Security

Cryptography Essentials

Objectives

- Gain understanding of three main ingredients of most security protocols & products
  - **Symmetric encryption**
  - **Public-key cryptography**
  - **Cryptographic hash functions**

- Learn about (public) key management using
  - **Digital certificates**

Introduction

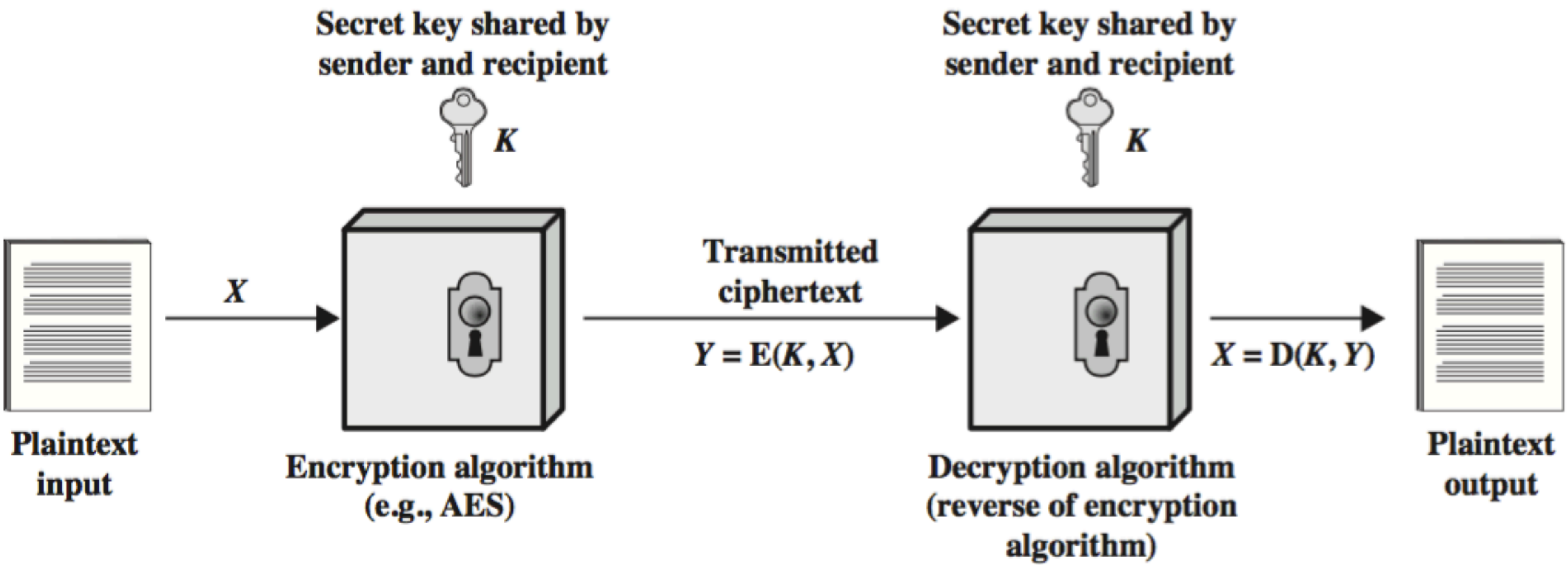
Some jargon

Cryptography:	Science of “secret writing”
Plaintext:	Original message
Ciphertext:	Transformed message
Encryption:	plaintext -> ciphertext process
Decryption:	ciphertext -> plaintext process
Cipher:	“Secret method of writing” (i.e. algorithm)
Key:	Some critical information used by the cipher, known only to sender and/or receiver
Cryptanalysis:	Attempting to discover plaintext or key or both

Symmetric Encryption

- Sender and receiver use <u>same</u> key (shared secret)
- Was the only method used prior to the 1970s & still the main "workhorse"
- Popular algorithms:
  - Advanced Encryption Standard (AES)
  - Triple Data Encryption Standard (3DES)
  - Rivest Cipher 4 (RC4) – *until recently!*
- Fast
- But how to share secret keys?
  - "chicken-and-egg" problem

Symmetric Encryption



Public-key Cryptography

- Major limitations of Symmetric Encryption:
  - Key distribution problem
  - Not suitable for authentication: receiver can forge message & claim it came from sender

- Addressed by Public-key Cryptography

- Public-key methods based on sender and receiver using <u>different</u> keys

Public-key Cryptography

- Each party has two keys:
  - a **public key**, known potentially to anybody, used to **encrypt messages**, and **verify signatures**
  - a **private key**, known only to its owner, used to **decrypt messages**, and **create signatures**

- Complements rather than replaces symmetric cryptography
  - Used for exchanging secret keys

Applications of Public-key Cryptography

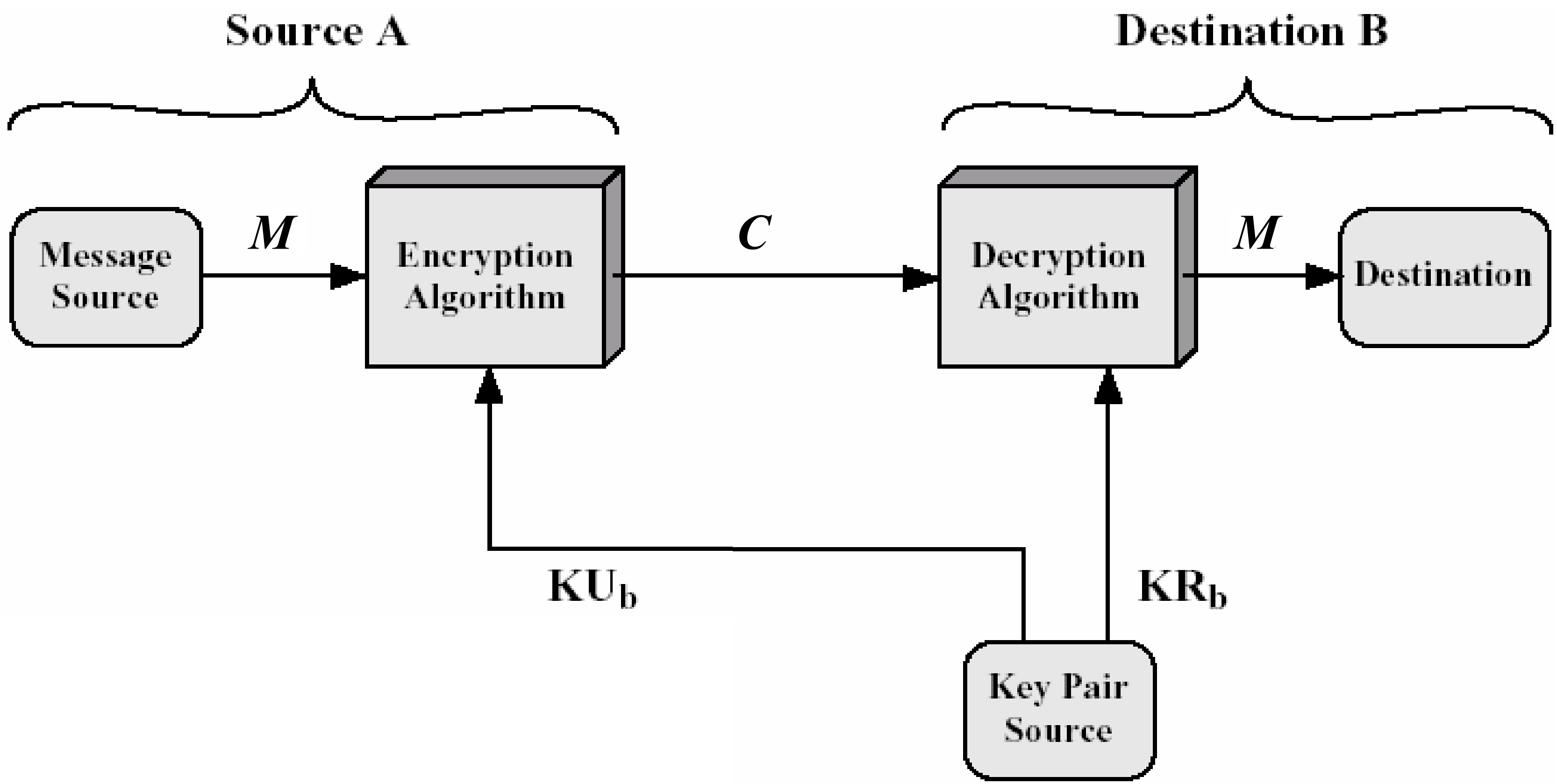
- Can classify uses of public-key cryptography into 3 categories:
  1) **encryption/decryption** (provides secrecy)
  2) **digital signatures** (provides authentication)
  3) **key exchange** for symmetric encryption
     - which is a special case of (1)

