COMPUTER & NETWORK SECURITY

Lecture 2:

Hash Functions

APPLIED CRYPTOGRAPHY

Cryptography is the study of mathematical techniques related to the design of cyphers

Cryptanalysis is the study of breaking them

Cryptology (or crypto) is the study of both

Crypto building blocks are otherwise known as **cryptographic primitives**

e.g. hash functions, block cyphers, stream cyphers, digital signatures

WEAK VS STRONG CRYPTOGRAPHY

There are two types of crypto in the world:

Crypto that stops your kid sister from reading your e-mail

Crypto that stops major governments from reading your e-mail and tracking your activities

We are concerned with the latter.

FUNCTIONS

A function f : X → Y is defined by

Two sets X (domain) and Y (codomain)

A rule f

If x ϵ X then

The image of x is the element in Y which rule f associates with x

The image y of x is denoted by y = f(x)

If y ϵ Y then

A preimage of y is an element x ϵ X for which f(x) = y

The set of elements in Y which have at least one preimage is called the image of f, or Im(f)

ONE WAY FUNCTIONS

A function f: $f(x) = \{0,1\}^n \rightarrow \{0,1\}^m$ is one way (OWF) if:

It is “easy” to compute f(x) for all $x \in X$

It is “computationally infeasible” to find any $x \in X$ given “essentially all” elements $y \in Im(f)$

That is, given a random $y \in Im(f)$, it is computationally infeasible to find any $x \in X$ such that $f(x) = y$

Intuitively:

Given x it is easy to compute f(x)

Given f(x) it is hard to compute x

ONE WAY FUNCTION EXAMPLES

Example:

Write a message m on the side of a plate

Drop the plate [f(m)]

Finding the inverse is difficult (but not impossible)

f(m) = E(m, k)

Where E is the Data Encryption Standard (DES) cypher

Given message m and DES(m, k) it is hard to find key k

f(m) = RSA(m, e, n) = m^e mod n

Represent message m as a number

e (encryption key) is public

n = pq is public where p and q are both large primes (but p & q are secret)

e.g. f(m) = m^3 mod (48611 · 53993)

This will make more sense later...

TRAPDOOR ONE WAY FUNCTIONS

A one-way function with a secret trapdoor

If you know it, you can easily compute x from f(x)

Also known as:

- Compression function
- Message digest
- Cryptographic checksum
- Fingerprint

Intuitively: it is easier to put a jigsaw puzzle back together if you have the plans

Consider $f_{n,e}(m) = RSA(m,e,n) = m^e \bmod n$ (p, q large primes)

- Where m is the message you want to keep secret, represented by a number
- If p and q are known, it is much easier to compute m from f(m)
- Again, this will make more sense later...

HASH FUNCTIONS

A hash function, *h*, is an efficiently computable mapping of arbitrarily long strings to short fixed length n-bit strings

Minimum properties:

Compression (typically n bits to 128 bits e.g. MD4, MD5)

Ease of computation, given h and x, h(x) is easy to compute

There are two classes of hash functions:

Unkeyed (sometimes known as message detection codes: **MDC**)

MDC = h(x)

Keyed (sometimes known as message authentication codes: **MAC**)

MAC = h(x, k) where k is a key

PROPERTIES OF HASH FUNCTIONS

Hash functions have the following desired properties:

1. **Preimage resistance**
   Given y it is "hard" to find a preimage x such that h(x) = y
   For all g ϵ time (t), Probability Pr_y [h(g(y)) = y] < ε

2. **Second preimage resistance**
   Given x it is "hard" to find x' ≠ x such that h(x) = h(x')
   For all g ϵ time (t), Pr_x [h(g(x))=h(x) and g(x) ≠ x] < ε

3. **Collision resistance**
   It is "hard" to find x ≠ x' such that h(x) = h(x')
   Pr_r[g(r) = (x,x') such that h(x) = h(x') and x ≠ x'] < ε

Note: 3 => 2 since (not 2) => (not 3)

PROPERTIES OF HASH FUNCTIONS

A one way hash function (OWHF) satisfies 1 and 2

A collision resistant hash function (CRHF) satisfies 3 (and hence 2)

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

- Rather than sending Alice a secret across the Internet, just send a hash
- If Alice knows the secret, she can hash it and verify that you know it too
- Safer than sending the secret (which can be intercepted)
- Also more efficient!
- Chance that an attacker can work out the secret from the hash is very low
- Provided the hash function is strong, a longer hash reduces this chance

HASH FUNCTION APPLICATIONS

Digital signatures

Signing message m is slow, but signing h(m) is fast

Much faster to sign a small number than a large file

Useful for an Internet timestamp service

The file itself does not need to be sent, only the hash

Properties 1 + 2 + 3 are required

Property 3 is needed to avoid chosen message attack:

h(m) = h(m')

sign(h(m)) = sign(h(m'))

