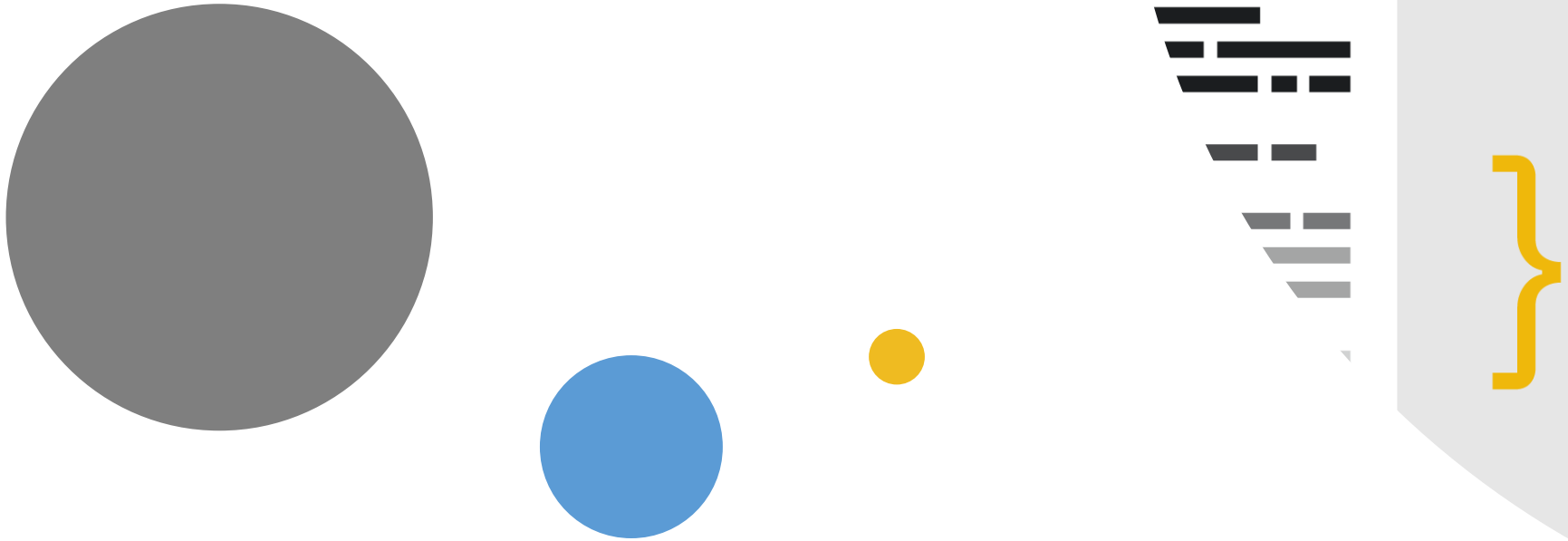


41900 – Fundamentals of Security

Hash Function and Basic Ciphers

Ashish Nanda

Ashish.Nanda@uts.edu.au

Context: Applied Cryptography

Cryptography is the study of mathematical techniques related to the design of ciphers.

Cryptanalysis is the study of breaking them.

Cryptology (or crypto) is the study of both i.e. cryptography and cryptanalysis.

Crypto building blocks are otherwise known as cryptographic primitives. For example:

- hash functions
- block ciphers
- stream ciphers
- digital signatures
- random number generators

What is a Function

A function **f : X → Y** is defined by:

- The domain, a set **X = {x_1, x_2, . . . , x_n}**.
- The codomain, a set **Y = {y_1, y_2, . . . , y_m}**.
- A rule **f** assigning each element of **X** to an element of **Y**.

Example: let a function **f : {−1, 0, 1} → {0, 1, 2}** be defined by **$f(x) = x^2$** .

- **f(−1) = 1**
- **f(0) = 0**
- **f(1) = 1**

Properties of Functions

When **f : X → Y** is a function,

Taking **f : {−1, 0, 1} → {0, 1, 2}** where **$f(x) = x^2$**

- The image of **x ∈ X** (domain) is called **f(x)**, an element of **Y**(codomain).
  - Image of **f(−1) = 1 ,** here **-1 ∈ X** and **1 ∈ Y**

- The range of **f** is the set of all images, and is a subset of **Y**.
  - The range of **f** is **{0, 1}**.

- If **f(x) = y**, then **x** is called a preimage of **y**.
- The set of all preimages of **y** is written **$f^{-1}(\{y\})$**.
  - The preimage of **1** is **$f^{-1}(\{1\}) = \{-1, 1\}$**.
  - The preimage of **2** is **$f^{-1}(\{2\}) = \{\}$**.