- Some public-key algorithms are suitable for all uses; others are specific to one of the above

Application: Secrecy

- Alice (A) sends message to Bob (B) by encrypting with his public key
- Message can only be decrypted with Bob's corresponding private key (known only to him)

Secrecy Model



KU_b B’s pUblic key

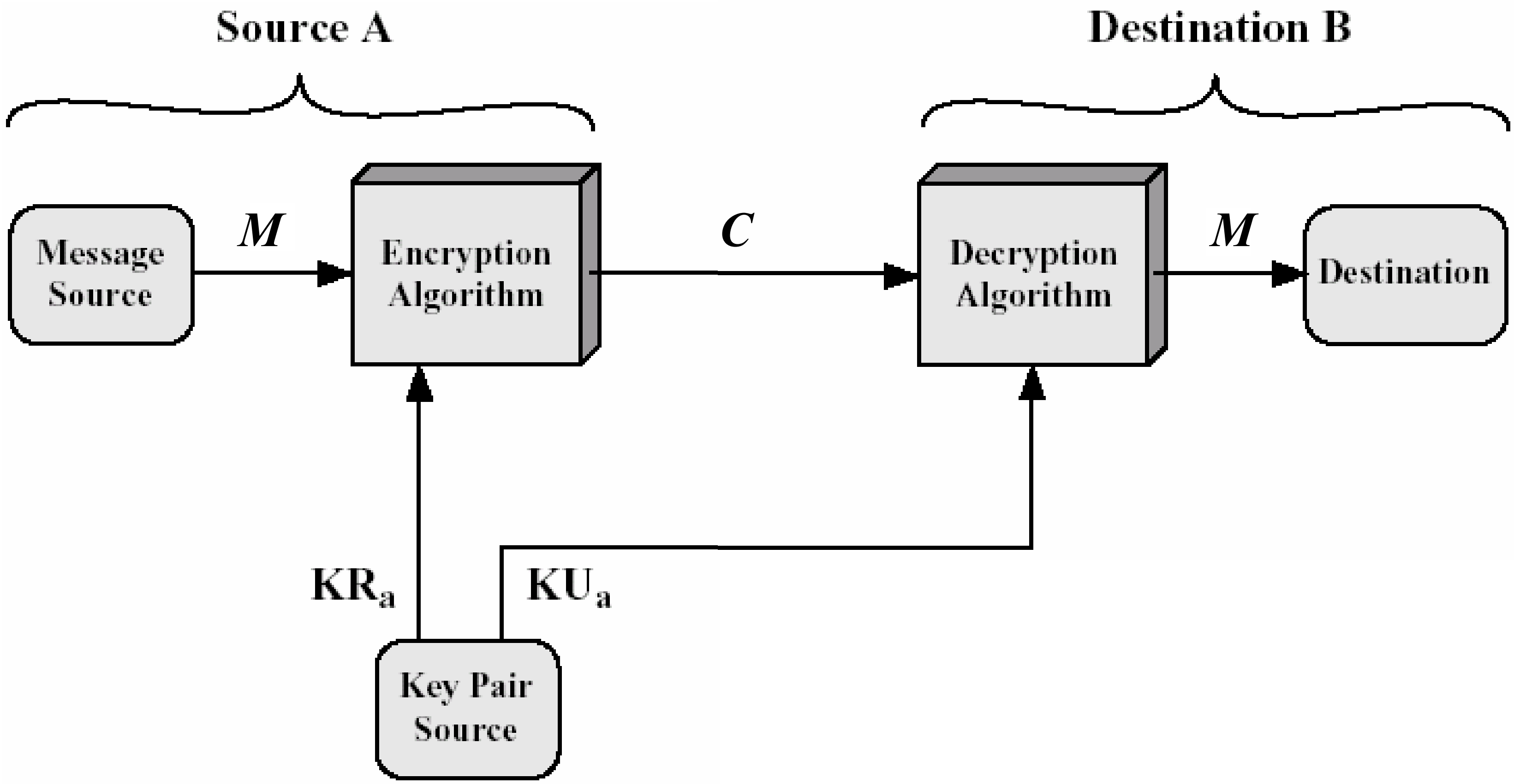
KR_b B’s pRivate key

Application: Authentication

- Alice (A) sends message to Bob (B) encrypting it with <u>her</u> own <u>private</u> key (i.e. she signs the message)

- Everyone with Alice's <u>public</u> key can decrypt the message. A message that can be decrypted with Alice's public key ***must have come from Alice***.

Authentication Model



KR_a A's pRivate key

KU_a A's pUblic key

Limitations of Public-key Cryptography

1. Processing speed
   – Calculations required for public-key algorithms (mainly multiplications) much slower than those of conventional algorithms (permutations & XORs)
   – Thus public-key methods not suitable for general-purpose encryption/decryption
   – Instead often just use public-key method to exchange session (secret) key at beginning of session & use session key thereafter

Limitations of Public-key Cryptography

2. Authenticity of public keys (MITM attack)
   - Bob's public key is in the public domain and only Bob has the corresponding private key
   - What happens though if an eavesdropper (Eve) generates another key pair and advertises the public key produced as belonging to Bob?
   - People then may send messages to Bob using the wrong public key, for which Eve has the corresponding private key.