Password files

e.g. the UNIX password file

Instead of storing passwords in the clear, store the hash

If the password file gets stolen, the hash needs to be inversed before an attacker can use it ("cracking passwords")

HASH FUNCTION APPLICATIONS

Virus protection / Host level intrusion detection

e.g. Tripwire

For each file x, h(x) is stored off system

Periodically hash all files and check the hashes match

Property 2 is critical as it should be hard to find x’ such that h(x) = h(x’)

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 keyspace (dictionaries) at the problem

e.g. cracking UNIX passwords

We can use brute force to attack pre-image resistance:

Say a hash produces a n-bit output $y = h(x)$

We must try 2^{n-1} hashes before $Pr[h(a) = y] \geq 0.5$ $(a \in Z)$
(intuitively: if the secret key is in one of 2^{10}=1024 boxes, you have to open half of them (2^9=512) on average before you find the secret key)

BIRTHDAY ATTACKS ON CRHFS

A **birthday attack** is an attack on collision resistance:

- How many people must be in a room such that any two share a birthday?
  - i.e. Pr[two people have the same birthday] > 0.5?
    (requires only 23 people – check out birthday problem on Wikipedia)
- For an n-bit hash, we must try $2^{n/2}$ hashes of random messages on average before the birthday attack succeeds.

If the hash function output is only 64 bits:

- We can find a collision in 2^{32} tries (trivial!)
- 128 bit hash functions can be broken in a month with US$10M [Wiener/Oorschot]

Strong message digests are usually *at least* 160 bits long

THE CURRENT STATE

Collision resistance

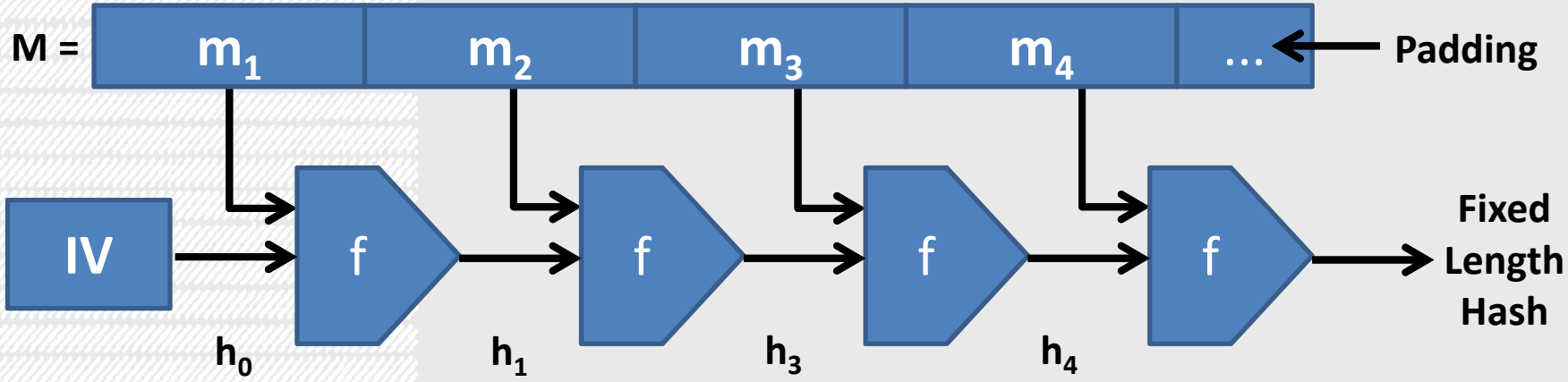
Hash function	Security claim	Best attack	Publish date	Comment
MD5	2^{64}	2^{18} time	2013-03-25	This attack takes seconds on a regular PC. Two-block collisions in 2^{18}, single-block collisions in 2^{41}.[1]
SHA-1	2^{80}	$2^{60.3}$ ... $2^{65.3}$	2012-06-19	Paper.[2] Attack is feasible with large amounts of computation power.[3]
SHA256	2^{128}	31 of 64 rounds ($2^{65.5}$)	2013-05-28	Two-block collision.[4]
SHA512	2^{256}	24 of 80 rounds ($2^{32.5}$)	2008-11-25	Paper.[5]

Preimage resistance

Hash function	Security claim	Best attack	Publish date	Comment
MD5	2^{128}	$2^{123.4}$	2009-04-27	Paper.[7]
SHA-1	2^{160}	45 of 80 rounds	2008-08-17	Paper.[8]
SHA256	2^{256}	43 of 64 rounds ($2^{254.9}$ time, 2^{6} memory)	2009-12-10	Paper.[9]
SHA512	2^{512}	46 of 80 rounds ($2^{511.5}$ time, 2^{6} memory)	2008-11-25	Paper,[10] updated version.[9]

ITERATED HASH CONSTRUCTION

Merkle-Damgard Method (MD-strengthening)



f is a compression function

Divide message M into n x r-bit blocks

f: $\{0,1\}^m$ x $\{0,1\}^r \rightarrow \{0,1\}^m$

Padding Block = | 1 0 0 0 0 0 0 0 ... | Message Length |

WHY USE MERKLE-DAMGARD?

