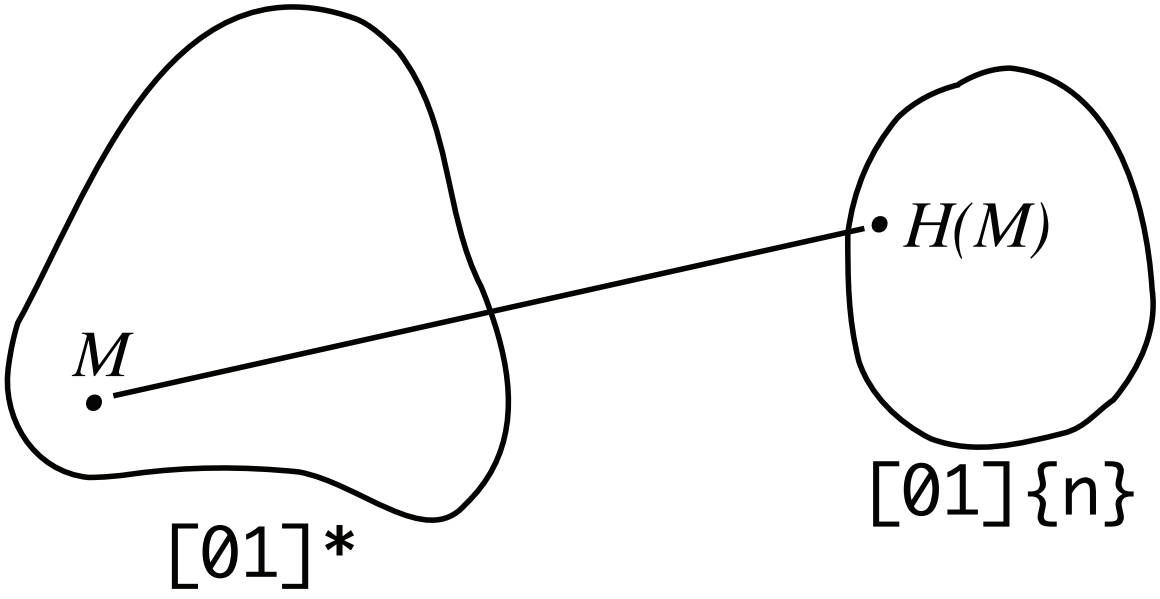
Attacking hash functions

- Generic attacks do not exploit any specific properties of the construction of a particular hash function
- The more “random” a function, the more secure it is.
- Finding preimages and 2nd preimages
  - There is no efficient way to find these with a generic attack.

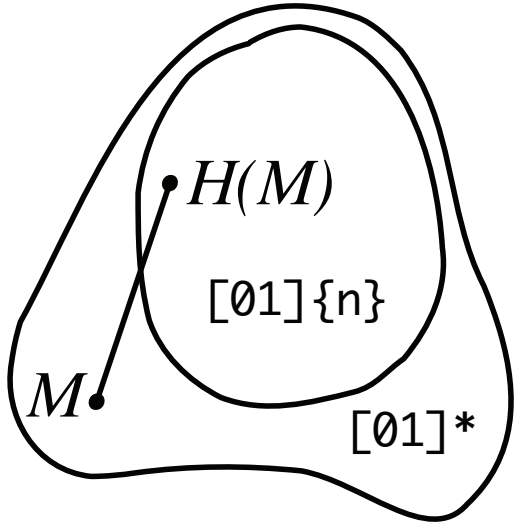
Attacking hash functions

- Generic attacks do not exploit any specific properties of the construction of a particular hash function
- The more “random” a function, the more secure it is.
- Finding preimages and 2nd preimages
  - There is no efficient way to find these with a generic attack.
  - Simply continue to pick arbitrary x until $H(x) = y$ or $H(x) = H(x_0)$