A ‘One Way’ Functions

Write $\{0, 1\}^n$ for the set of all binary strings of length **n**.

For example:

- $\{0, 1\}^1 = \{0, 1\}$.
- $\{0, 1\}^2 = \{00, 01, 10, 11\}$.

A function $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ is one way if:

- It is “*easy*” to compute **f(x)** for all $x \in X$
- It is “*computationally infeasible*” to find a preimage.

Intuitively:

- Given **x**, it is easy to compute **f(x)**
- Given **f(x)**, it is hard to compute **x**

One Way Function Examples

- **Example**
  1. Write a message on the side of a plate: **x**
  2. Smash the plate: **f(x)**
  3. Finding the inverse $\mathbf{f^{-1}(x)}$ is difficult (but not impossible)

- **Data Encryption Standard Cipher (DES)**
  - **f(x) = DES(m, k) = c**
  - Given **c** and **m**, it is difficult to find **k**

Hash Functions

A hash function **(h)**, is an efficiently computable mapping of arbitrarily long strings to short fixed length strings.

Minimum properties:

- Compression

  Any number of input bits to a fixed number of output bits

  Example: SHA256 has fixed output of 256 bits

- Easy of computation

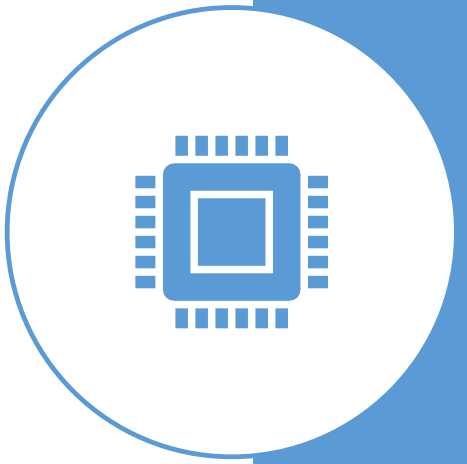
  Given **h** and **x**, it is easy to compute **h(x)**.

Hash Functions

Keyed Hash Functions

Some hash functions take both a key **(k)** and a message **(m)**

$\mathbf{MAC_k(m) = h(m, k)}$



There are two types used

- Message authentication codes (**MAC**)
- Hash-based message authentication codes (**HMAC**).

Properties of Secure Hash Functions

Not all hash functions are secure. In cryptography we consider secure hash functions.

Let **h : X → Y** be a hash function. In order to be secure, it must satisfy the following properties:

Preimage Resistance

- Given **y** it is “hard” to find a preimage **x** such that **h(x) = y**.

Second Preimage Resistance

- Given a particular **x**, it is “hard” to find **x’ ≠ x** such that: **h(x’) = h(x) = y**.

Collision Resistance

- It is “hard” to find any pair **x ≠ x’** such that **h(x)** = **h(x’)**.

Properties of Secure Hash Functions

A one way hash function satisfies

- **Preimage Resistance**
- **Second Preimage Resistance**

A collision resistant hash functions satisfies

- **Collision Resistance**
  - In doing so It also satisfies **Second Preimage Resistance**

Some Uses of hash functions

Hash functions are extremely useful for confirmation of knowledge without revealing what you know.

Example: Lets say Jane and Bob both know a secret, however they are not sure if they know the same secret.

- **Challenge:** Neither Jane nor Bob want to confirm they know the same secret by telling the other person their secret.
- **Solution:** Any one person (Say Bob) can create a hash of his secret and send it to Jane. Jane can then create a hash of her secret and compare it with the hash she received from Bob
- **Justification:** If the Secret is the same, the hash will match and both can be sure they have the same.

Some Uses of hash functions

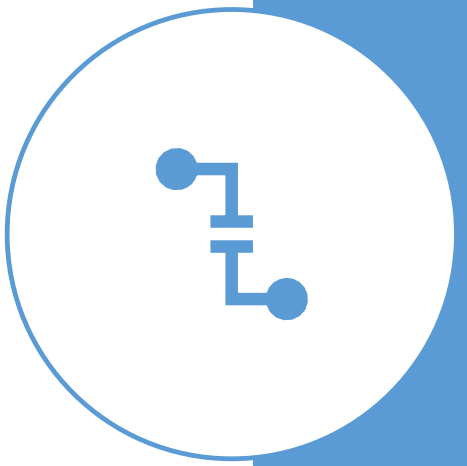
Hash functions are also good for storing passwords

- Instead of storing the password as plaintext, store it's hash in the database/file.
- When a user tries to login, calculate a hash of their password and compare to the hash stored in the database/file.
- If the password file gets stolen, the hash needs to be reversed before the attacker is able to crack the passwords.