Lemma:

Suppose the compression function **f(m, h)** is collision resistant.

Then the resulting hash function **h** is also collision resistant.

To construct a CRHF it is enough to construct CR compression functions

$$\mathbf{f: \{0,1\}^m \times \{0,1\}^r \rightarrow \{0,1\}^m}$$

SAMPLE OUTPUT

MD5

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

SHA-1

Input	Hash Value (as hex byte strings)
“”	**da39a3ee5e6b4b0d3255bfef95601890afd80709**
“a”	**86f7e437faa5a7fce15d1ddcb9eaeaea377667b8**
“abc”	**a9993e364706816aba3e25717850c26c9cd0d89d**

KEYED HASH FUNCTIONS (MACS)

Well known as **Message Authentication Codes** (MACs)

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 $\mathbf{h_k(m)}$ can be thought of as a cryptographic checksum

Goal:

- 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 MACs (m_i , $h_k(m_i)$) it is hard to construct a valid new pair :

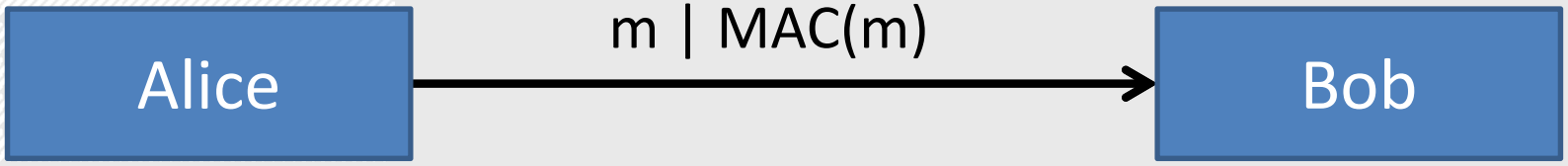
$\mathbf{(m, h_k(m))}$ for $\mathbf{m \neq m_i}$

Formally, a MAC is (ε, t, q, l) - secure if

Given q pairs of each length ≤ l in time t and adversary can succeed in constructing new (message, MAC) pairs with probability < ε

USING MACS – EXAMPLE 1

Network Example:



Alice computes MAC and appends to message

Bob verifies MAC, message is valid only if MAC is valid

Alice and Bob share a secret key **k**

An adversary can't send a message with a valid MAC

MAC(m) = h_k(m)

USING MACS – EXAMPLE 2

Say a hash function is used for virus protection and stores the signatures for each file in a database.

Couldn’t the virus also modify the database?

With a MAC, the virus can’t because it doesn’t know the key.

If it had write permissions it could however corrupt the database or replace the verification program with a trojan / fake.

CONSTRUCTING MACS

Cryptographic

Non-keyed hash functions (HMAC) - fast

Block cyphers (CBC-MAC) - slow

Information Theoretic

Based on universal hashing (outside scope of course)

HASH BASED MAC (HMAC)

MAC based on non-keyed hash function, **h**

Attempt 1 $\mathbf{MAC_k(m) = h(k|m)}$

Insecure: attacker can arbitrarily add to the end of the message **m**!

Attempt 2 $\mathbf{MAC_k(m) = h(m|k)}$

Insecure: vulnerable to the birthday attack!

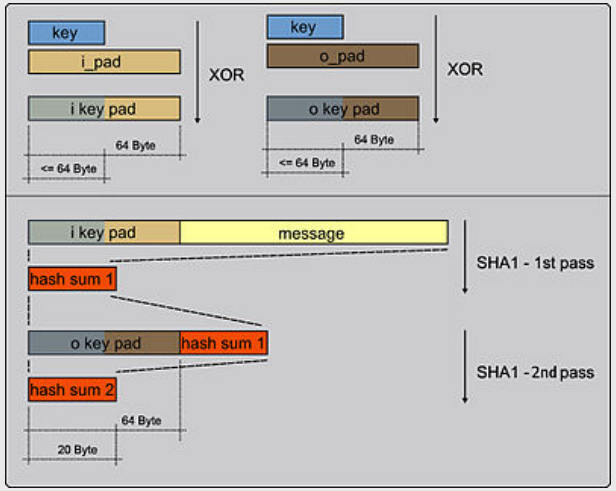
Attempt 3 $\mathbf{MAC_{k,k'}(m) = h(k|m|k')}$