$\Rightarrow$ *Need to be able to* ***trust*** *that a public key belongs to whom it is reputed to belong.*

Cryptographic strength & cryptanalysis

Kerckhoff’s principle

- Security should depend on the secrecy of the **key**, not the secrecy of the algorithm

- Attempts to keep algorithms secret are usually ineffective (they leak out)
- … and counterproductive as review by the wider crypto community allows weaknesses to be found early on, before deployment.

Cryptanalysis

- Cryptanalysis is the process of trying to find the plaintext or key
- Two main approaches
  - Brute Force
    - try all possible keys
  - Exploit weaknesses in the algorithm or key
    - e.g. key generated from password entered by user, where user can enter bad password

Cryptanalysis: Brute Force Attack

- Try all possible keys until code is broken
- On average, need to try half of all possible keys
- Infeasible if key length is sufficiently long

Key size (bits)	No. of keys	Time required at 1 encryption per *μs*	Time required at 10^6 encryptions per *μs*
32	4.3×10^9	36 minutes	2 milliseconds
56	7.2×10^{16}	1142 years	10 hours
128	3.4×10^{38}	5.4×10^{24} years	5.4×10^{18} years
168	3.7×10^{50}	5.9×10^{36} years	5.9×10^{30} years

Age of universe: ~ 10^{10} years

Note: DES has a 56 bit key; AES key has 128+ bits

Symmetric Block Ciphers

XOR

- Modern techniques use bits rather than text letters
- Most transformations use eXclusive OR
- **Revsersibility** and **speed** are the main benefits of using XOR

XOR truth table:

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

XOR properties:

$A \oplus A = 0$

$A \oplus 0 = A$

$(A \oplus B) \oplus B = A$

Block Cipher

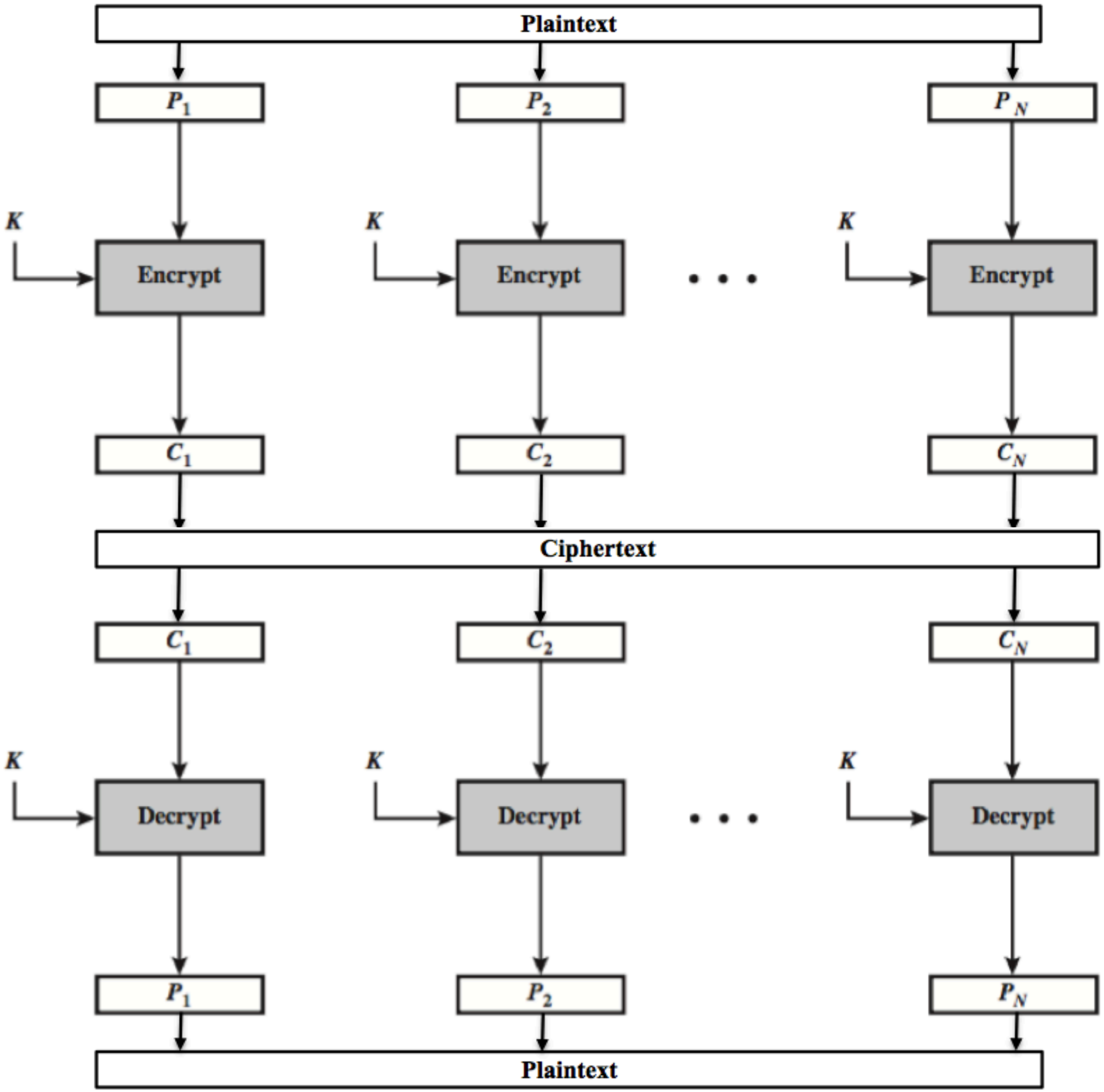
- A block cipher divides the plaintext into fixed-sixed blocks and transforms each block into a corresponding block of ciphertext
- Padding is required where the plaintext size is not an integer multiple of the block size
- Iterated block ciphers are based on a number of rounds where a round function is applied at each round.
- The round function usually takes a round key as one of its inputs.
  - Each round key based on bits extracted from the key

Block Cipher – modes of operation

- Electronic Codebook (ECB) mode
  - Each block treated independently.
  - Insecure, as repeated plaintext blocks map to repeated ciphertext blocks
- Cipher Block Chaining (CBC) mode
  - Each plaintext block XORed with previous ciphertext block before encryption
- Counter (CTR) mode
  - For each plaintext block encrypt a counter and XOR the result with the plaintext block. Increment the counter for the next block

