# Overview of Cryptographic Tools for Data Security

Murat Kantarcioglu

# Cryptographic Primitives

- We will discuss the following primitives in this course
  - Symmetric Encryption
  - Message Authentication
  - Public Key Cryptography
  - Digital Signatures
  - Pseudo-random Number Generators

# Block Ciphers

- Consider a block cipher as a permutation defined on n bit strings to n bit strings based on the secret key.
- It is assumed that if the key is secret the output of the block cipher will look like random

# Iterated Block Cipher

- Requires the specification of an invertible round function g and key schedule function Ks and Number of rounds Nr.

$$
\begin{aligned}
&F(K, x) \\
&\{ \\
&\quad (K^1, \ldots K^{Nr}) \leftarrow Ks(K) \\
&\quad w^0 \leftarrow x \\
&\quad w^i \leftarrow g(w^{i-1}, K^{i-1}) \text{for } Nr \geq i \geq 1 \\
&\quad \text{Return } w^{Nr} \\
&\}
\end{aligned}
$$

# Inverting an Iterated Block Cipher

- Since function g is invertible. We can easily decipher the output of an iterated cipher

$$F^{-1}(K, y)$$
$$\{$$
$$(K^1, \ldots K^{Nr}) \leftarrow Ks(K)$$
$$w^{Nr} \leftarrow y$$
$$w^{i-1} \leftarrow g^{-1}(w^i, K^i) \text{ for } Nr > i \geq 1$$
$$\text{Return } w^0$$
$$\}$$

# History of AES

- Due to limitations of DES (small key and block sizes), NIST started a open process to select a new block cipher.
- 15 proposals submitted to NIST around 1998.
- Rijndael from Belgium chosen as the AES in 2001 after an open process.
- Rijndael is chosen because of its security, performance, efficiency, implementability, and flexibility.

# Overview of AES

- AES has 128 bits block size
- AES has three allowable key sizes |K|={128,192,256}
- AES has variable number of rounds
  - If |K|=128 then Nr=10
  - If |K|=192 then Nr=12
  - If |K|=256 then Nr=14

# Block Ciphers

- Block length is fixed (*n*-bit)
- How to encrypt large messages?
  - Partition into *n*-bit blocks
  - Choose mode of operation
    - Electronic Codebook (ECB),
    - Cipher-Block Chaining (CBC),
    - Cipher Feedback (CFB),
    - Output Feedback (OFB),
    - Counter (CTR)
- Padding schemes

# Evaluation criteria

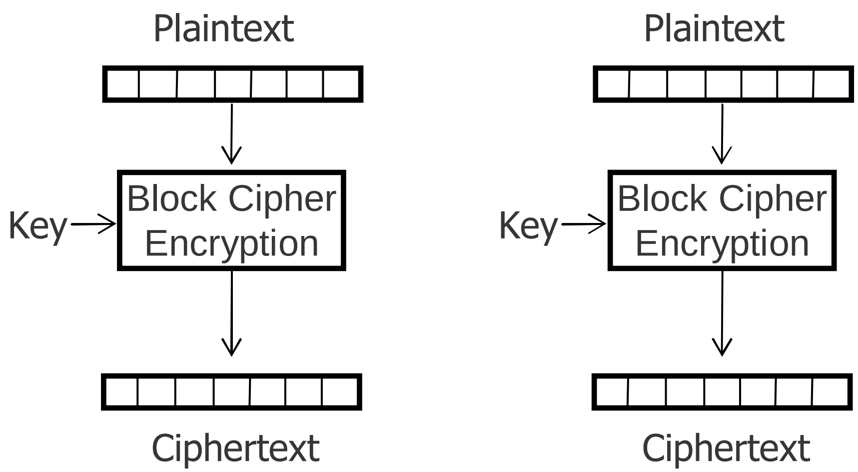
- Identical messages
  - under which conditions ciphertext of two identical messages are the same
- Chaining dependencies
  - how adjacent plaintext blocks affect encryption of a plaintext block
- Error propagation
  - resistance to channel noise
- Efficiency
  - preprocessing
  - parallelization: random access

