MIT Digital Currency Initiative and the University of Brasilia presents

Cryptocurrency Design and Engineering

Lecture 3: Hash Functions
Taught by: Ethan Heilman
Date: September 16, 2025
MAS.S62

Introduction

The two cryptographic primitives you need for a cryptocurrency are:

1. cryptographic hash functions
2. and digital signatures

Overview (Lectures 3 & 4)

1. Hash Functions
2. Commitments
   a. Simple commitments
   b. Merkle trees
   c. Exclusion proofs
3. Digital Signatures
   a. Lamport/Schnorr
   b. Multi-Signatures
   c. Commitments + Signatures

Hash functions

$$h: \{0, 1\}^* \to \{0, 1\}^n$$

$$h(x) \to y$$

The input is the preimage

The output is the image

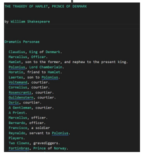
A **cryptographic hash function** is a function that maps:

- an arbitrary length input **x** to a fixed length output **y** of **n** bits
- where mapping is random-looking but deterministic (same input, same output)

Hash functions

SHA256 is a commonly used hash function, produces n= 256-bit outputs

Preimage (any length)	**→ Image (always 256-bits)**
SHA256("a")	→ 8e4621379786ef42a4fec155cd525c291dd7db3c1fde3478522f4f61c03fd1bd
SHA256("b")	→ 679e273f78fc8f8ba114db23c2dce80cc77c91083939825ca830152f2f080d08
SHA256("abcef")	→ f908c6d716117609c77e22b0d65a455b46357b7f58cb06321b5aef2be89ddaeb
SHA256("THE TRAGEDY OF HA…")	→ 40662c61c1b3e19554a79f47a9e2309def650ecbfae07a1a33da6a10ffdcd686


THE TRAGEDY OF HAMLET, PRINCE OF DENMARK

by William Shakespeare

Dramatis Personae

Claudius, King of Denmark.
Marcellus, Officer.
Hamlet, son to the former, and nephew to the present king.
Polonius, Lord Chamberlain.
Horatio, friend to Hamlet.
Laertes, son to Polonius.
Voltemand, courtier.
Cornelius, courtier.
Rosencrantz, courtier.
Guildenstern, courtier.
Osric, courtier.
A Gentleman, courtier.
A Priest.
Marcellus, officer.
Bernardo, officer.
Francisco, a soldier
Reynaldo, servant to Polonius.
Players.
Two Clowns, gravediggers.
Fortinbras, Prince of Norway.

All 192 KB of Hamlet

Hash functions are very useful because they allow us to create a deterministic fingerprint of some information

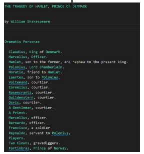
Hash functions

Hash functions allow us to create an information fingerprint

Let’s say Alice and Bob want to make sure they have the same copy of Hamlet?

- They could send over the entire file
- **OR** they could hash the file locally and compare the output (image)

SHA256(“THE TRAGEDY OF HA…”) → 40662c61c1b3e19554a79f47a9e2309def650ecbfae07a1a33da6a10ffdcd686


THE TRAGEDY OF HAMLET, PRINCE OF DENMARK

by William Shakespeare

Dramatis Personae

Claudius, King of Denmark.
Marcellus, Officer.
Hamlet, son to the former, and nephew to the present king.
Polonius, Lord Chamberlain.
Horatio, friend to Hamlet.
Laertes, son to Polonius.
Voltemand, courtier.
Cornelius, courtier.
Rosencrantz, courtier.
Guildenstern, courtier.
Osric, courtier.
A Gentleman, courtier.
A Priest.
Marcellus, officer.
Bernardo, officer.
Francisco, a soldier
Reynaldo, servant to Polonius.
Players.
Two Clowns, gravediggers.
Fortinbras, Prince of Norway.