Electronic Codebook Mode (ECB) Encryption

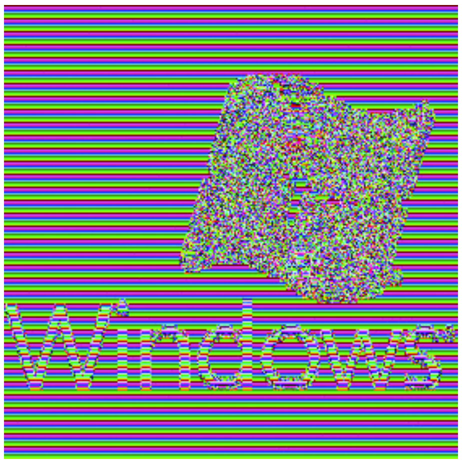
ECB Decryption



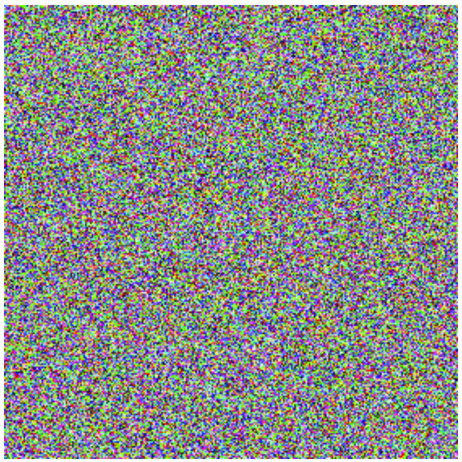
Comparing CBC with ECB

- Codebooks are a problem as patterns in the plaintext may remain in the ciphertext

Plaintext



Ciphertext (ECB)



Ciphertext (CBC)

Source: msdn.microsoft.com

DES

- Data Encryption Standard (1976)
- Block size: 64 bits
- Key size: 56 bits
- No. of rounds: 16
- Based on design by Horst Feistel, IBM
  - Chosen by NBS (now called NIST), US national standards body
  - Influenced by NSA
- Very influential algorithm
- Now obsolete, but lives on in Triple DES (3DES)

AES

- Advanced Encryption Standard (2001)
- Chosen by design competition
  - Organised by NIST (US National Standards Inst.)
  - Winner: Rijndael (Belgium)
- Block size: 128 bits
- Key sizes: 128, 192, 256
- Relatively small memory requirement
- Suitable for variety of hardware and software architectures
- Royalty-free
- Considered secure
- <u>Very</u> widely used

AES

- You can find a nice AES animation here:

  http://www.securityfit.cz/download/kib/rijndael_ingles2004.swf

  or **http://tinyurl.com/aesflash**

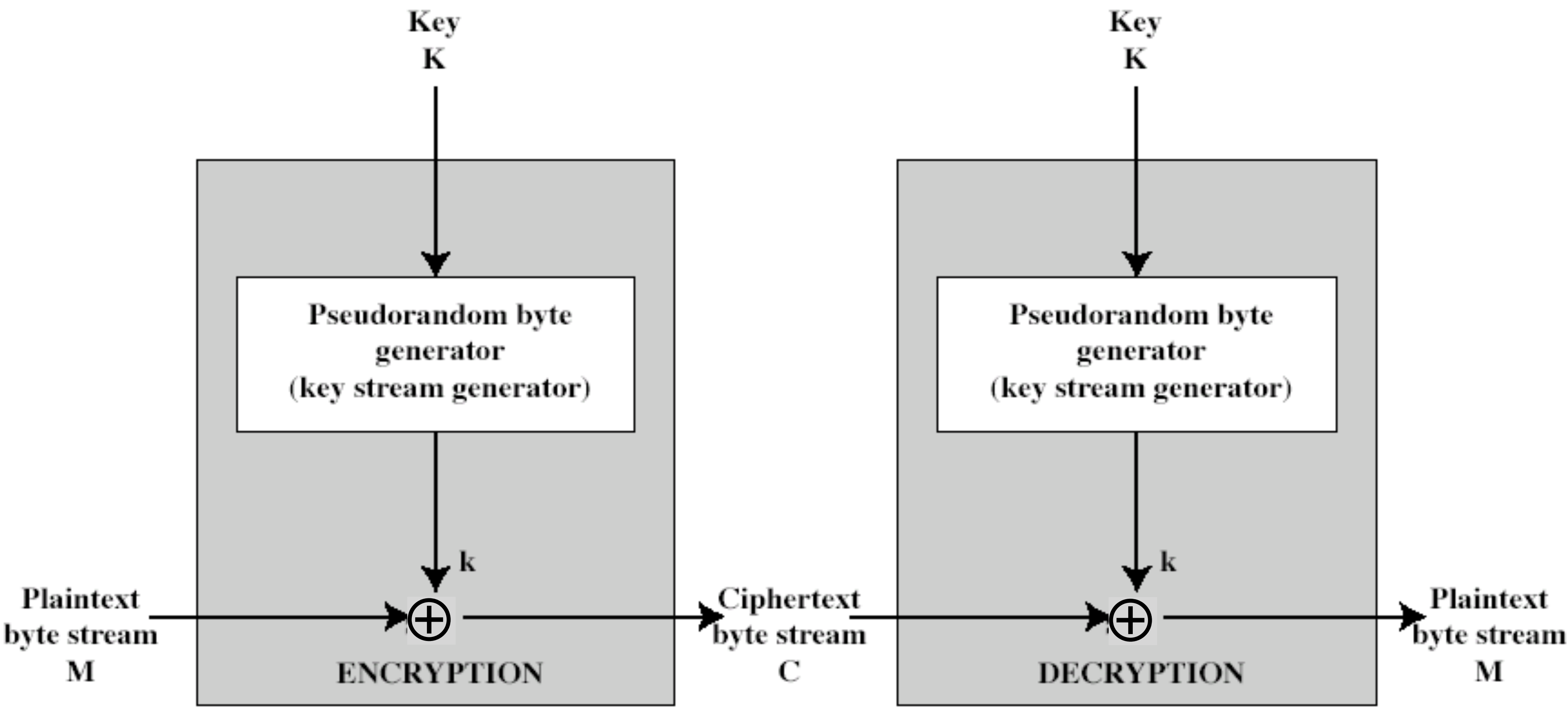
Stream Ciphers

Stream Ciphers

- Process message “continuously”
  - Optimised for real-time and two-way comms
  - Usually one byte at a time
  - As distinct from a block cipher

- Typically simple XOR of each plaintext bit with the output of a pseudo-random number generator (PRNG)

Stream Cipher Structure



$C_i = M_i \oplus k_i$

$M_i = C_i \oplus k_i$

Danger with Stream Cipher

- If plaintext-ciphertext pairs can be gathered, then it is easy to record the keystream:
  - as $M_i \oplus C_i = k_i$
- Thus the cipher is broken if any way to predict key stream for next ciphertext
- Key streams should never be re-used (or re-started with the same seed)

Public-key Algorithms

Trapdoor functions

- Public-key cryptography relies on functions that are computationally easy in one direction and computationally infeasible in the other
- Examples:

"Easy" problem	"Hard" problem	Technique
Multiplying prime numbers, $n = pq$	Factoring n	RSA
Modular exponentiation, $g^x \pmod{n}$	Calculating discrete log; solving for x in $a = g^x \pmod{n}$	Diffie-Hellmann
Elliptic curve point multiplication, $R = kP$	Finding elliptic curve multiplicand, k	Elliptic curve cryptography

RSA

- Rivest, Shamir & Adleman, MIT, 1977
- Very well known versatile public-key scheme
- Uses large integers as keys (>1000 bits)
- Security due to extreme difficulty of factoring large "semiprime" integers
  - i.e. factoring product of two prime numbers

RSA

- Based on three related integers: e, d, n
- RSA function ("encryption"):
  - Input: $M < n$
  - Output: $\boxed{C = M^e \pmod{n}}$
- Inverse RSA ("decryption"):
  - Input: C
  - Output: $\boxed{M = C^d \pmod{n}}$

d and e are mathematically related: e is chosen and d is calculated from e and the **factors** of n

Diffie-Hellman

- Public-key Technique for exchanging secret keys
  - First public-key technique (1976)
- The secret key is calculated by both parties
- Requires some global public parameters
- Based on difficulty in solving for x:

$$\boxed{a = g^x \pmod{n}}$$

a, g, n known

Elliptic Curve Cryptography

- Majority of public-key crypto (RSA, D-H) use either integer or polynomial arithmetic with very large numbers/polynomials
- Imposes a significant load in storing and processing keys and messages
- An alternative is to use elliptic curves
- Offers same security as RSA with smaller bit sizes and lower processing and memory overhead
- Recent growth in use

Integrity and Authentication

Data Integrity

- Integrity refers to assurance of non-alteration
- Many systems and components have checksums or cyclic redundancy checks that are designed to detect *accidental* errors, etc.
  - For example, a credit card number contains a digit that is used to verify the others
- But such schemes are not sufficient to prevent *deliberate* modifications

Cryptographic Hash Functions

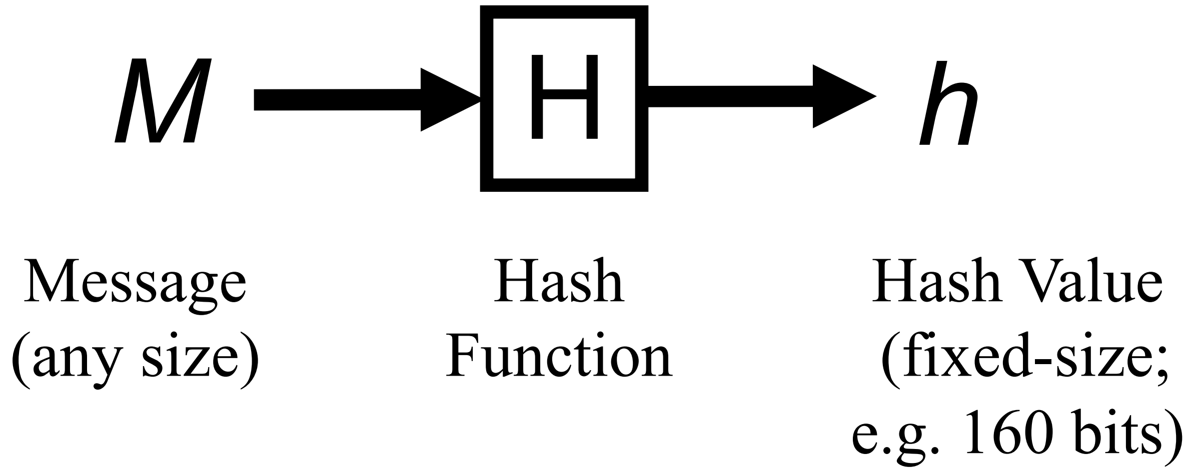
- Used to provide integrity of a message
- Purpose is to produce a fixed-size *hash-value:*

