Practical Cryptography

Handout 2 – Hash and Message Authentication Code

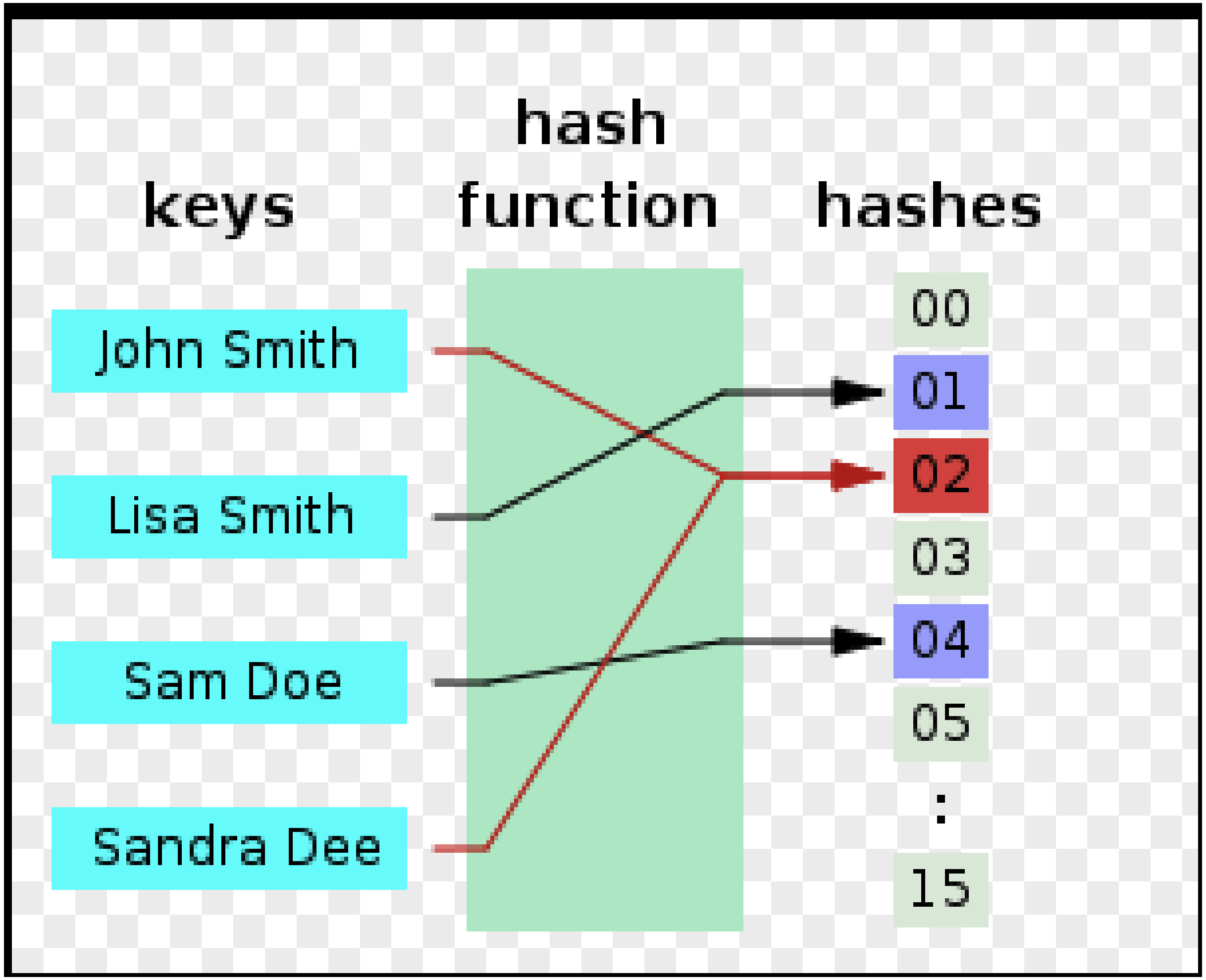
Kasun de Zoysa
kasun@ucsc.cmb.ac.lk

UNIVERSITY OF COLOMBO SCHOOL OF COMPUTING

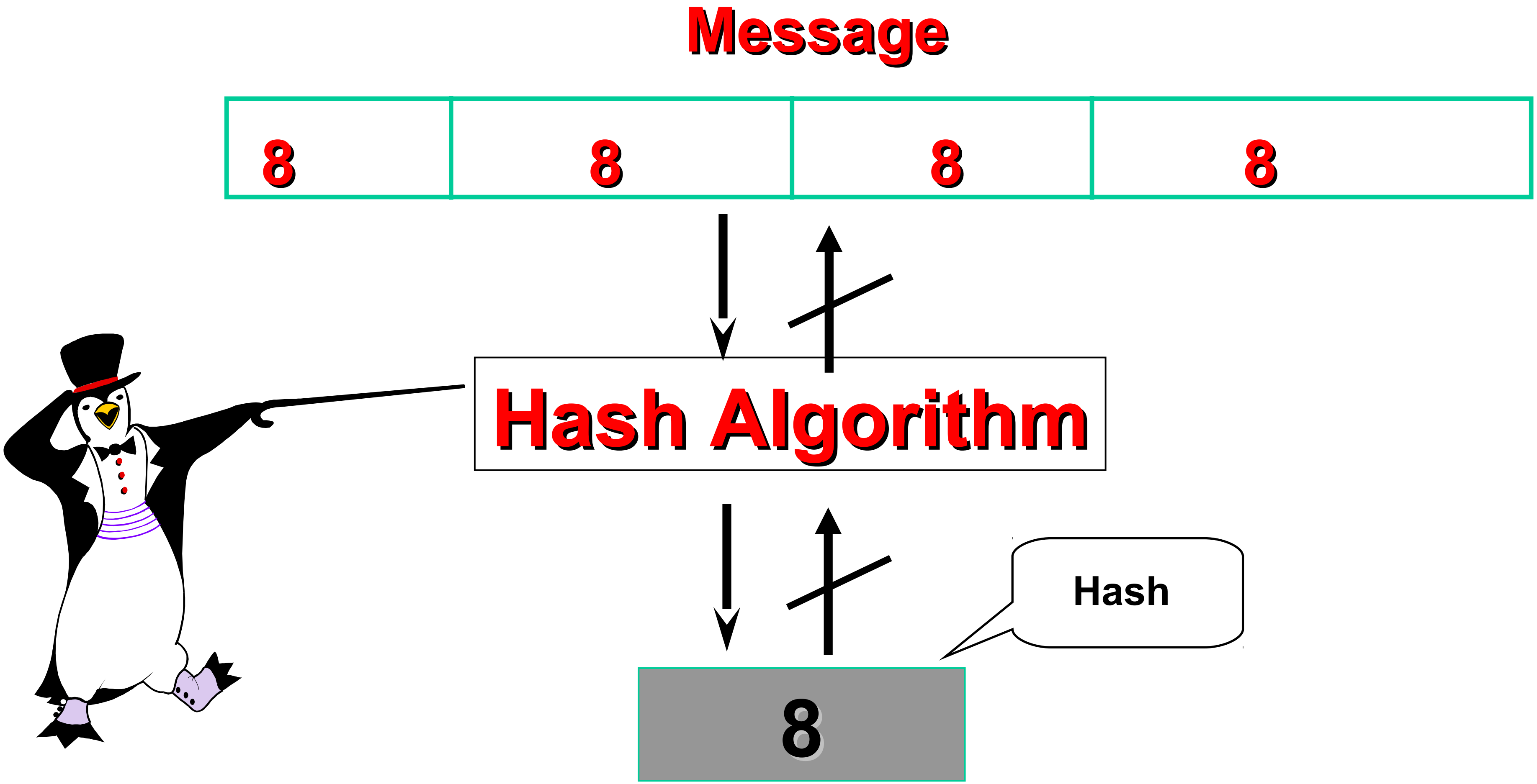
Hash Functions

- Condenses arbitrary message to fixed size
- Usually assume that the hash function is public and not keyed
  - MAC which is keyed (will discuss soon)
- Hash used to detect changes to message
- Can use in various ways with message
  - most often to create a password, digital signature etc.