Some Uses of hash functions

Virus Protection & Host Intrusion Detection

- For each file **x**, **h(x)** is stored off system on a database.
- Periodically, the system will calculate the hash for all files and check if it matches with the ones stored in the database.
- **Second Preimage Resistance** is critical as it should be hard to find **x'** such that **h(x) = h(x')**



Real World Hash Functions

Name	bits	h("lolcats")
MD2	128	4301aae7e3e791826b53b952859d0a14
MD4	128	52bb2839f24583f5af2fe74522db3e2e
MD5	128	c8ba0a4b74948d105bdb6f77b77a432e
RIPEMD	160	3dd2dec7cecec77219f644788e81ff26d328423c
SHA-1	160	ec9c175f8e3780cec9e93b66aea4f98b200764de
SHA224	224	c995ce647c889fdc3bbc7c8e4b43b3f5b5c3faf1525b640abc60ce54
SHA256	256	e06297effe5bcf6af177cead11f5c5d4a73777590a7a98a464287d5a6a7cdc2a
SHA384	384	14c41e98d0fb9b357922274adb9f70352f601d7b56aac8e4... ...39ed860b634b31a7c0f56e3d63284cfd4d04bde07ff3351d
SHA512	512	9a0552b9d165360fc08090a88f8c5274ca263c485417c73acb5c1a820b288549... ...12f8885bebbd49f9c229eac9be43441e061408f99e6e25dafaa5c4a946f50693

Attacks on Hash Functions

To **brute force** in cryptanalysis is to search the entire space of possible alternatives.

A subset of this is a **dictionary attack** where we throw subsets of the key space (dictionaries) at the problem.

Brute force to attack preimage resistance:

- Say a hash produces an **n**-bit output: **h(x) = y**
- We must try $\mathbf{2^{n-1}}$ hashes before the probability $\mathbf{Pr[h(a) = y] \geqslant 0.5}$ (where $\mathbf{a \in Z}$)
- Example: if **a** is one of $\mathbf{2^{10} = 1024}$ possibilities, you have to try half of them $\mathbf{2^9 = 512}$ on average before you can find a match **y** such that **h(a) = y = h(x)** and therefor **a = x**

SHA-1 Break

Recently February 23, 2017!

Found by **Google** and **CWI**

- **Complexity:**
  - 9, 223, 372, 036, 854, 775, 808 SHA-1 compressions
  - Nine **QUINTILLION**
- **Comparisons:**
  - Shattered: 110GPU = 1 year
  - Brute Force: 12 million GPU = 1 year
- **What is affected?**
  - Digital Certificates, Email, Software Updates, **GIT**.