# Notation

- Message $x$ consists of plaintext blocks of size $n$
  - $x = x_1 \,||\, x_2 \,||\, \ldots \,||\, x_t$
- Ciphertext of plaintext block $x_i$ denoted as $c_i$

- Chaining requires an initialization vector that first plaintext block $x_1$ will depend on. Initialization vector denoted as *IV*.
  - *IV* should be selected randomly for each message ($x$)

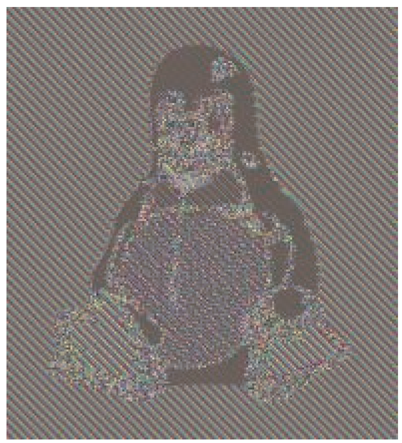
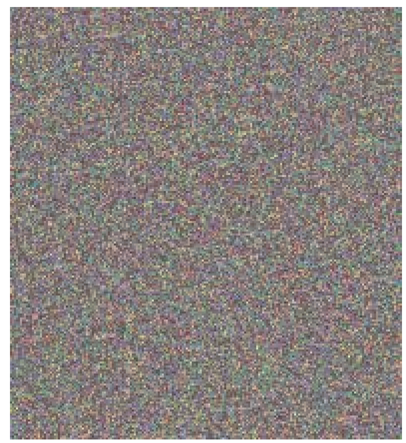
# Electronic Codebook (ECB)




- Each block encrypted independently
- Identical plaintexts encrypted similarly
- No chaining, no error propagation

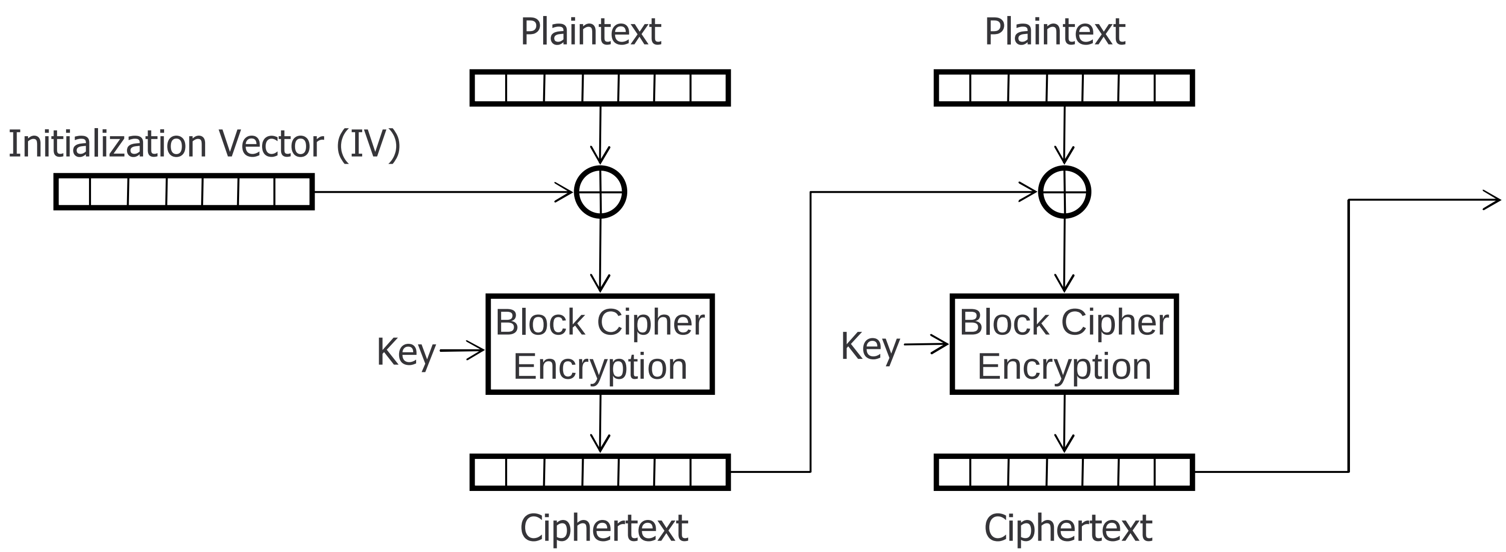
# Electronic Codebook (ECB)