Hash Function



Hash Functions



Simple Hash Functions

- There are several proposals for simple functions
- Some are based on XOR of message blocks
- Not secure since one can manipulate any message and either not change hash or manipulate the hash as well
- need a stronger cryptographic function

Hash Function Properties

- A Hash Function produces a fingerprint of some file/message/data

$$h = H(M)$$

- condenses a variable-length message M to a fixed-sized fingerprint
- Assumed to be public

Requirements for Hash Functions

- Can be applied to any sized message M
- Produces fixed-length output h
- Easy to compute $h = H(M)$ for any message M
- Given h, it is infeasible to find x s.t. $H(x) = h$

  one-way property
- Given x, it is infeasible to find y s.t. $H(y) = H(x)$

  weak collision resistance
- It is infeasible to find any x,y s.t. $H(y) = H(x)$

  strong collision resistance

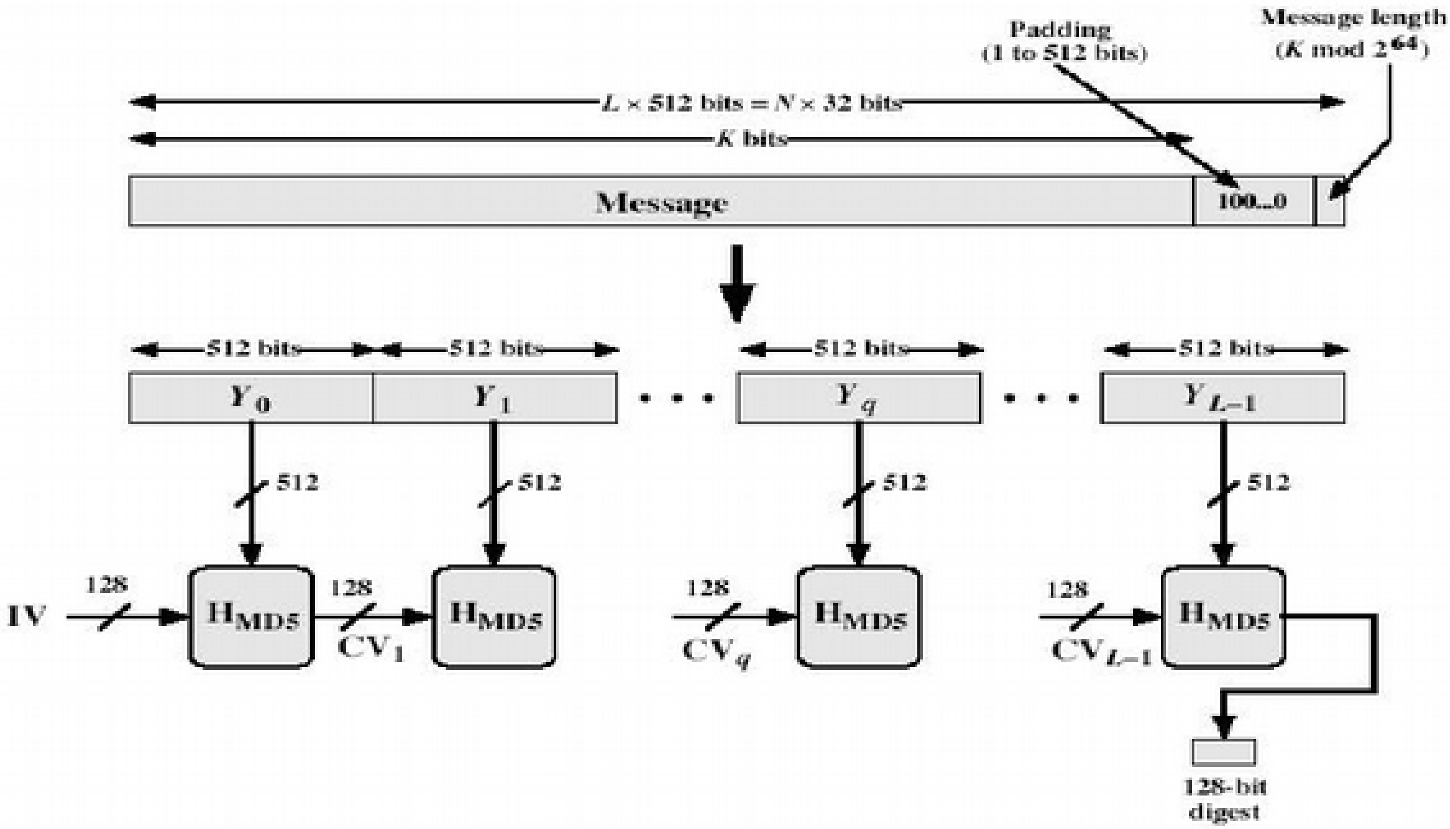
MD5

- Designed by Ronald Rivest (the R in RSA)
- Latest in a series of MD2, MD4
- Produces a 128-bit hash value
- Until recently was the most widely used hash algorithm
  - in recent times have both brute-force & cryptanalytic concerns
- Specified as Internet standard RFC1321

MD5 Overview

1. Pad message so its length is 448 mod 512
2. Append a 64-bit length value to message
3. Initialize 4-word (128-bit) MD buffer (A,B,C,D)
4. Process message in 16-word (512-bit) blocks:
   - using 4 rounds of 16 bit operations on message block & buffer
   - add output to buffer input to form new buffer value
5. Output hash value is the final buffer value