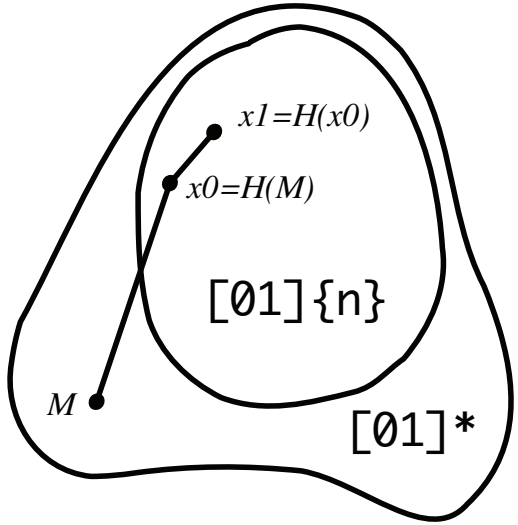
Finding collisions



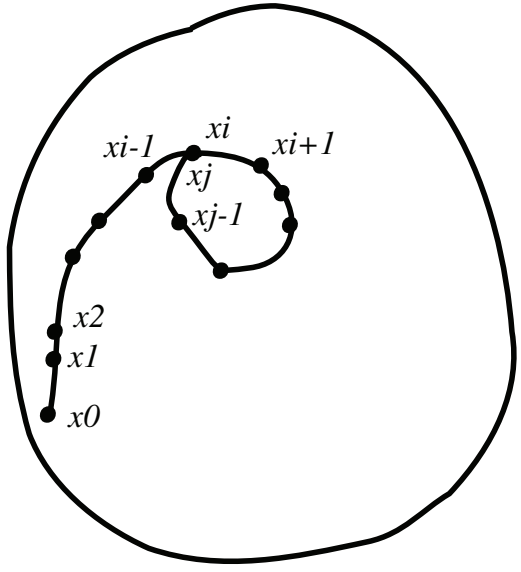
Finding collisions



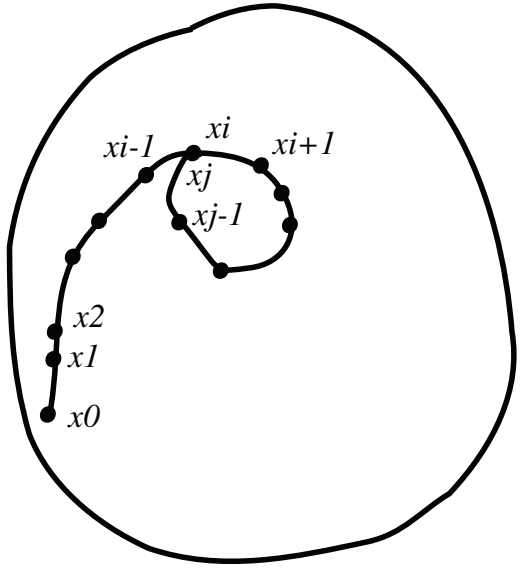
Finding collisions



Finding collisions



Finding collisions



Expected running time:

$$j = \sqrt{\frac{\pi}{2} 2^n}, i = \frac{j}{2}$$

Problems with finding collisions

- So for MD5, the expected running time, is approximately 2^{64} calculations. With todays computers, this is “feasible”.

Problems with finding collisions

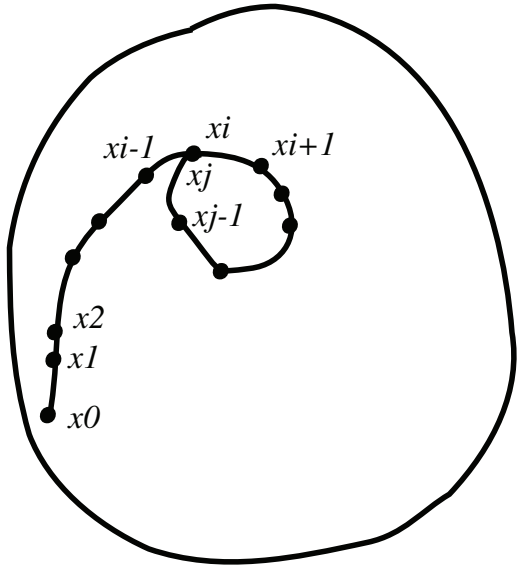
- So for MD5, the expected running time, is approximately 2^{64} calculations. With todays computers, this is “feasible”.
- However, implementing this algorithm would require a hash table containing 2^{64} entries.