$$h = H(M)$$

where h is the hash value
H is the hash function
M is the message

- Any change in M, however small, should produce a different h-value

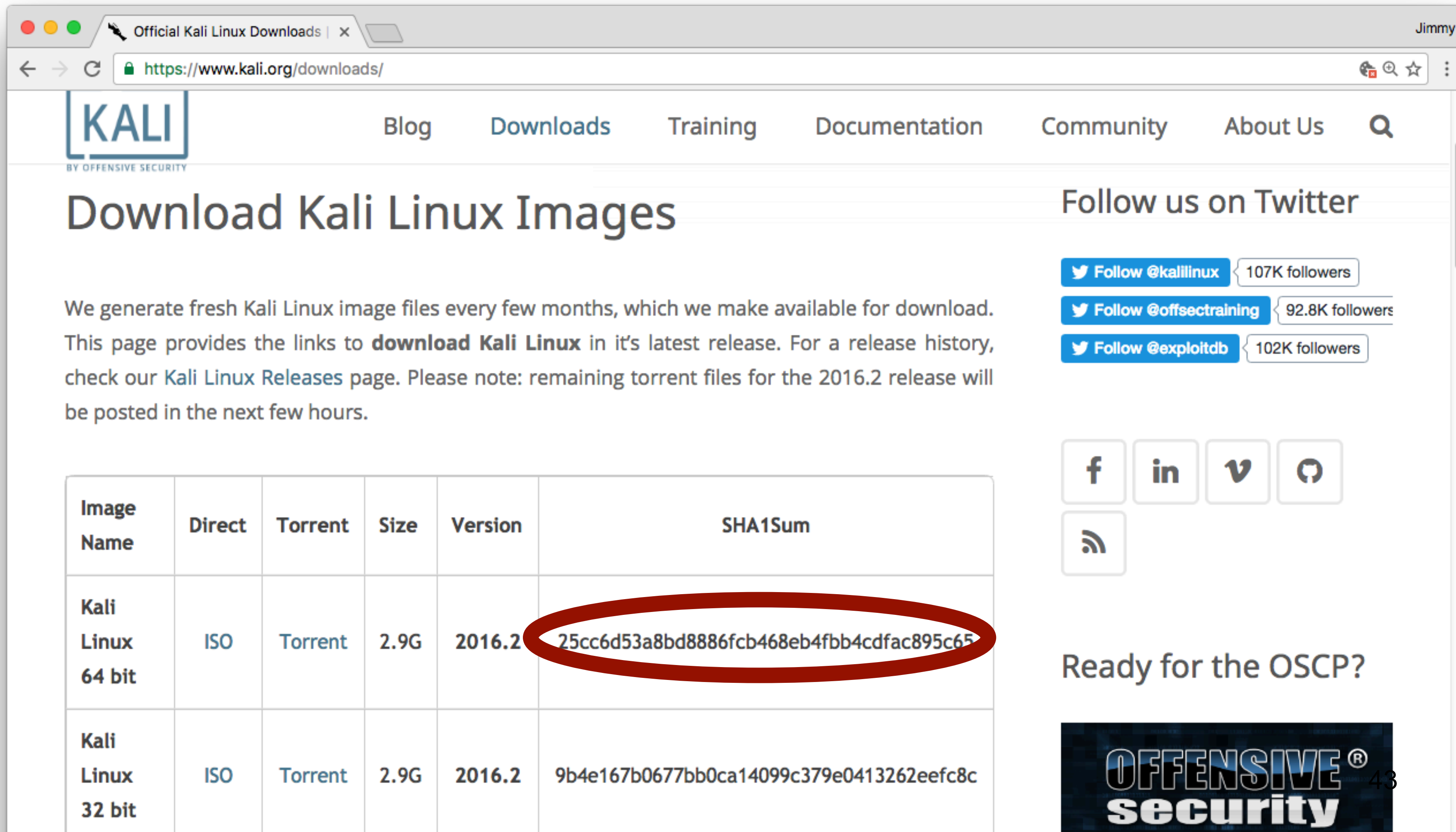
Cryptographic Hash Functions



- Note that a hash function is a many-to-one function. Potentially many messages can have the same hash, but finding these should be very difficult

Applications of Hash Functions

- As cryptographic checksum
  – e.g. to verify software downloads

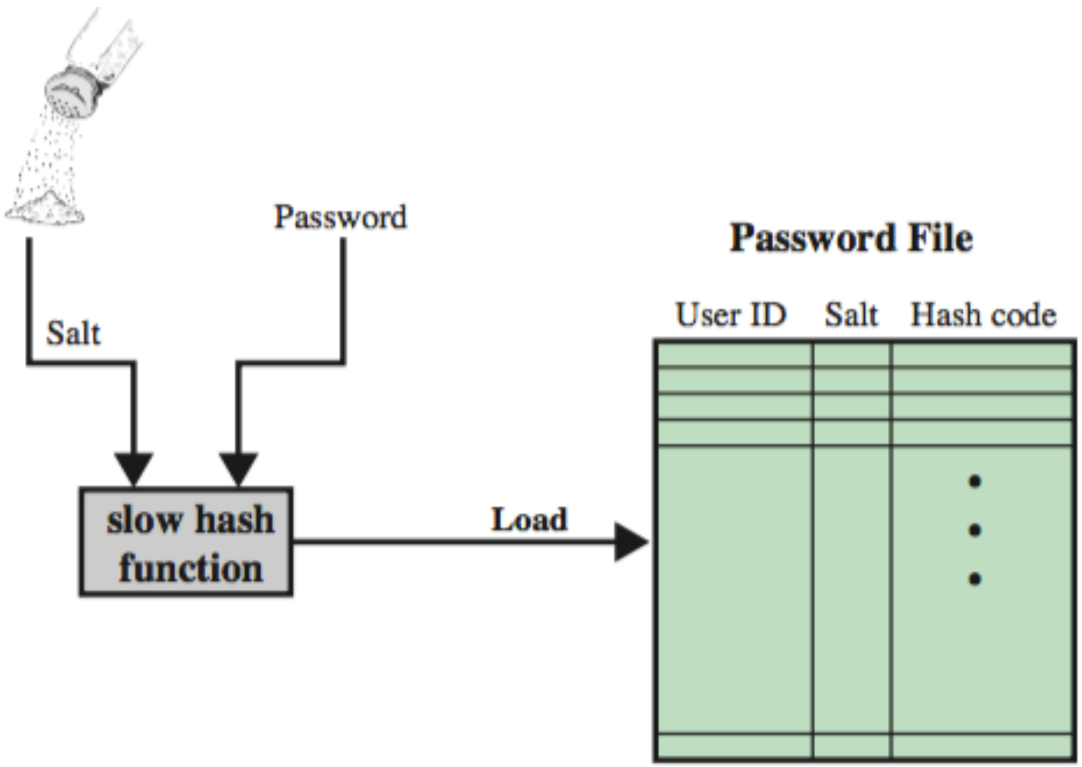


Applications of Hash Functions

- Authentication
  - It usually makes more sense to sign the hash of a message (with a private key) than to sign the original message
  - This is done with digital certificates and many other authentication schemes

Applications of Hash Functions

- Password storage
  - Store only the hash of password (+ *salt*)
  - e.g. Unix password scheme



Cryptanalysis: Breaking hash functions

- Strength depends on the length, n, in bits of the hash value

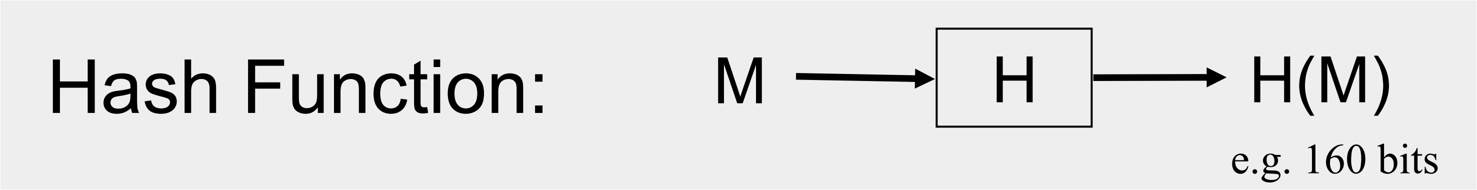
- Brute force attacks require time proportional to:
  - one-way property: 2^n
  - weak collisions property: 2^n
  - strong collisions property: $2^{n/2}$
    - This means the ability to find ***any*** two messages that hash to the same value:

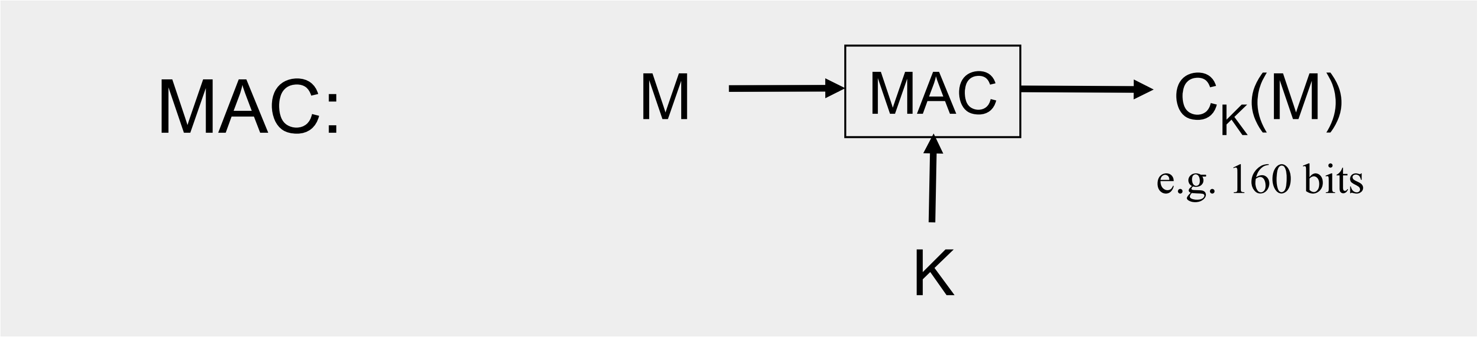
Main Hash Algorithms

- MD5
  - Produces 128-bit hash value (i.e. 64-bit security)
  - Collisions found (2004)
  - No longer recommended for use
- SHA-1
  - Produces 160-bit hash value (80-bit security)
  - Collisions found (2017)
  - No longer recommended for use
- SHA-2
  - Set of 4 hash functions with different size outputs
  - SHA-224, SHA-256, SHA-384, SHA-512
  - Considered safe to use
    - (though new SHA-3 has been established due to concerns over structural similarities with SHA-1)