More secure: envelope method

Best $\mathbf{HMAC_k(m) = h((k \oplus opad) \mid h((k \oplus ipad) \mid m))}$

opad is outer padding (0x5c5c5c…5c5c, one block hex constant)
ipad is inner padding (0x363636…3636, one block hex constant)

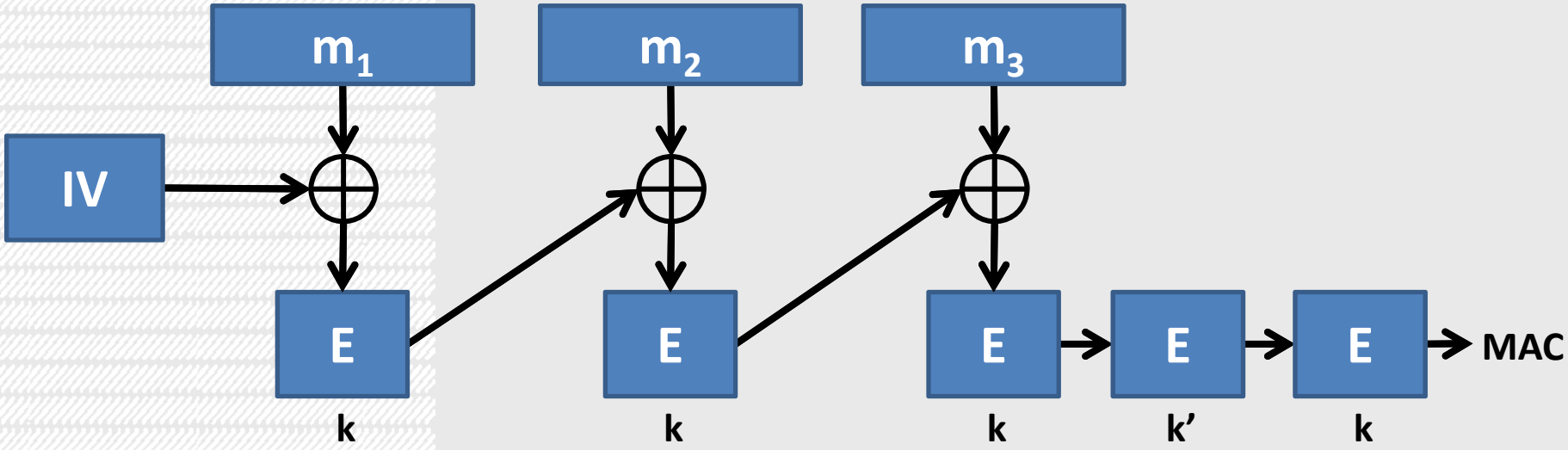
(from RFC 2104)



CYPHER BASED MAC (CBC-MAC)

Uses a technique known as **Cypher Block Chaining** (**CBC**)

- Turn message into blocks
- Repeated encryption using a block cypher is XORd
- Secret key = (k, k', IV)
- IV: Initialisation Vector (random)



If E is a MAC then CBC-E is also a MAC

Often used in the banking industry

CBC-MAC LENGTH

Typical key length is small (e.g. 40 bits)

Security ~2^{40} (easily guessed)

No birthday attack on MACs

Implies MACs are shorter than message digests

CBC-MAC LENGTH

Name	Key Size (bits)	Hash length (bits)	Relative Speed	Class	Notes
Blowfish	**Up to 448**	**64**	**23**	**Block Cypher**	**Bruce Schneier**
DES	**56**	**64**	**10.6**	**Block Cypher**	**Lucifer /NSA**
3DES	**112**	**64**	**3.7**	**Block Cypher**	**Banking**
IDEA	**128**	**64**	**11.8**	**Block Cypher**	**Massey & Lai**
RC5 (r=12)	**Up to 2048**	**32, 64, 128**	**19.6**	**Block Cypher**	**Ron Rivest (RSA)**
AES (r=10, 128 bits)	**128,192,256**	**128,192,256**	**21.1**	**Block Cypher**	**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

KEEP UP TO DATE

Due to a steady stream of breaks against hash functions like MD5, and theoretical attacks on SHA-1, NIST perceived a need for an alternative, dissimilar cryptographic hash, to be called SHA-3.

NIST completed the competition in October 2012.

Numerous entrants were been found to have substantial weaknesses… (and they're written by some of the world's best..!)

The winner was Keccak.

The authors claim 12.5 cycles per byte on an Intel Core 2 CPU. Hardware implementations were "notably faster" than all other finalists.

REFERENCES

Handbook of Applied Cryptography

§1

§9 - §9.4.1

Skim §9.4.2-9.4.3

§9.5 - §9.5.2