The Current State: Collision Resistance

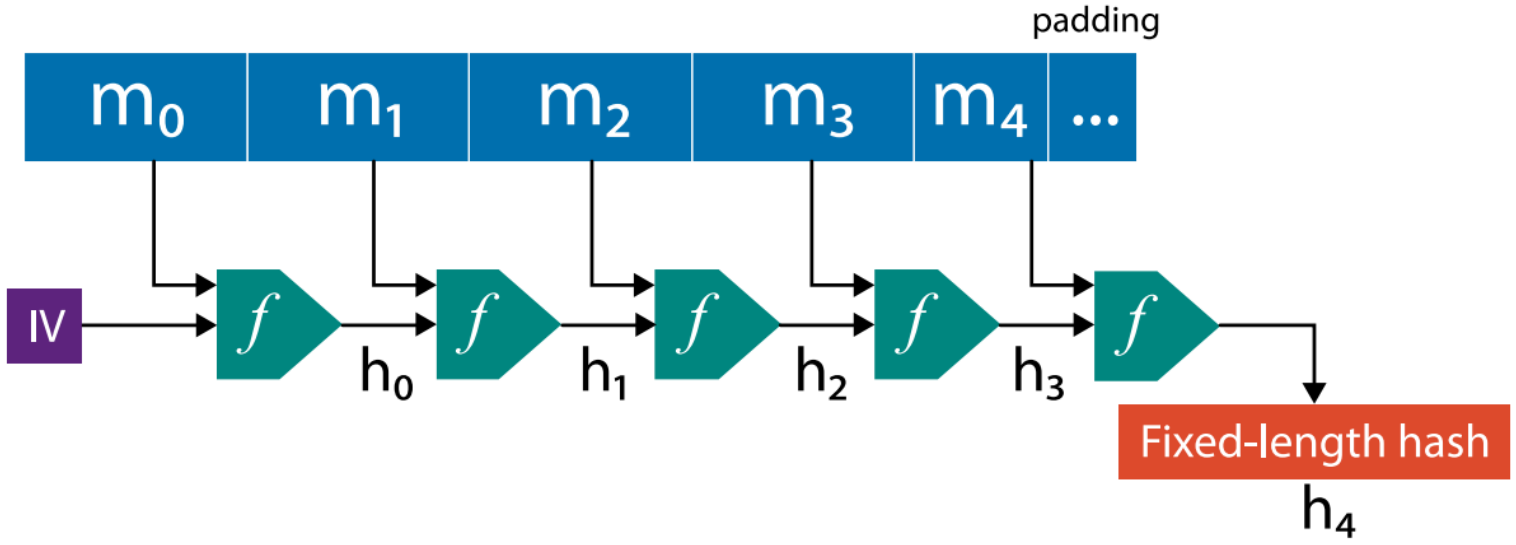
Hash Function	Security Claim	Best Attack	Publish Date	Comment
MD5	2^{64}	2^{18} time	25-03-2013	This attack takes seconds on a regular PC. Two-block colliusions in 2^{18}, single block collisions in 2^{41}
SHA-1	2^{80}	$2^{60.3} \ldots 2^{65.3}$	19-06-2012 23-02-2017	SHAttered was the first public release of a collision. Attack is feasible with large amounts of computation.
SHA256	2^{128}	31/64 rounds ($2^{65.5}$)	28-05-2013	
SHA512	2^{256}	24/80 rounds ($2^{32.5}$)	25-11-2008	

The Current Stat: Preimage Resistance

Hash Function	Security Claim	Best Attack	Publish Date
MD5	2^{128}	$2^{123.4}$	27-04-2009
SHA-1	2^{160}	45 of 80 rounds	17-08-2008
SHA256	2^{256}	43 of 64 rounds ($2^{254.9}$ time, 2^{6} memory)	10-12-2009
SHA512	2^{512}	46 of 80 rounds ($2^{511.5}$ time, 2^{6} memory)	25-11-2008

Iterated Hash Construction

Merkle-Damgård construction is a technique for building hash functions.



- f is a one-way compression function.
- h_n is the hash generated at each step
- IV is the initialization vector
- Message M is divided into n fixed-bit blocks m_n
- To maintain fixed block size, partial blocks pass through a padding function.

Sample Output

MD5

Input	Hash Value (as hex byte string)
“”	d41d8cd98f00b204e9800998ecf8427e
“a”	0cc175b9c0f1b6a831c399e269772661
“abc”	900150983cd24fb0d6963f7d28e17f72

SHA-1

Input	Hash Value (as hex byte string)
“”	da39a3ee5e6b4b0d3255bfef95601890afd80709
“a”	a9993e364706816aba3e25717850c26c9cd0d89d
“abc”	a9993e364706816aba3e25717850c26c9cd0d89d

Keyed Hash Functions (MACs)

Message Authentication Codes (MACs) are a one-way hash function with the addition of a key
$\mathbf{h_k : \{0, 1\}^* \rightarrow \{0, 1\}^n}$

The key is secret and necessary to verify the hash $\mathbf{h_k(m)}$ and can be
thought of as a cryptographic checksum.

Goals:

- Provides message authentication where sender and receiver share a secret.
- An eavesdropper cannot fake a message with a valid MAC.
- Used for message integrity, not message secrecy.

Properties of MACs

- Given **m** and **k** it is easy to construct $\mathbf{h_k(m)}$.
- Given pairs of messages and MAC $\mathbf{(m_i, h_k(m_i))}$ it is hard to construct a valid new pair $\mathbf{(m_j, h_k(m_j))}$ where $\mathbf{m_j \neq m_i}$ Without knowledge of **k**.

More Formally: A MAC is **(ε, t, q, l)**.

- It is secure if Given **q** pairs, each of length $\leqslant$ **l**, in time **t** an adversary can succeed in constructing new (message, MAC) pairs with probability < **ε**.