Message Authentication Code (MAC)

- Very similar to Hash Function
- Difference is the use of a **key**



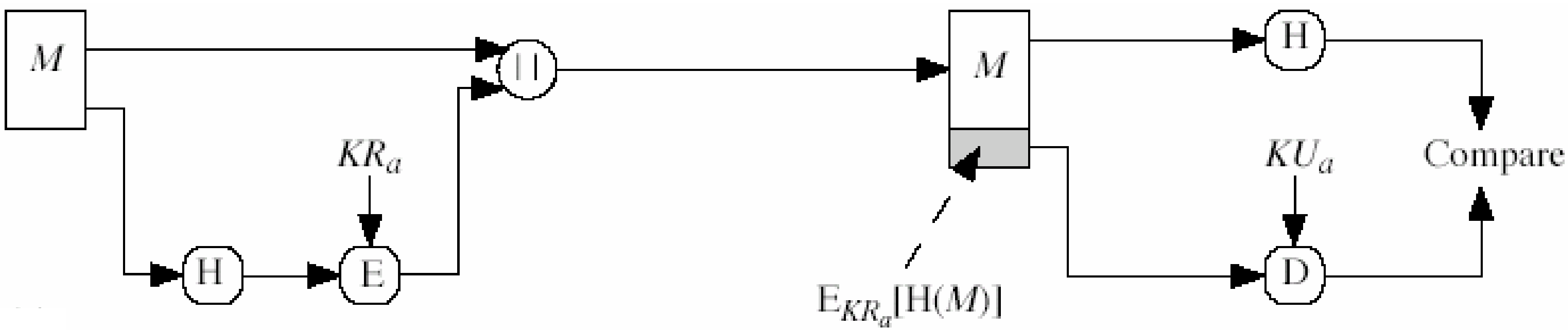


Digital Signatures: signing the hash

- Digital signature created by adding a small authentication block to a message
- Often done by taking the hash of the message and **encrypt the hash** with the **sender's private key**
- The result is a very compact signature (relative to message size)
- And is just as secure as encrypting the entire message with the sender's private key
  - assuming that a secure hash function is used

Typical Use of Hash Function with Digital Signature

- Just sign the hash
  - much more efficient than signing full message



KR_a: Sender's Private Key

KU_a: Sender's Public Key

<u>Note</u>: The | | symbol means *concatenate*; i.e. join inputs together

Digital Certificates

(Public) Key Management

- **Q.** When you receive a public key, how can you be sure that it is authentic?

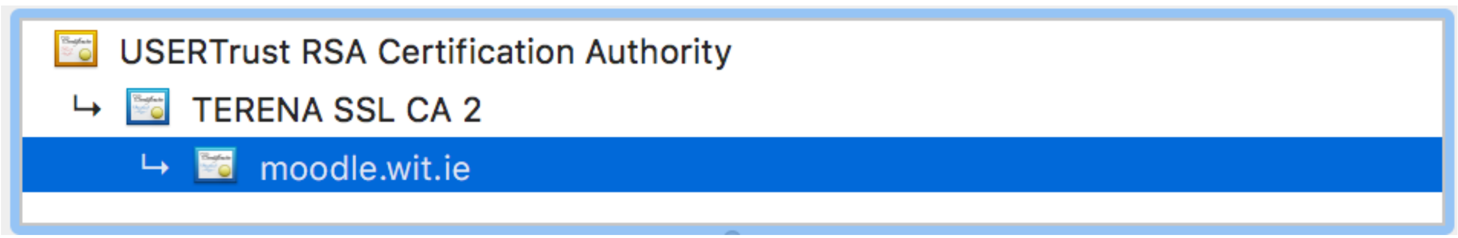
- **A.** If the received public key is **digitally signed** by someone whose own public key you have and are sure is correct **and** you trust them to sign keys responsibly.

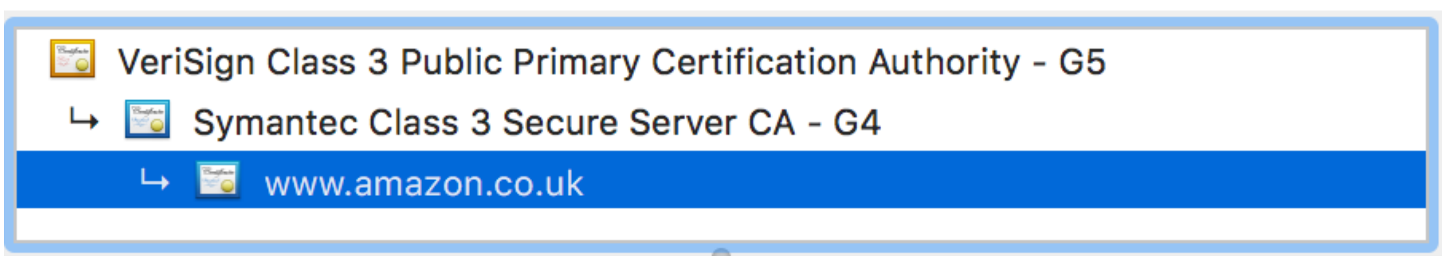
Digital Certificate – components

- Most important components of a digital certificate:
  - Subject (owner)
    - The name on the certificate – i.e. to whom it was issued
  - Subject’s public key
    - The purpose of a certificate is to validate the public key of the subject
  - Issuer (Certificate Authority)
    - The identity of entity that signed the certificate
  - Issuer’s digital signature
  - Serial number
    - Unique identifier for checking against revocation lists
  - Validity period
    - Start date; expiry date