MD5 Overview



Strength of MD5

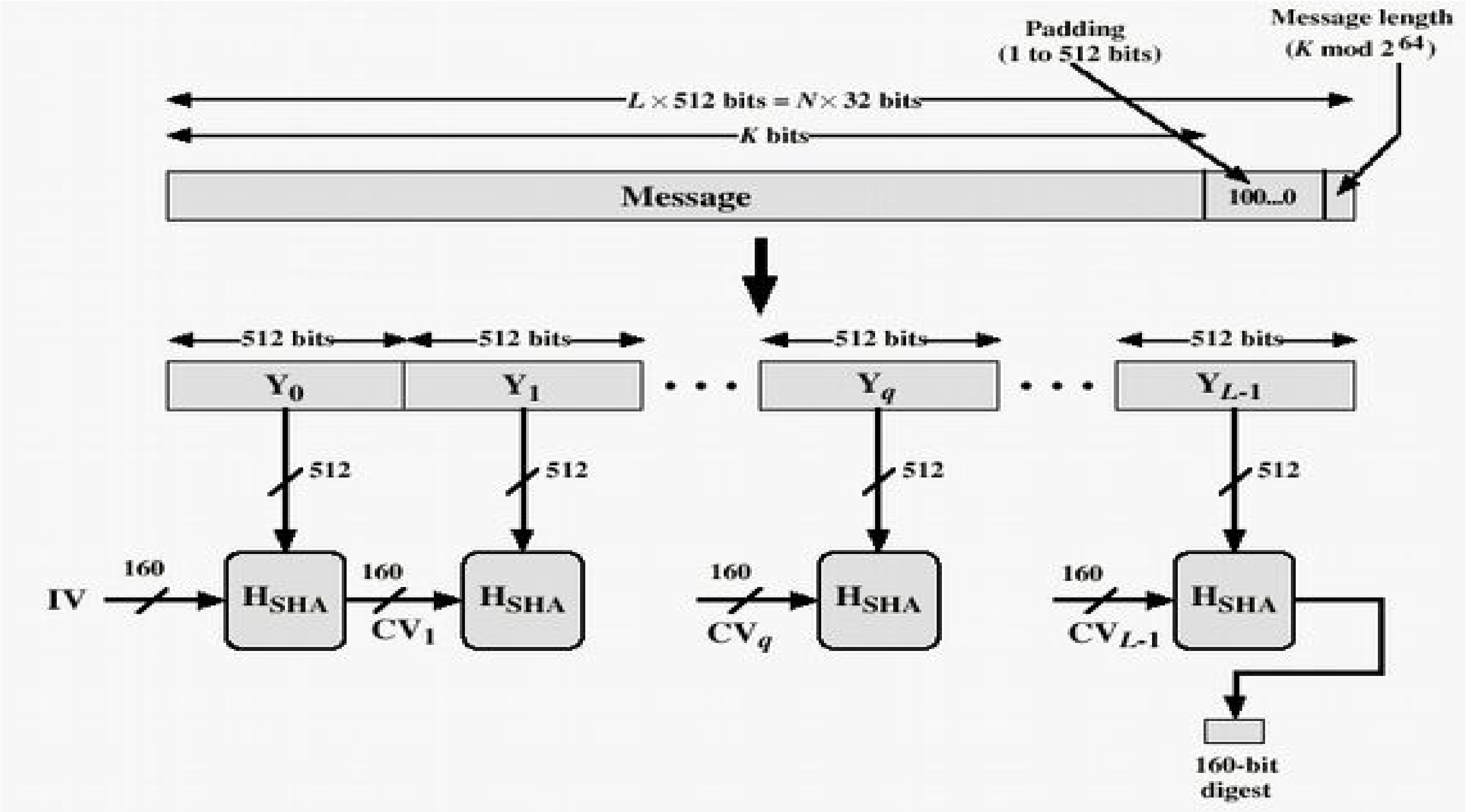
- •MD5 hash is dependent on all message bits
- •Rivest claims security is good as can be
- •Known attacks are:
  - – Berson (92) attacked any 1 round using differential cryptanalysis (but can’t extend)
  - – Boer & Bosselaers (93) found a pseudo collision (again unable to extend)
  - – Dobbertin (96) created collisions on MD compression function (but initial constants prevent exploit)
  - – Crypto 2004 attacks on SHA-0 and MD5
- •Conclusion is that MD5 has been shown to be vulnerable
- •MD5 Collision Demo: http://www.mscs.dal.ca/~selinger/md5collision/

Secure HASH Functions

- Purpose of the HASH function is to produce a ”fingerprint.
- Properties of a HASH function H :

1. H can be applied to a block of data at any size
2. H produces a fixed length output
3. H(x) is easy to compute for any given x.
4. For any given block x, it is computationally infeasible to find x such that H(x) = h
5. For any given block x, it is computationally infeasible to find with H(y) = H(x).
6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y)

Message Digest Generation Using SHA-1



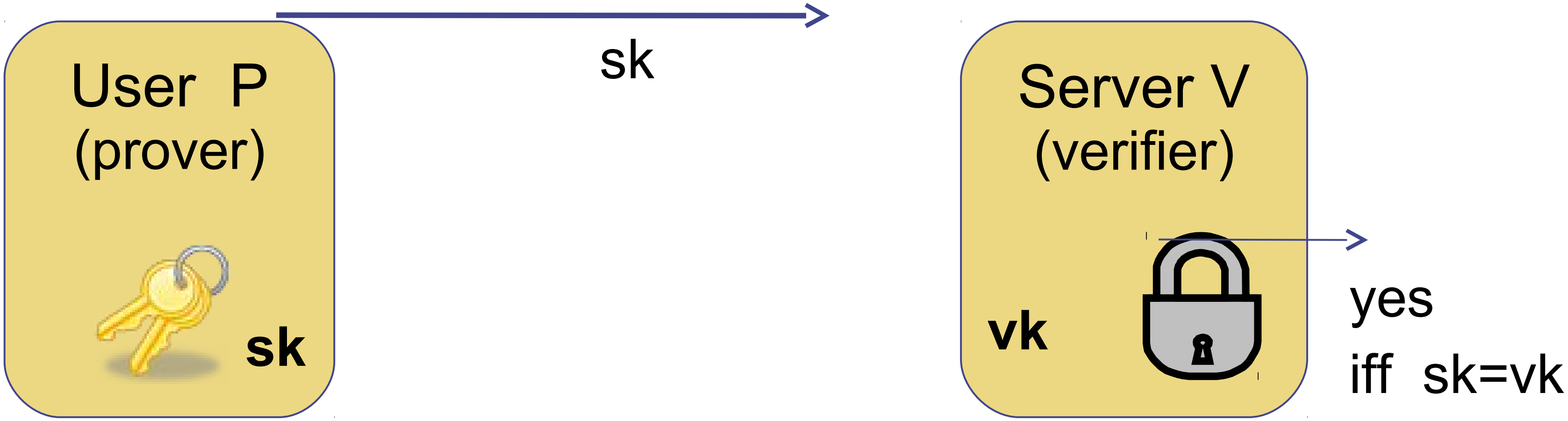
Comparison-Secure HASH functions

	SHA-1	MD5	RIPEMD-160
Digest length	160 bits	128 bits	160 bits
Basic unit of processing	512 bits	512 bits	512 bits
Number of steps	80 (4 rounds of 20)	64 (4 rounds of 16)	160 (5 paired rounds of 16)
Maximum message size	2^{64}-1 bits	∞	∞

Basic Password Protocol (incorrect version)

- **PWD**: finite set of passwords

- Algorithm G (KeyGen):
  - choose rand pw in PWD. output sk = vk = pw.



Basic Password Protocol (incorrect version)

- Problem: VK must be kept secret
  - Compromise of server exposes all passwords
  - Never store passwords in the clear!

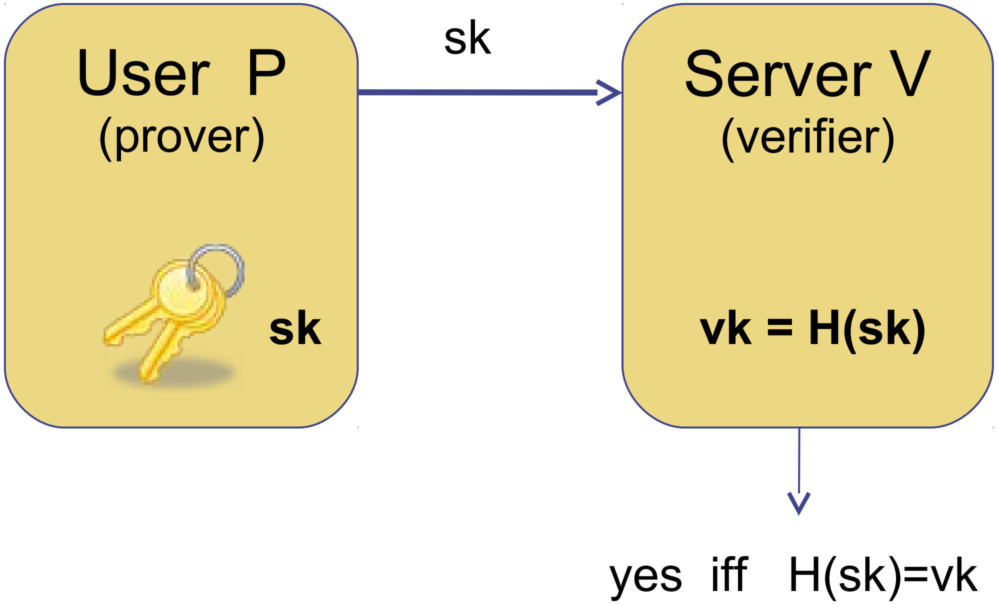
password file on server

Alice	pw_{alice}
Bob	pw_{bob}
…	…

Basic Password Protocol: version 1

H: one-way hash function from PWD to X

"Given H(x) it is difficult to find y such that H(y)=H(x)"