Using MACs - Over the Network

- Alice and Bob share a secret key **k**
- They use MAC to verify the data they share with each other
- A hacker tries to impersonate Alice and send a message **m** to Bob containing a malware along with the MAC using some key **k'** where **k ≠ k'**
- Upon receiving the message, Bob will calculate the MAC on the message using the secret key **MAC(m) = h_k(m)**.
- Bob will then compare the calculated MAC with the MAC received and realize something is wrong as **h_k(m) ≠ $h_{k'}$(m)**.
- This is because the hacker does not have correct key **k**, so the MAC generated by the attacker is not the same as generated by Bob.

Constructing MACs

Cryptographic

Non-Keyed hash functions (HMAC) – **fast**

Block cyphers (CBC-MAC) – **slow**

Information Theoretic

Based on universal hashing (outside of scope)

Hash Based MAC (HMAC)

MAC based on non-keyed hash function h

Attempt 1: MAC_k (m) = h(k,m)

Insecure: attacker can arbitrarily add to the end of the message. (Merkle-Damgård construction)

Attempt 2: MAC_k (m) = h(m,k)

Insecure: vulnerable to the birthday attack!

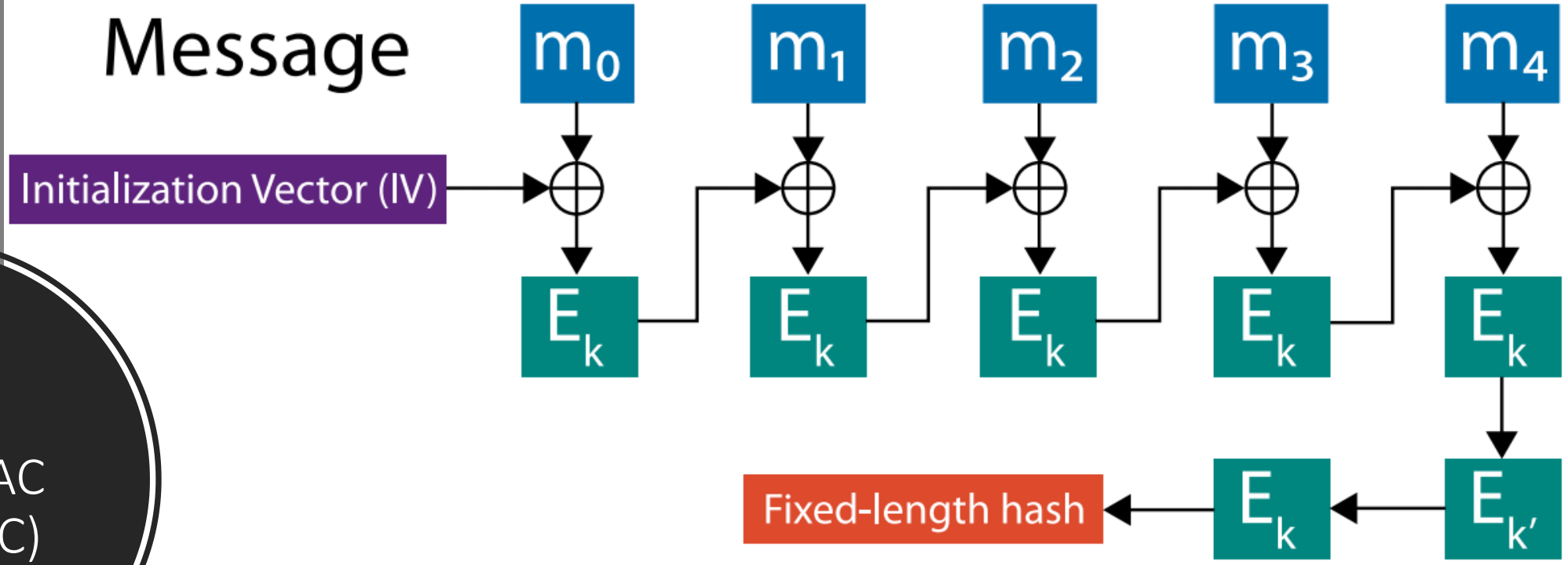
Attempt 3: $MAC_{k,k'}$ (m) = h(k,m,k')

More secure: enveloped method

BEST: $MAC_{k,k}$ (m) = h ((k ⊕ opad) + h((k ⊕ ipad) + m))

- **opad** is the **outer** padding
- **ipad** is the **inner** padding

Cipher-Based MAC (CBC-MAC)



CBC-MAC uses a technique known as Cipher Block Chaining.