Problems with finding collisions

- So for MD5, the expected running time, is approximately 2^{64} calculations. With todays computers, this is “feasible”.
- However, implementing this algorithm would require a hash table containing 2^{64} entries.
  This requires storage in the order of 10^8 terabytes!

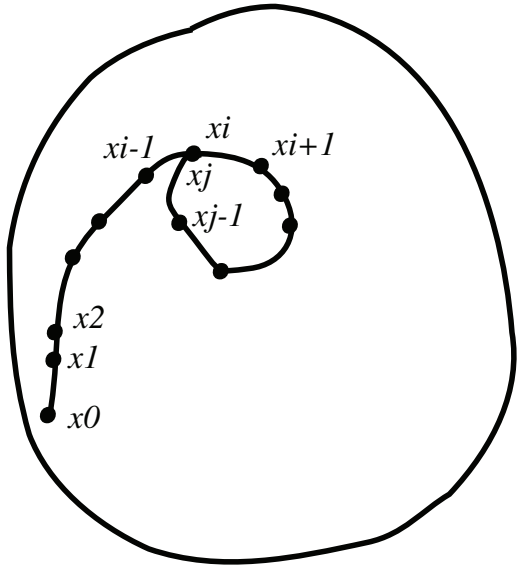
A better approach to finding collisions

In 1993, Van Oorschot & Wiener discovered an attack that had approximately the same running time, but negligible space requirements.



A better approach to finding collisions

In 1993, Van Oorschot & Wiener discovered an attack that had approximately the same running time, but negligible space requirements.

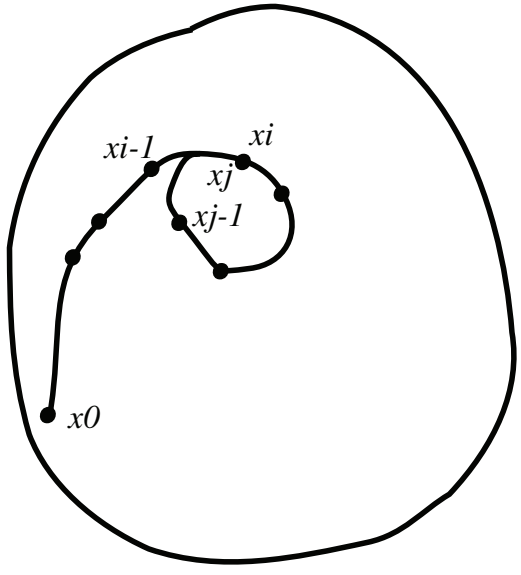


Pick some significant points, ie. The first 32 bits all zero.

($\sigma = 32$)

A better approach to finding collisions

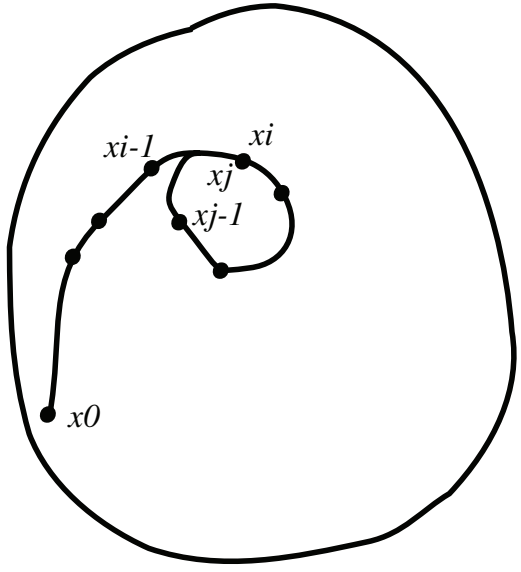
In 1993, Van Oorschot & Wiener discovered an attack that had approximately the same running time, but negligible space requirements.