User P

(prover)

sk

sk

Server V

(verifier)

vk = H(sk)

yes iff H(sk)=vk

password file on server

Alice	$H(pw_A)$
Bob	$H(pw_B)$
…	…

Weak Passwords and Dictionary Attacks

- People often choose passwords from a small set:
  - The 6 most common passwords (sample of 32×10^6 pwds):
    123456, 12345, Password, iloveyou, princess, abc123
    ('123456' appeared 0.90% of the time)
  - 23% of users choose passwords in a dictionary of size 360,000,000

- **Online dictionary** attacks:
  - Defeated by doubling response time after every failure
  - Harder to block when attacker commands a bot-net

Preventing Dictionary Attacks

- Public salt:
  - When setting password, pick a random n-bit salt S
  - When verifying pw for A, test if **$H(pw, S_A) = h_A$**

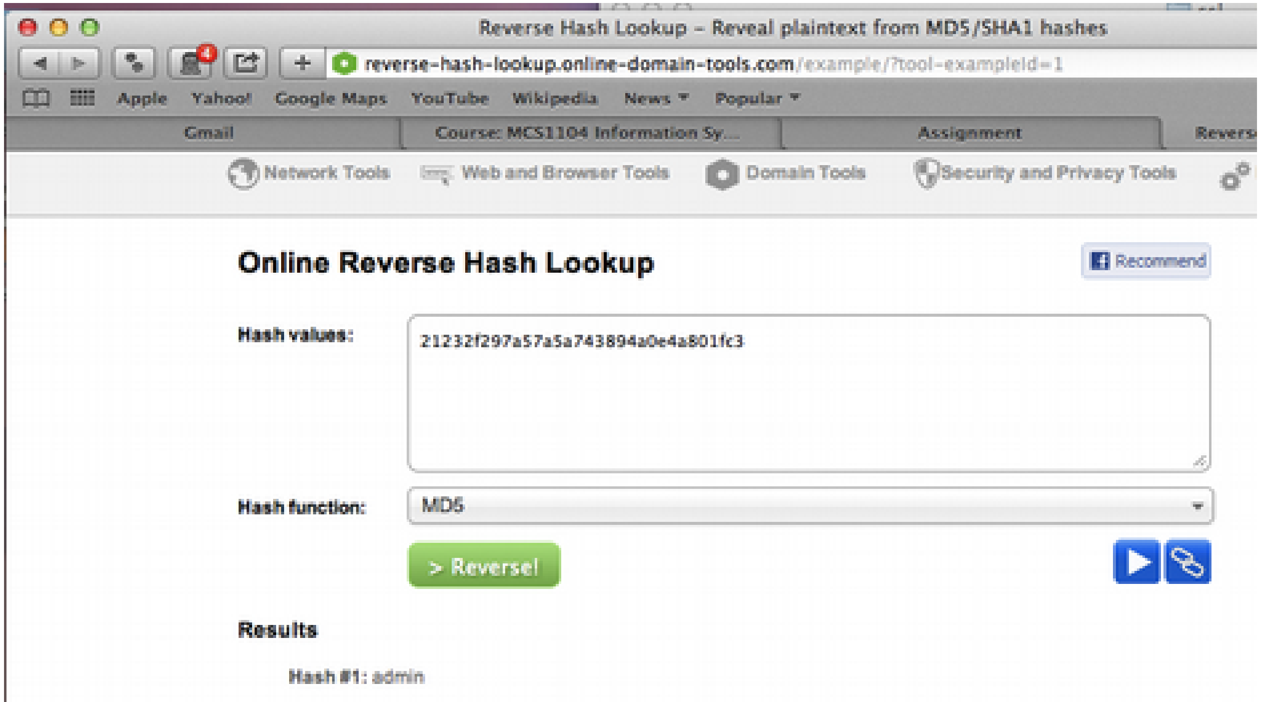
id	S	h
Alice	S_A	$H(pw_A , S_A)$
Bob	S_B	$H(pw_B , S_B)$
…	…	…

- Recommended salt length, n = 64 bits
  - Pre-hashing dictionary does not help

Authenticate the Evidence

- Prove that the evidence is indeed what the criminal left behind.
  - Contrary to what the defense attorney might want the jury to believe, readable text or pictures don't magically appear at random.
  - Calculate a hash value for the data
    - MD5
    - SHA-1,SHA-256,SHA -512

SHA1 Reverse Lookup


Reverse Hash Lookup – Reveal plaintext from MD5/SHA1 hashes
reverse-hash-lookup.online-domain-tools.com/example/?tool-exampleId=1
Apple
Yahoo!
Google Maps
YouTube
Wikipedia
News
Popular
Gmail
Course: MCS1104 Information Sy...
Assignment
Revers
Network Tools
Web and Browser Tools
Domain Tools
Security and Privacy Tools
Recommend
> Reverse!

Online Reverse Hash Lookup

Hash values: 21232f297a57a5a743894a0e4a801fc3

Hash function: MD5

Results

Hash #1: admin

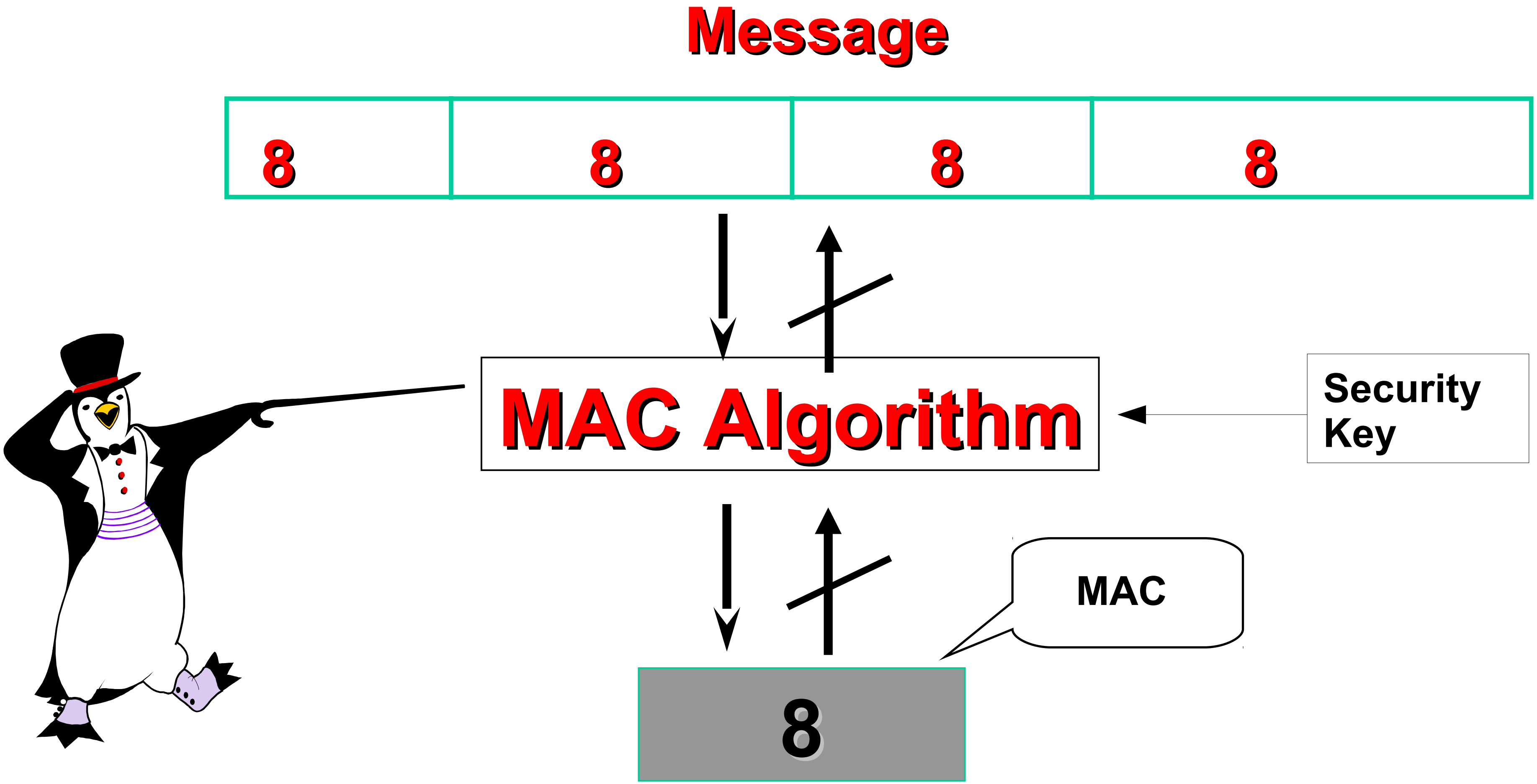
reverse-hash-lookup.online-domain-tools.com

```
echo -n 'kasun'| md5
```