All 192 KB of Hamlet

Hash functions

A **cryptographic hash function** is a function that maps:

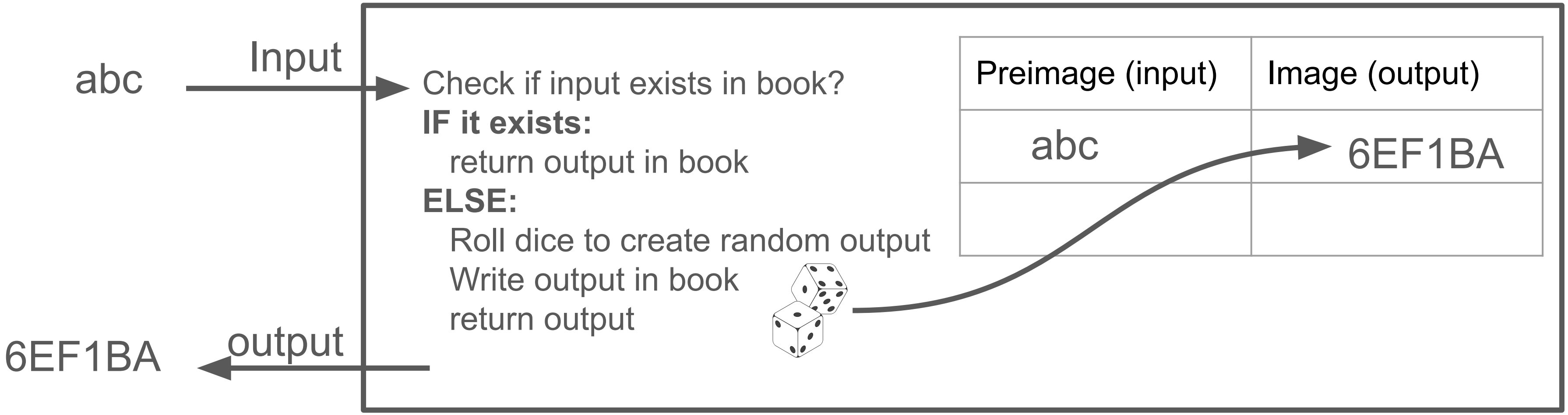
- an arbitrary length input **x** to a fixed length output **y** of **n** bits
- where the mapping is **random but deterministic**

What does this mean?

How do we define this?

Hash functions: Ideal

What would the ideal construction of a hash function look like?



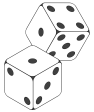
We keep adding to the book each time get a new input

Hash functions: Ideal

What would the ideal construction of a hash function look like?

abc → Input →

Check if input exists in book?
IF it exists:
  Return output in book
ELSE:
  Roll dice to create random output
  Write output in book
  Return output



Preimage (input)	Image (output)
abc	6EF1BA
qwerty	F135E7
...	....

6EF1BA ← output

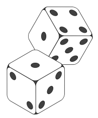
If anyone asks for a value we already have we return it

Hash functions: Ideal

This is called a Random Oracle

Input

Check if input exists in book?
IF it exists:
  Return output in book
ELSE:
  Roll dice to create random output
  Write output in book
  Return output



output

Preimage (input)	Image (output)
abc	6EF1BA
qwerty	F135E7
...	....

Random

Deterministic

Not very convenient as it assumes magic box everyone can talk with

Hash functions

What we want is a function that behaves like a Random Oracle but anyone can run on their computer.

A cryptographic hash function is considered secure IF:

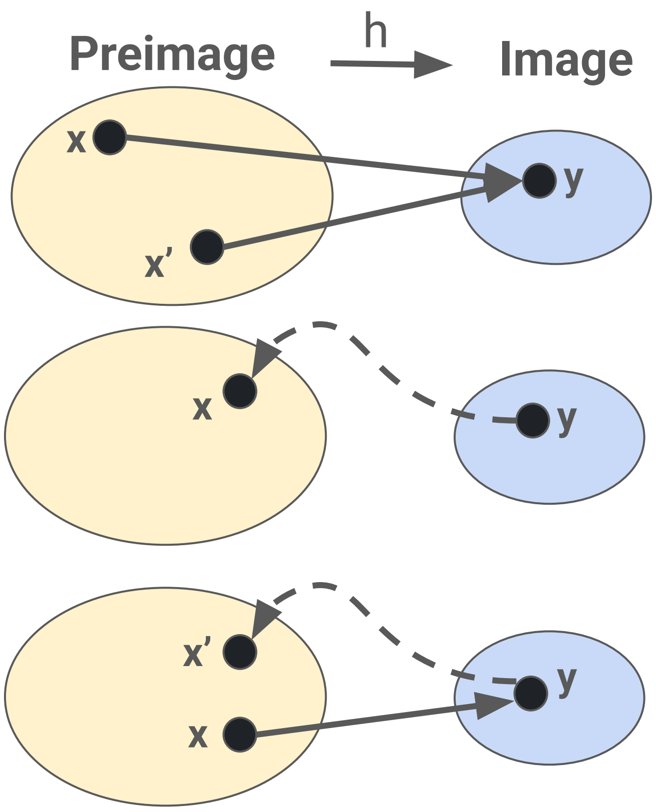
No one is able to distinguish it from a random oracle
… on inputs and outputs they haven't seen before