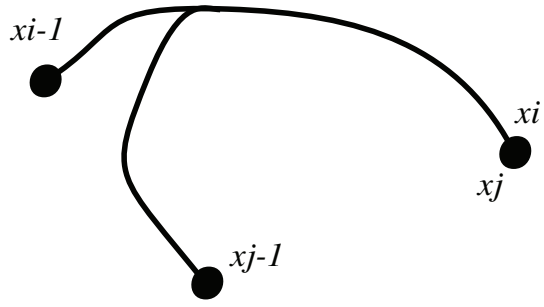
Pick some significant points, ie. The first 32 bits all zero.

($\sigma = 32$)

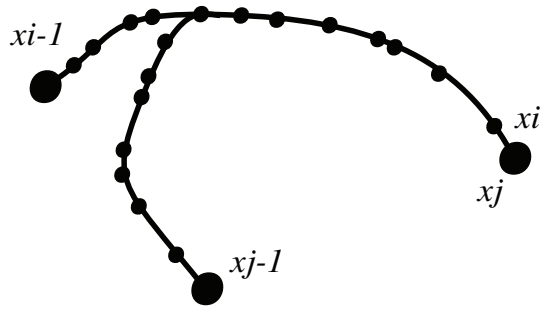
A better approach, page 2



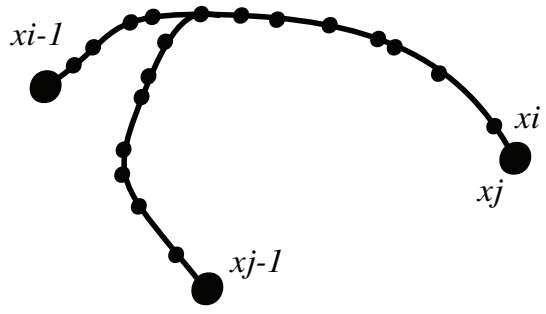
A better approach, page 2



A better approach, page 2



A better approach, page 2

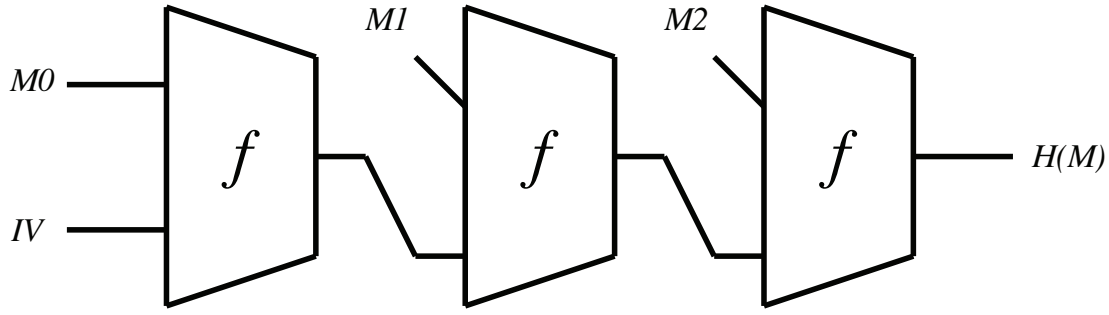


$$\sigma = 32, j = \sqrt{\frac{\pi}{2} 2^n} \approx 2^{64}$$

$$j' = \frac{j}{2^\sigma} \approx 2^{32}$$

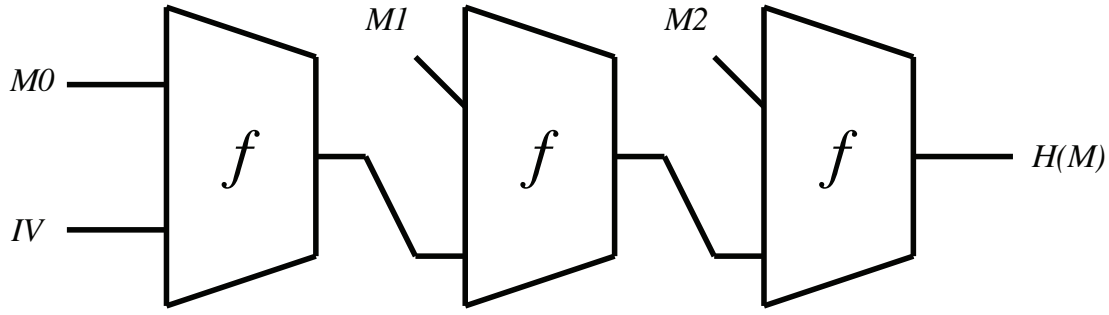
Iterated hash functions

Iterated hash functions



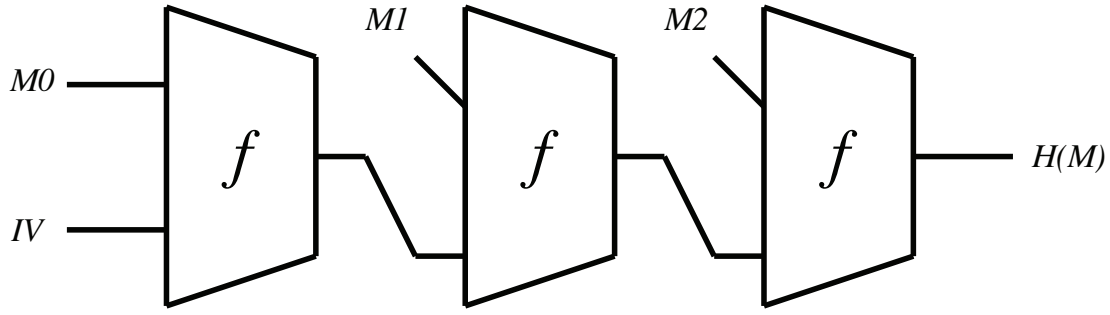
- Messages are broken up into blocks and fed in to successive iterations along with the output from the previous iteration

Iterated hash functions



- Messages are broken up into blocks and fed in to successive iterations along with the output from the previous iteration
- **MD5** (128 bits) and **SHA-1** (160 bits) are both examples of this type of hash function.

Iterated hash functions



- Messages are broken up into blocks and fed in to successive iterations along with the output from the previous iteration
- **MD5** (128 bits) and **SHA-1** (160 bits) are both examples of this type of hash function.
- More effecient algorithms of finding collisions have been found for both functions.

How do we find meaningful collisions?

How do we find meaningful collisions?

- So we can find to arbitrary (and probably useless) inputs that have the same hash value.

How do we find meaningful collisions?

- So we can find to arbitrary (and probably useless) inputs that have the same hash value.
- So what?

How do we find meaningful collisions?

- So we can find to arbitrary (and probably useless) inputs that have the same hash value.
- So what?
- We need to generate useful messages

How do we find meaningful collisions?

- So we can find to arbitrary (and probably useless) inputs that have the same hash value.
- So what?
- We need to generate useful messages
- . . . or useful versions of existing messages.

Finding meaningful messages

Finding meaningful messages

- Create a function that generates “similar” output, for two different messages, M_1 and M_2. $G_1(x)$ and $G_2(x)$

Finding meaningful messages

- Create a function that generates “similar” output, for two different messages, M_1 and M_2. $G_1(x)$ and $G_2(x)$
- Define a new hash function as a composite of this function and your existing hash function,

$$H'(x) = \begin{array}{ll} H(G_1(\frac{x-1}{2})) & \text{if } x \text{ is odd} \\ H(G_2(\frac{x}{2})) & \text{if } x \text{ is even} \end{array}$$