Turn a message into blocks and repeat encryption using a block cipher is **XORed (⊕)**

Secret key = **(k, k', IV)** **IV:** Initialisation Vector (random bits)

CBC-MAC Length

Name	Key Size (bits)	Hash length (bits)	Relative Speed	Class	Notes
Blowfsh	Up to 448	64	23	Block Cipher	Bruce Schneier
DES	56	64	10.6	Block Cipher	Lucifer/NSA
3DES	112	64	3.7	Block Cipher	Banking
IDEA	128	64	11.8	Block Cipher	Massey and Lai
RC5 (r=12)	Up to 2048	32, 64, 128	19.6	Block Cipher	Ron Rivest (RSA)
AES (r=10, 128 bits)	128,192,256	128,192,256	21.1	Block Cipher	Rijndael
CRC32	-	32	173	Checksum	Very weak - linear
MD4	-	128	176	Hash Function	Ron Rivest (RSA)
MD5	-	128	127	Hash Function	Ron Rivest (RSA) Block collisions
SHA-1	-	160	81.5	Hash Function	NSA Hash Collisions

Keep up-to-date

There has been a steady stream of breaks against popular hashing functions like MD5 and SHA-1.

Pay attention to the new hash functions that are currently are recommended and also the ones that have been deprecated.

In 2012, NIST ran a competition to choose the latest generation of hash functions, now known as the SHA-3 family.

Ciphers

Substitution Ciphers

Substitution ciphers are the oldest form of cipher.
The secret key consists of a table which maps letter substitutions between plaintext and ciphertext.
The most famous is the **Caesar cipher** where each letter is shifted by 3 **(modulo 26)**:

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Similar to **ROT13** which shifts plaintext 13 places – largest advantage is that encrypting twice results in the plaintext:

ROT13 (ROT13(m)) = m

Substitution Ciphers

There are **26!** (factorial) different possible keys **(≈ 2^{88} or 88-bits)**.

Monoalphabetic (single character) substitution cipher:

Substitution ciphers are easy to break using frequency analysis of the letters:

- Single letters
- Digraphs (pairs of letters)
- Trigraphs (three letters)

Src	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**	**y**	**Z**
Key	X	N	Y	A	H	P	O	G	Z	Q	W	B	T	S	F	L	R	C	V	M	U	E	K	J	D	I
m	t	h	i	s	c	o	u	r	s	e	r	o	c	k	s	t	h	e	b	l	o	c	k			
c	M	G	Z	V	Y	F	U	C	V	H	C	F	Y	W	V	M	G	H	N	B	F	Y	W			

Homophonic Ciphers

Homophonic ciphers are substitution ciphers that replace a common letter with multiple symbols (i.e. **E** can go to **[C, ε, O])**

Peaks or troughs in the letter frequency are hidden as they're broken down into multiple smaller spikes.

- "**D**" would become "**F**"
- "**E**" would be randomly chosen as one of: **[Z, 7, 2, 1].**

As a result, the high frequency of the letter "**E**" (the most common letter in English) is spread amongst several characters, making frequency analysis much more difficult.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	X	S	F	Z	E	H	C	V	I	T	P	G	A	Q	L	K	J	R	U	O	W	M	Y	B	N
9				7				3					5	0				4	6						
				2																					
				1																					

Homophonic Ciphers

Still difficult to decipher even using modern computing:

- Success rate is measured in terms of alphabet size and ciphertext length.

Deciphered using nested hill climbing (heuristic algorithm / educated guessing):

- Outer layer determines the number of symbols each letter maps to
- Inner layer determines the exact mapping

Vigenere Cipher

- Originated in Rome in the sixteenth century
- A Vigenere cipher is a polyalphabetic substitution cipher (made of multiple monoalphabetic substitution ciphers).
- The secret key is a repeated word with encryption performed by adding the key modulo 26.

Plaintext:	**l**	**a**	**u**	**n**	**c**	**h**	**m**	**i**	**s**	**s**	**i**	**l**	**e**	**s**	**a**	**t**	**l**	**o**	**s**	**a**	**n**	**g**	**e**	**l**	**e**	**s**
Keystream:	c	r	y	p	t	o	c	r	y	p	t	o	c	r	y	p	t	o	c	r	y	p	t	o	c	r
==========																										
Ciphertext:	n	r	s	c	v	v	o	z	q	h	b	z	g	j	y	i	e	c	u	r	l	v	w	z	g	J

L + C = 11 + 2 = 13 mod 26 = 13thchar →N

N + Y = 13 + 24 = 37 mod 26 = 11thchar →L