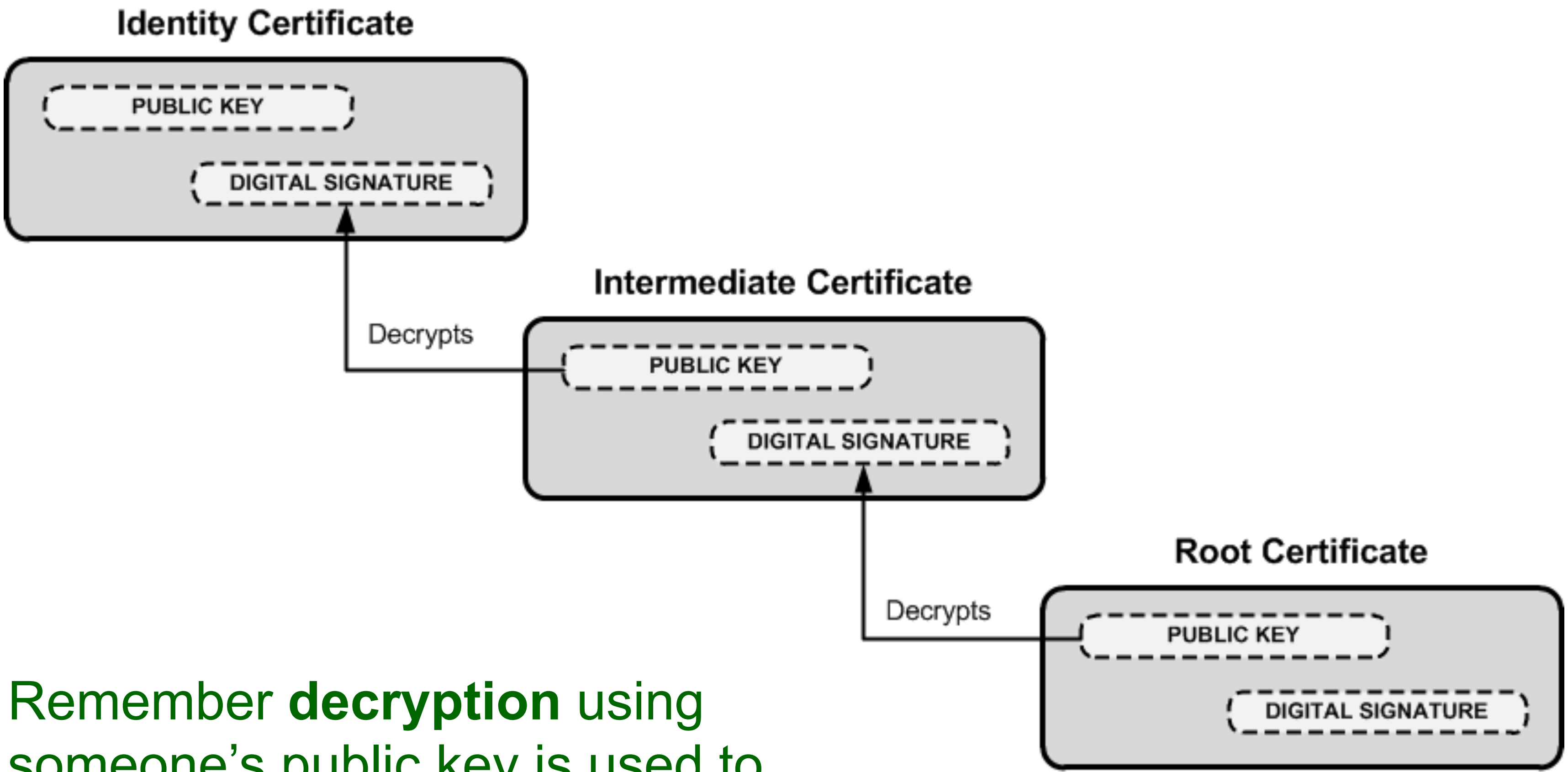
Chain of trust

- Can build up a chain of trusts with linked digital certificates
- This is the basis of what are known as Public Key Infrastructures (PKIs)





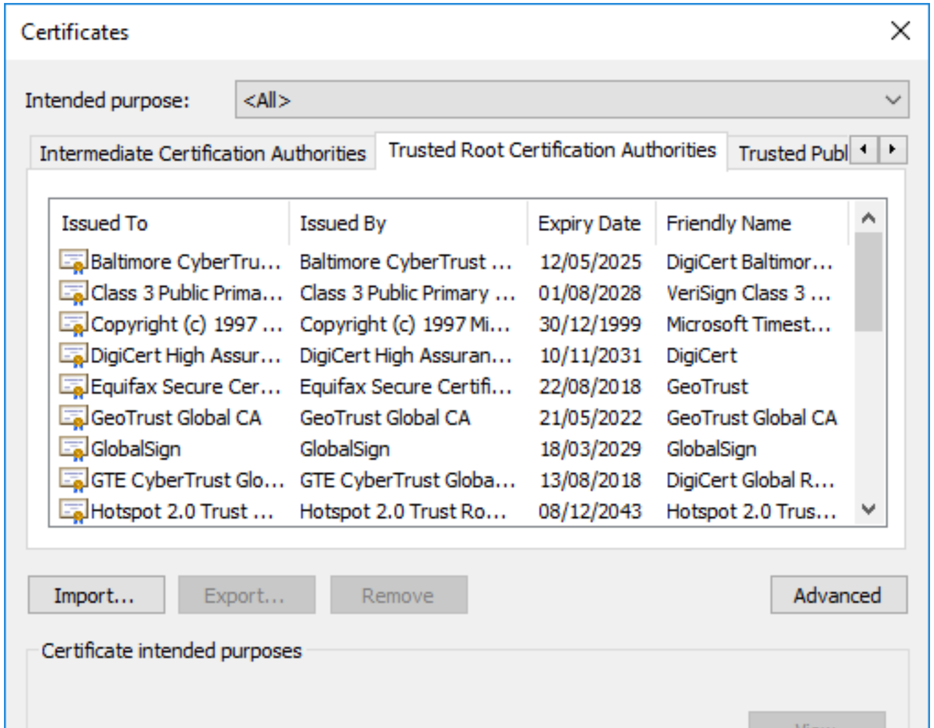
Verification using chain of trusts



Remember **decryption** using someone's public key is used to **verify** their signature

Chain of trust

- The buck must stop somewhere. Ultimately, at the end of the chain, you must trust a public key that is not signed (usually belonging to some recognised “authority”).
  - In your browser, this is one of the trusted root certificate authorities



Certificate Expiry & Revocation

- A Digital Certificate doesn't last for ever
- It normally **expires** after a certain time and must be renewed
- It may be **revoked:**
  - If the subject's private key is compromised
  - If there is a change in status of the subject
  - If the CA's private key is compromised
- Revoked certificates are placed on a Certificate Revocation List (CRL)

Certificate Revocation

- An issue is where to find CRL to check if cert has been revoked
  - One solution is to provide as part of certificate URL pointing to CRL
  - Another solution is OCSP (online certificate status protocol) which allows real time queries.
  - Another is to just rely on local list which is refreshed by browser updates (Chrome does this)