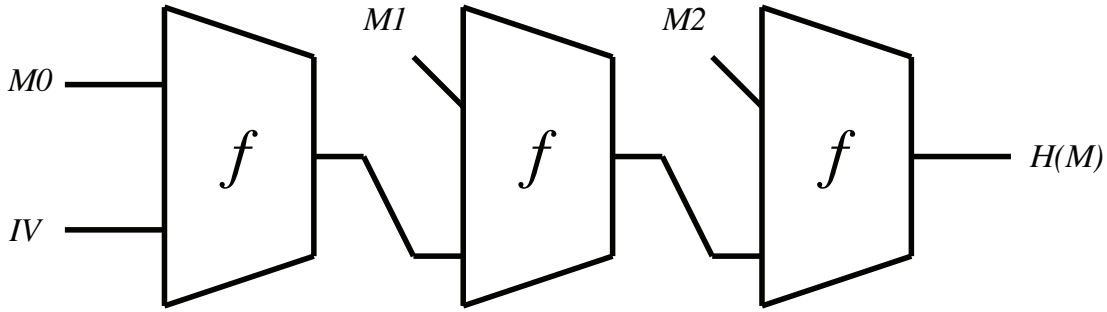
Finding meaningful messages

- Create a function that generates “similar” output, for two different messages, M_1 and M_2. $G_1(x)$ and $G_2(x)$
- Define a new hash function as a composite of this function and your existing hash function,

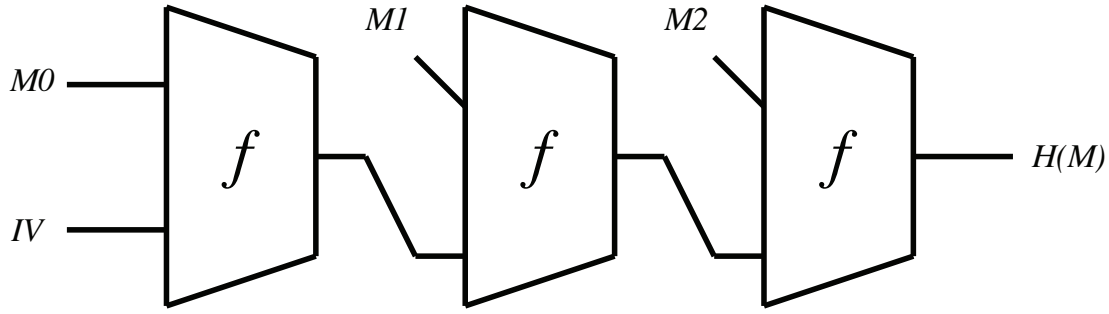
$$H'(x) = \begin{cases} H(G_1(\frac{x-1}{2})) & \text{if } x \text{ is odd} \\ H(G_2(\frac{x}{2})) & \text{if } x \text{ is even} \end{cases}$$

- Find a collision for this new function H'. 50% of the time, the two values will be one based on G_1 and one on G_2

Forging signatures

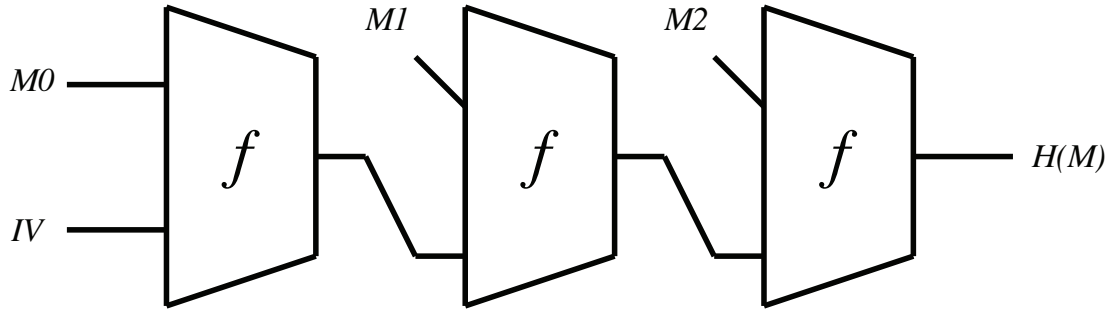


Forging signatures



- At every iteration, the output is only based on the current block, and the output from the previous block.

Forging signatures



- At every iteration, the output is only based on the current block, and the output from the previous block.
- So if for two different messages, the output at a specific stage is the same, the output from then onward is the same, if the message from then onward is the same.

Thank you!

Some references: (Well, one, anyway)

- Most work on attacking the MDx and SHA family of hashes was done by Dr. Xiaoyun Wang. There are several papers describing the attacks on her website: `http://www.infosec.sdu.edu.cn/people/wangxiaoyun.htm`