Message Authentication Code (MAC)

- Generated by an algorithm that creates a small fixed-sized block depending on both message and some key
- need not be reversible
- Receiver performs same computation on message and checks if it matches the MAC
- Provides assurance that message is unaltered and comes from sender

Message Authentication Code (MAC)



Message

8 8 8 8

MAC Algorithm

Security Key

MAC

8

MAC Properties

- A MAC is a cryptographic checksum

  MAC = CK(M)

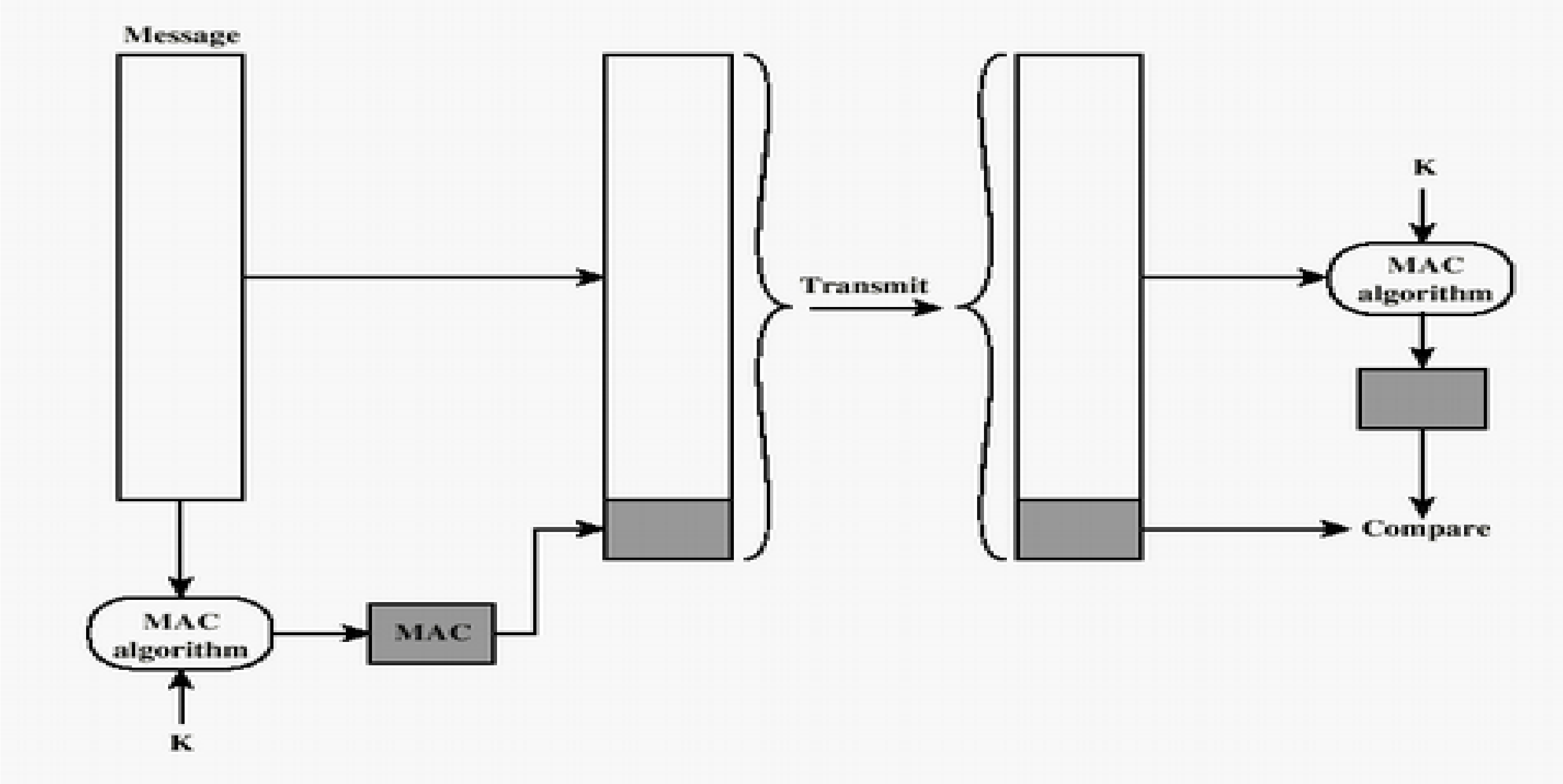
- condenses a variable-length message M
- using a secret key K to a fixed-sized authenticator
- It is a many-to-one function
- potentially many messages have same MAC but finding these needs to be very difficult

Requirements for MACs

- Given a message and a MAC, it should be infeasible to find another message with same MAC
- MACs should be uniformly distributed
- MAC should depend equally on all bits of the message

Approaches to Message Authentication

- Authentication Using Conventional Encryption
  - Only the sender and receiver should share a key
- Message Authentication without Message Encryption
  - An authentication tag is generated and appended to each message
- Message Authentication Code
  - Calculate the MAC as a function of the message and the key. MAC = F(K, M)



Message Authentication Using a Message Authentication Code (MAC)

Keyed Hash Functions (HMAC)

- Create a MAC using a hash function rather than a block cipher
  - because hash functions are generally faster
  - not limited by export controls unlike block ciphers
  - Hash includes a key along with the message
- Original proposal:

  $\mathit{KeyedHash} = \mathit{Hash}(\mathit{Key}|\mathit{Message})$

  - some weaknesses were found with this
- Eventually led to development of HMAC

HMAC Design Criteria

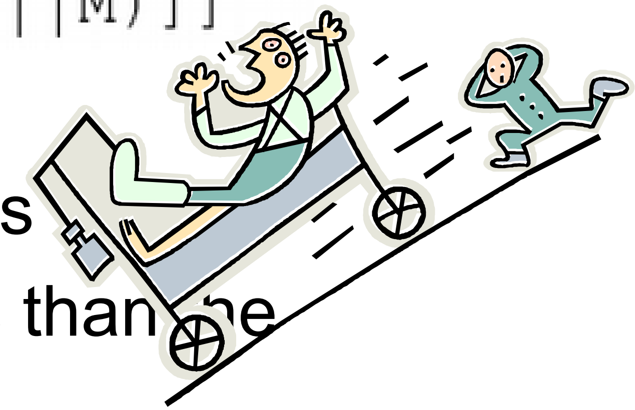
- To use, without modifications, available hash functions.
- To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required.
- To preserve the original performance of the hash function without incurring a significant degradation.
- To use and handle keys in a simple way.
- To have a well-understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions on the embedded hash function.