Hash functions

1. **<u>Collision Resistance:</u>** no one can compute
   …two diffs. inputs that result in the same output
   Find x, x’ **s.t.** x != x’ AND h(x)=h(x’)

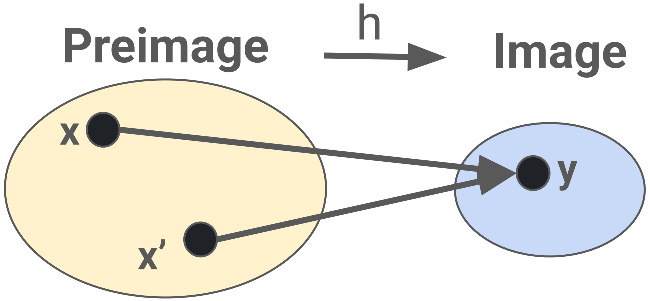
2. **<u>Preimage Resistance:</u>** no one can compute
   …an input (preimage) x from just the output y
   Given y find x **s.t.** h(x) = y

3. **<u>Second preimage resistance:</u>** no can compute
   …another preimage from given x and y
   Given x, h(x)→y find x’ **s.t.** h(x’) = y



These all must be possible, just hard to find.

Hash functions



<u>Collision Resistance:</u> no one can compute
…two diffs. inputs that result in the same output
Find x, x' **s.t.** x != x' AND h(x)=h(x')

It must be the case that collisions exist, it is just hard to find them,
…but how hard?

For a secure hash function with an n-bit output length:

$\sim 2^{n/2}$ attempts before a collision is found.

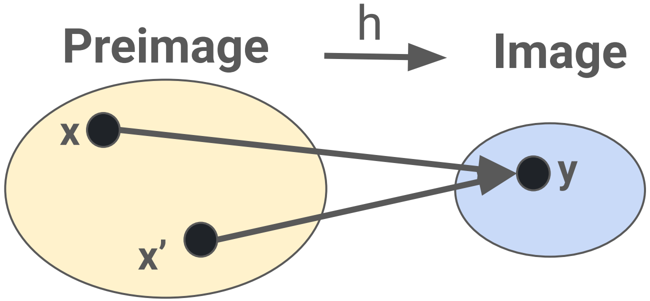
This is often called the birthday bound

Hash functions

<u>Collision Resistance:</u> no one can compute

…two diffs. inputs that result in the same output

Find x, x' **s.t.** x != x' AND h(x)=h(x')



This seems like it should NOT be possible:

- I give you a compute program,
- There is no secret key, no secrets at all
- You can run the program and look at every step
- Collisions must exist and yet you can't find them

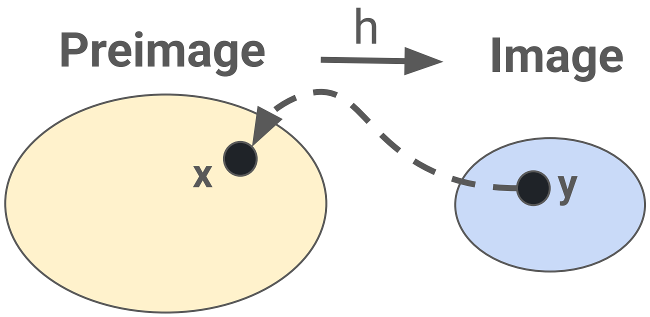
First time I heard about it, I spent all night trying to find collisions

Hash functions

<u>Preimage Resistance:</u> no one can compute

…an input (preimage) x from just the output y

Given y find x **s.t.** h(x) = y



For a secure hash function with an n-bit output length:

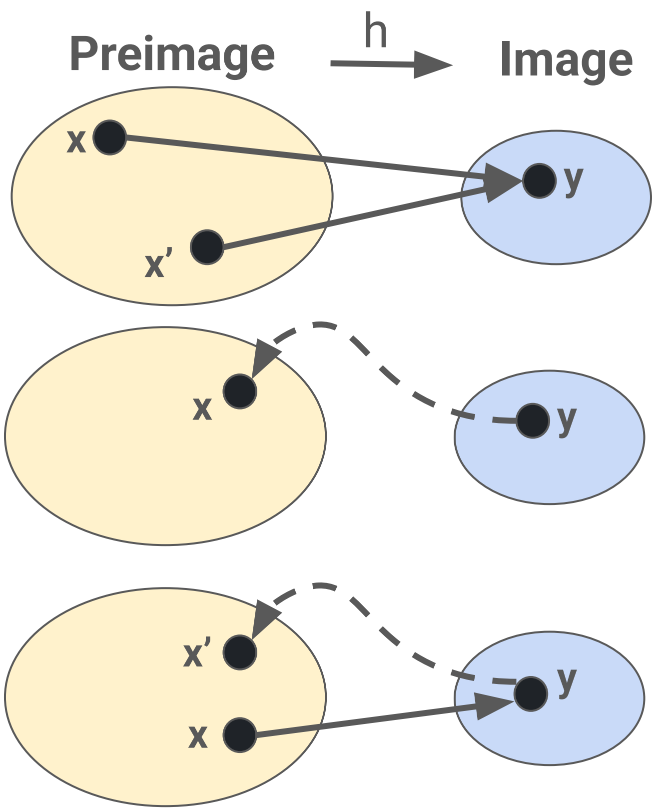
2^n attempts before a collision is found. (n=256 → 2^{256})

Hash functions

1. **Collision Resistance:** no one can compute
   …two diffs. inputs that result in the same output $2^{n/2}$
   Find x, x’ **s.t.** x != x’ AND h(x)=h(x’)

2. **Preimage Resistance:** no one can compute
   …an input (preimage) x from just the output y 2^n
   Given y find x **s.t.** h(x) = y

3. **Second preimage resistance:** no can compute
   …another preimage from given x and y 2^n
   Given x, h(x)→y find x’ **s.t.** h(x’) = y



Also should be indistinguishable from random

Hash functions: Work?

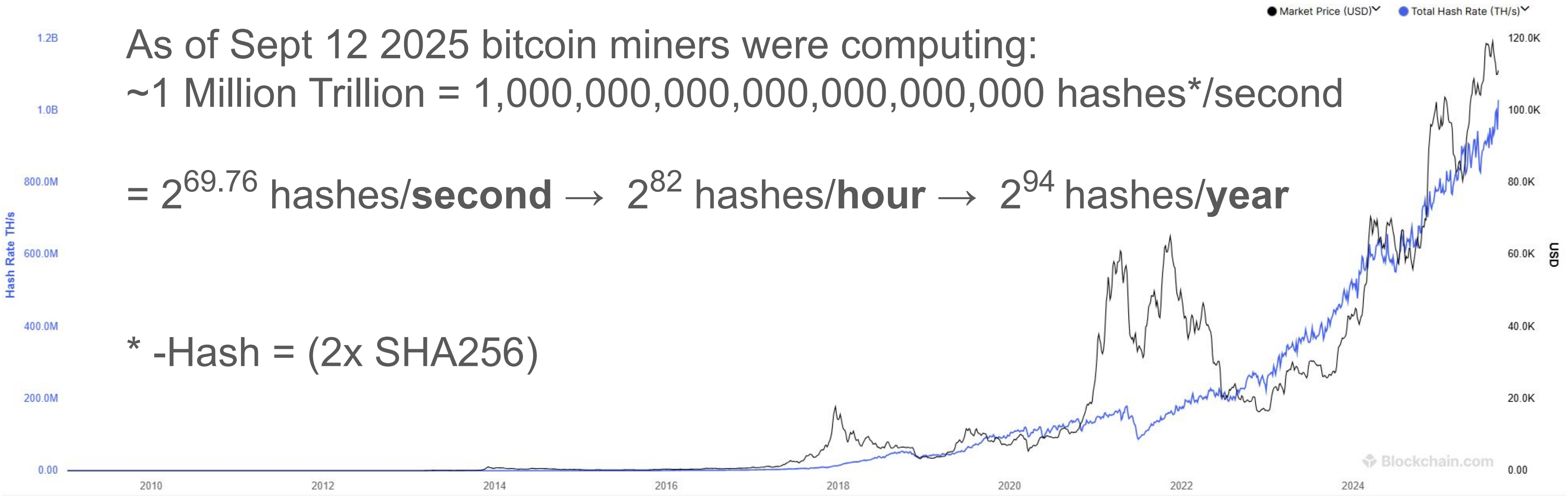
		n=160	SHA-256 (n=256)	SHA-512 (n=512)
Collisions resistance	$\sim 2^{n/2}$	$\sim 2^{80}$	$\sim 2^{128}$	$\sim 2^{256}$
Preimage resistance	2^n	2^{160}	2^{256}	2^{512}
2nd preimage…	2^n	2^{160}	2^{256}	2^{512}

Hash functions: Work?

As of Sept 12 2025 bitcoin miners were computing:
~1 Million Trillion = 1,000,000,000,000,000,000,000 hashes*/second

= $2^{69.76}$ hashes/**second** → 2^{82} hashes/**hour** → 2^{94} hashes/**year**

* -Hash = (2x SHA256)



Hash functions: Powers of 2?

2^{79}	Stars in the known universe
2^{82}	Bitcoin network hashes/hour
2^{94}	Bitcoin network hashes/year
2^{104}	Bitcoin network hashes/1000 years
2^{124}	Bitcoin network hashes/1 billion years
2^{128}	Collision resistance (SHA256)
2^{166}	Atoms in the earth
2^{256}	Preimage resistance (SHA256)
2^{265}	Atoms in known universe
2^{512}	Preimage resistance (SHA512)

Hash functions:

How long would it take Bitcoin ($2^{69.76}$ hashes/sec) to break these?

		n=160	SHA-256 (n=256)	SHA-512 (n=512)
Collisions resistance	$\sim 2^{n/2}$	$\sim 2^{80}$ **15 mins**	$\sim 2^{128}$ **17 billion years**	$\sim 2^{256}$ **2^{164} years**
Preimage resistance	2^n	2^{160} **78 billion billion years**	2^{256} **2^{164} years**	2^{512} **2^{418} years**
2nd preimage…	2^n	2^{160} **78 billion billion years**	2^{256} **2^{164} years**	2^{512} **2^{418} years**

Our universe is 2^{34} years old, this is 2^{130} = 136,112,946,768,375,385,385,349,842,972,707,284,582 times longer

These are lowerbounds that assume infinite space

Hash functions:

How to find collisions via brute force:

1. Hash an input,
2. Check list of previously seen hashes
3. IF YES: Collision found
4. IF NO: add hash and value to previously seen hashes

collision!

h(777) → $\boxed{\text{8d0e479671…}}$

Input: x	Output: y ← h(x)
123	01f9bf4bb49…
456	$\boxed{\text{8d0e479671…}}$
789	a6343f75e3..

For n=160 requires > 2^{80} storage!!!

https://www.qwantz.com/index.php?comic=854

Hash functions:

For collisions via simple brute force

n=160 requires $>2^{80}$ storage!!! >90 trillion 12 TB hard drives!

		n=160	SHA-256 (n=256)	SHA-512 (n=512)
Collisions resistance	$\sim 2^{n/2}$	$\sim 2^{80}$ 90 trillion HDs **15 mins**	$\sim 2^{128}$ **17 billion years**	$\sim 2^{256}$ **2^{164} years**
Preimage resistance	2^n	2^{160} **78 billion billion years**	2^{256} **2^{164} years**	2^{512} **2^{418} years**
2nd preimage…	2^n	2^{160} **78 billion billion years**	2^{256} **2^{164} years**	2^{512} **2^{418} years**

Hash functions:

Better approaches:

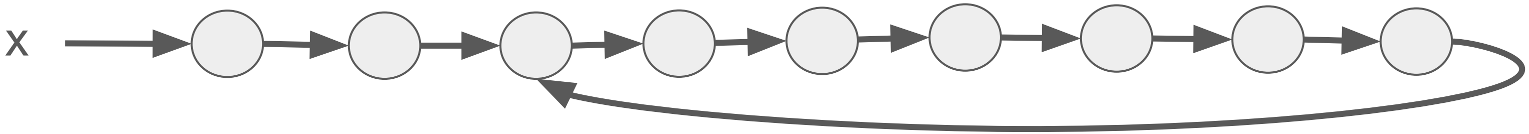
1. Trade-off memory for compute
   Works for things that are practical in time but not space