HMAC

- Specified as Internet standard RFC2104
- Uses hash function on the message:

$$\mathrm{HMAC}_K = \mathrm{Hash}[(K^{+} \text{ XOR } opad) \,||\, \mathrm{Hash}[(K^{+} \text{ XOR } ipad) || M)]]$$

- K+ is the key padded out to size
- opad, ipad are specified padding constants
- Overhead is just 3 more hash calculations than the message needs alone
- Any of MD5, SHA-1, RIPEMD-160 can be used



HMAC Security

- know that the security of HMAC relates to that of the underlying hash algorithm
- attacking HMAC requires either:
  - brute force attack on key used
  - birthday attack (but since keyed would need to observe a very large number of messages)
- choose hash function used based on speed verses security constraints

The birthday paradox

- How many people must there be in a room so that the probability of two of them having the same birthday is larger than 50%?

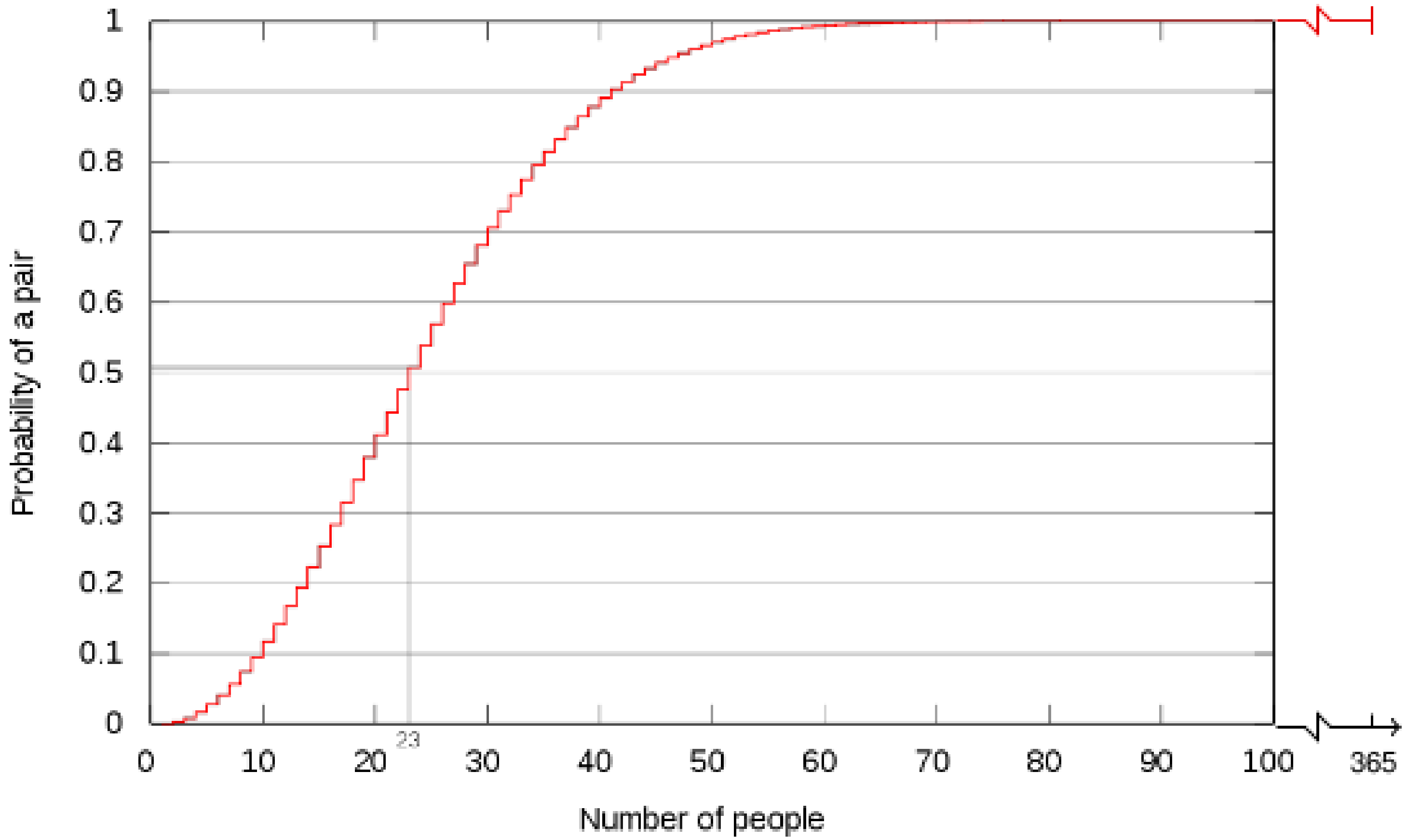
- One tends to be selfish in these cases and think: ”The chance that another person has the same birthday as me is 1/365. The chance that two other person has the same birthday as me is (almost) 2/365.
- So, close to 183.” But this is wrong! This calculation is correct when looking for matches to one specific person.

The birthday paradox

- How many people must there be in a room so that the probability of two of them having the same birthday is larger than 50%?
- The correct calculation is the following: The chance that a second person does not have the same birthday as the first is 364/365.
- If the two first do not have the same birthday, the chance that a third person does not have the same birthday as the two first is 363/365.