2. Attack the function itself
   Don’t wait the age of the universe

Hash functions:

Observation: if you repeatedly hash a value, collisions be cycles

H(x) → y0, H(y0) → y1, H(y1) → y2, …



Keep a list of every N-th output you create,
then you detect when a collision has happened

There are much better but more complex time-memory collision algorithms such as Oorschot and Wiener's Parallel Collision Search with Cryptanalytic Applications (1996),

Hash functions:

Everything we have looked at is generic and assumes the hash function is secure, but what if it isn't?

Collisions for the compression function of MD5

Bert den Boer
Philips Crypto B.V.
P.O. Box 218
5600 MD Eindhoven
The Netherlands

Antoon Bosselaers
ESAT Laboratory, K.U. Leuven
Kard. Mercierlaan 94
B-3001 Heverlee, Belgium
antoon.bosselaers@esat.kuleuven.ac.be

Abstract. At Crypto '91 Ronald L. Rivest introduced the MD5 Message Digest Algorithm as a strengthened version of MD4, differing from it on six points. Four changes are due to the two existing attacks on the two round versions of MD4. The other two changes should additionally strengthen MD5. However both these changes cannot be described as well-considered. One of them results in an approximate relation between any four consecutive additive constants. The other allows to create collisions for the compression function of MD5. In this paper an algorithm is described that finds such collisions.
A C program implementing the algorithm establishes a work load of finding about 2^{16} collisions for the first two rounds of the MD5 compression function to find a collision for the entire four round function. On a 33MHz 80386 based PC the mean run time of this program is about 4 minutes.

1993 MD5 weakness found

Collisions for Hash Functions

MD4, MD5, HAVAL-128 and RIPEMD

Xiaoyun Wang[1], Dengguo Feng[2], Xuejia Lai[3], Hongbo Yu[1]
The School of Mathematics and System Science, Shandong University, Jinan250100, China[1]
Institute of Software, Chinese Academy of Sciences, Beijing100080, China[2]
Dept. of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai, China[3]
xywang@sdu.edu.cn[1]
revised on August 17, 2004

1 Collisions for MD5

MD5 is the hash function designed by Ron Rivest [9] as a strengthened version of MD4 [8]. In 1993 Bert den Boer and Antoon Bosselaers [1] found pseudo-collision for MD5 which is made of the same message with two different sets of initial value. H. Dobbertin[3] found a free-start collision which consists of two different 512-bit messages with a chosen initial value IV_0'.

$IV_0': A_0' = 0x12AC2375,\ B_0' = 0x3B341042, C_0' = 0x5F62B97C, D_0' = 0x4BA763ED$

Our attack can find many real collisions which are composed of two 1024-bit messages with the original initial value IV_0 of MD5:

$$IV_0 : A_0 = 0x67452301, B_0 = 0xefcdab89, C_0 = 0x98badcfe, D_0 = 0x10325476$$

$$M' = M + \Delta C_1, \Delta C_1 = (0,0,0,0,2^{31},...,2^{15},...,2^{31},0)$$

$$N_i' = N_i + \Delta C_2, \Delta C_2 = (0,0,0,0,2^{31},...,-2^{15},...,2^{31},0)$$

(non-zeros at position 4,11 and 14)

such that

2004 - MD5 broken

CWI, Google announce first collision for Industry Security Standard SHA-1

by 'industry Deprecation Proved To Be Too Slow', Centrum Wiskunde & Informatica

Credit: shatterered.io

Today, researchers at the Dutch research institute CWI and Google jointly announce that they have broken the SHA-1 internet security standard in practice. This industry standard is used for digital signatures and file integrity verification, which secure credit card transactions, electronic documents, GIT open-source software repositories and software [...]: "Many applications still use SHA- [...] 2011

2017 - SHA1 collisions

Hash functions: Uses

Uses for hash functions

- As a secure pointer to some information
- For PoW as we saw in the last class
- To hide passwords:

Bob h("sekrit p4ssw0rd") →8be37df…

user=bob, password hash= 8be37df…

User	Hash of password
root	47a9e23…
alice	ee7497b…
bob	8be37df…

We can do much better things for passwords than just hashing