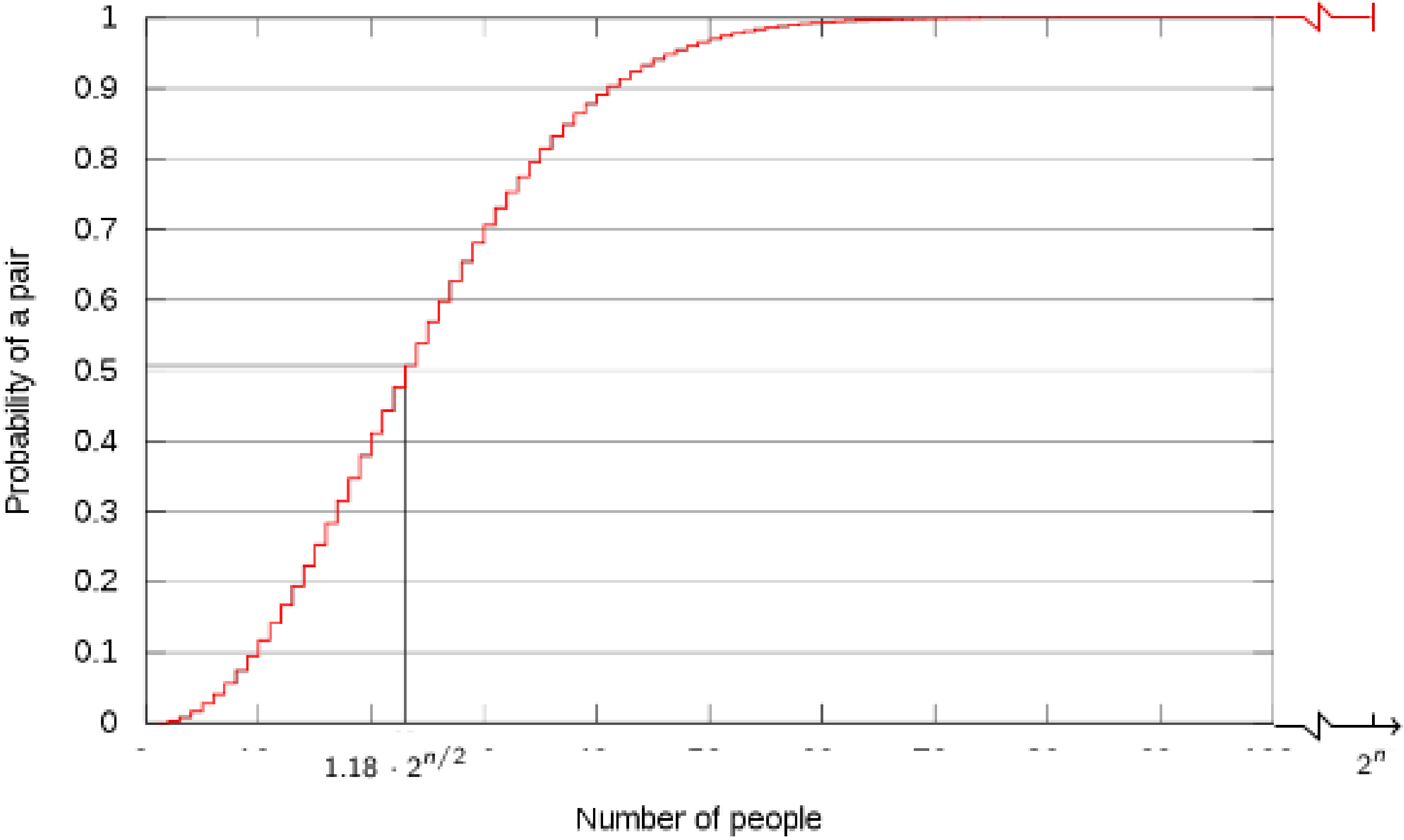
The birthday paradox

How many people must there be in a room so that the probability of two of them having the same birthday is larger than 50%?

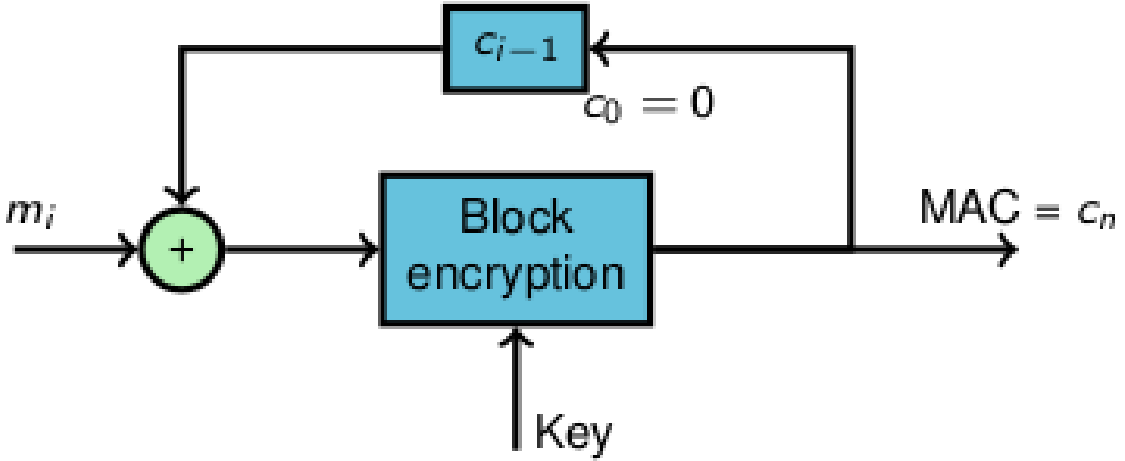


The birthday paradox

How many people must there be in a room so that the probability of two of them having the same birthday is larger than 50%?



Birthday attack on CBC-MAC



- Accumulate many message+MAC pairs, wait for collision
- This will take 2^{64} steps for a 128-bit block cipher because of the birthday paradox (to find a collision, not to match a specific value)

The Java Cryptography Architecture (JCA)

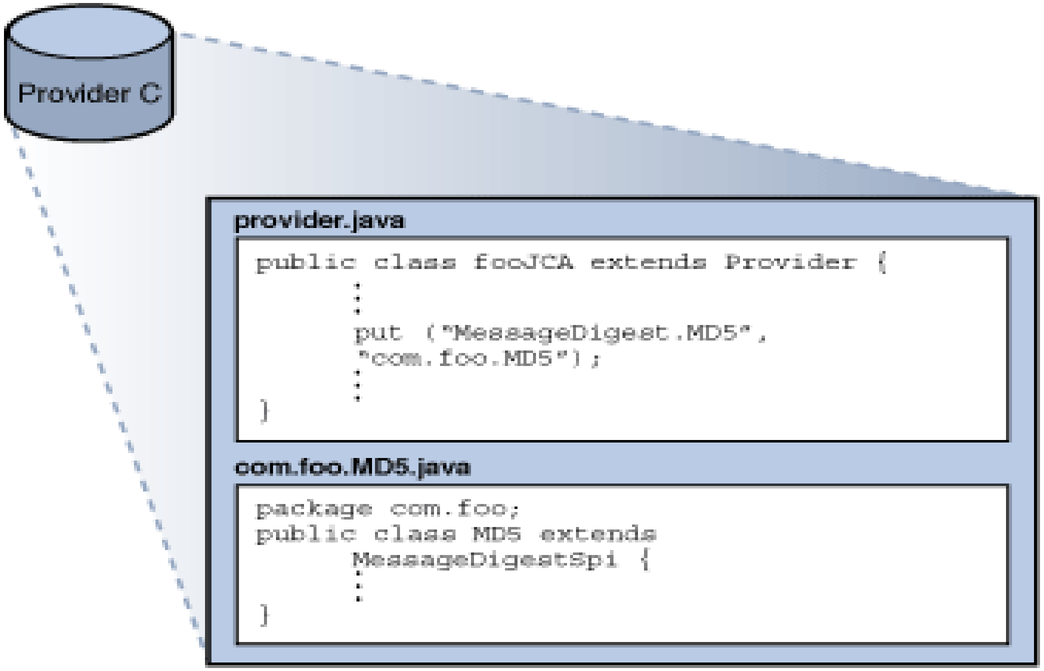
These APIs allow developers to easily integrate security into their application code. The architecture was designed around the following principles:

Implementation independence: Applications do not need to implement security algorithms. Rather, they can request security services from the Java platform.

Implementation interoperability: Providers are interoperable across applications. Specifically, an application is not bound to a specific provider, and a provider is not bound to a specific application.

Algorithm extensibility: The Java platform includes a number of built-in providers that implement a basic set of security services that are widely used today. However, some applications may rely on emerging standards not yet implemented, or on proprietary services.

Provider Class



```
public class fooJCA extends Provider {
          :
          put ("MessageDigest.MD5",
          "com.foo.MD5");
          :
}
```

```
package com.foo;
public class MD5 extends
          MessageDigestSpi {
          :
}
```

There are several types of services that can be implemented by provider packages;

Engine Classes and Algorithms.

Engine Class

SecureRandom: used to generate random or pseudo-random numbers. MessageDigest: used to calculate the message digest (hash) of specified data.

Signature: initilialized with keys, these are used to sign data and verify digital signatures.

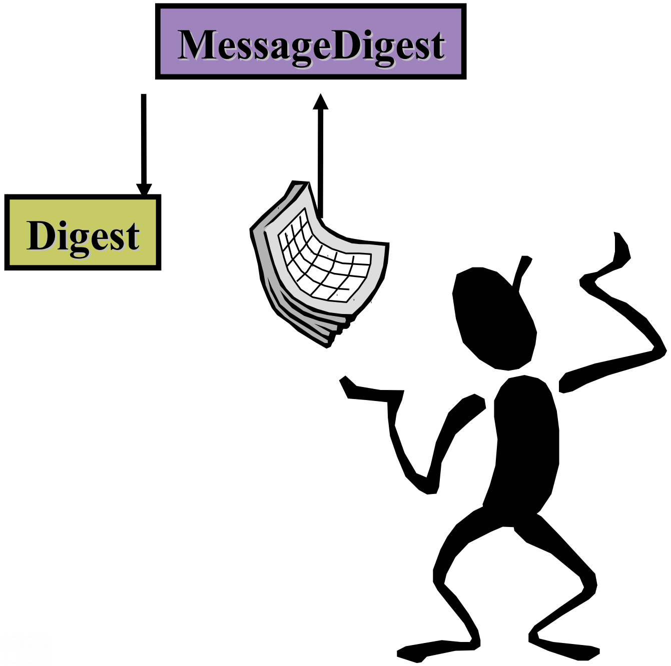
Cipher: initialized with keys, these used for encrypting/decrypting data. There are various types of algorithms: symmetric bulk encryption (e.g. AES, DES, DESede, Blowfish, IDEA), stream encryption (e.g. RC4), asymmetric encryption (e.g. RSA), and password-based encryption (PBE).

Message Authentication Codes (MAC): like MessageDigests, these also generate hash values, but are first initialized with keys to protect the integrity of messages.

Message Digest

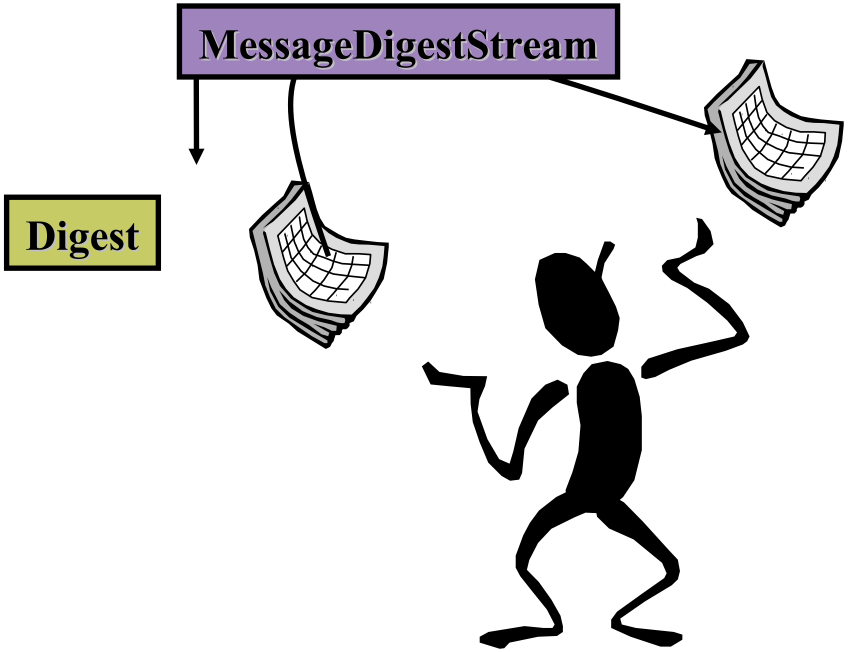
1. **Message Digest Class**

2. **Message Digest Stream Class**



MessageDigest

Digest



MessageDigestStream

Digest

Message Digest Class

Calculate Digest :

1. Get Message Digest Algorithm

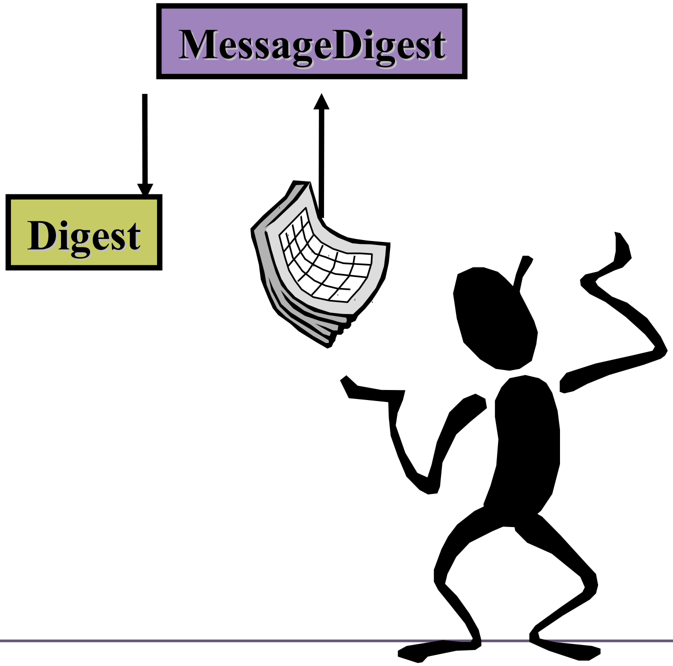
```
MessageDigest md=MessageDigest.getInstance("SHA")
```

2. Feed data into the digest class

```
md.update(buf)
```

3. Calculate the digest

```
md.digest()
```



Message Digest Class

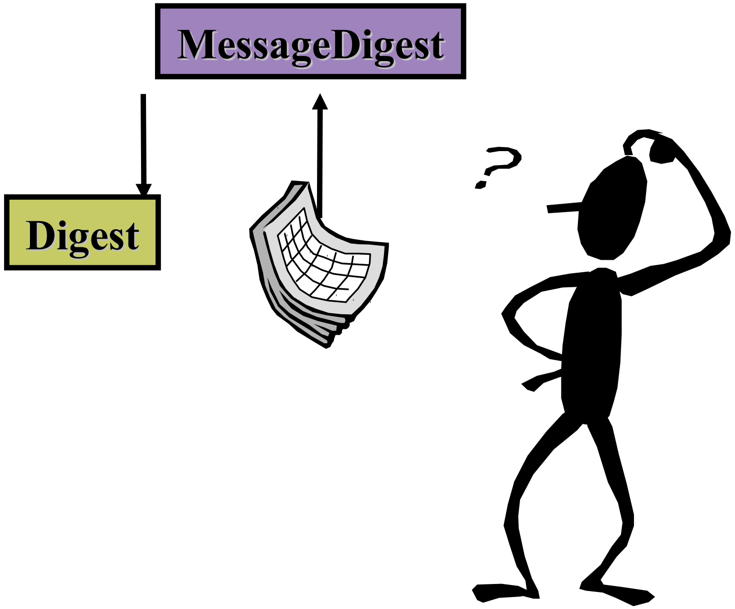
Verify Digest :

1. Get the Message Digest Algorithm
MessageDigest md=MessageDigest.getInstance(“SHA”)

2. Feed data into the digest class
md.update(buf)

3. Calculate the digest
md.digest()

4. Compare with the original digest
md.isEqual(newDigest,originalDigest)


MessageDigest

Digest

UCSC

e-mail: kasun@ucsc.cmb.ac.lk