- Does not hide data patterns, unsuitable for long messages
  - Wiki example: pixel map using ECB

- Susceptible to replay attacks
  - Example: a wired transfer transaction can be replayed by re-sending the original message)

# Cipher-Block Chaining (CBC)

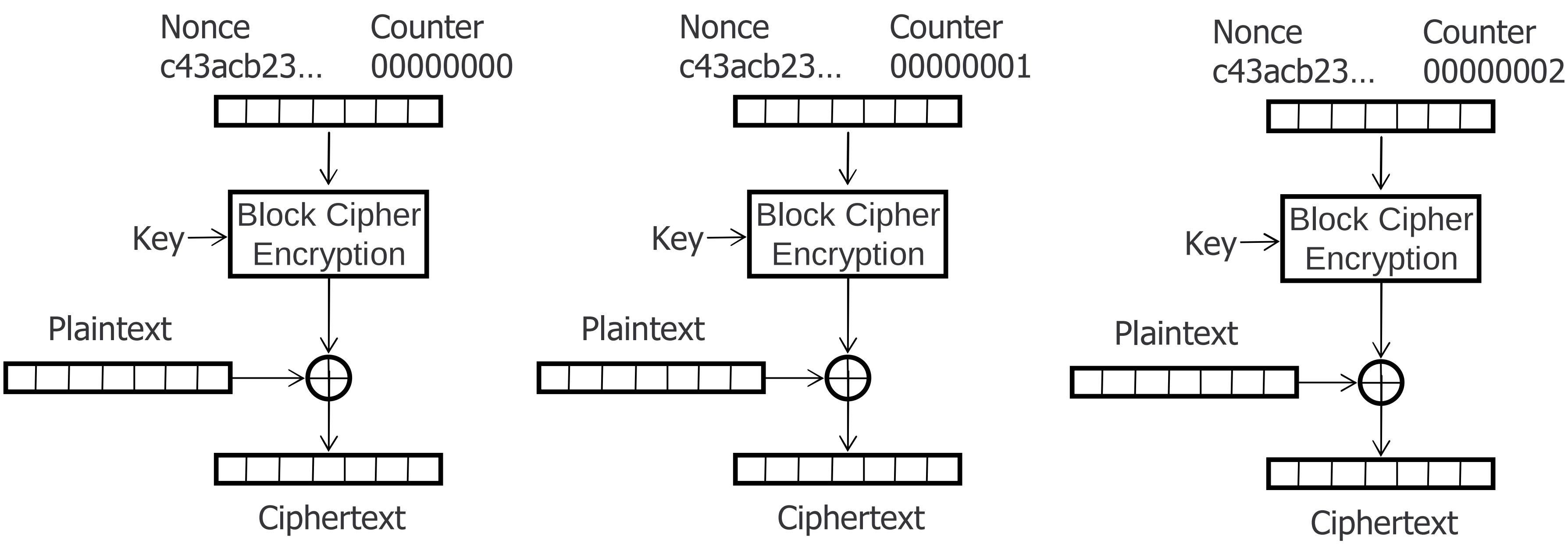



- Allows random access to ciphertext
- Decryption is parallelizable
  - Plaintext block $x_j$ requires ciphertext blocks $c_j$ and $c_{j-1}$

# Cipher-Block Chaining (CBC)

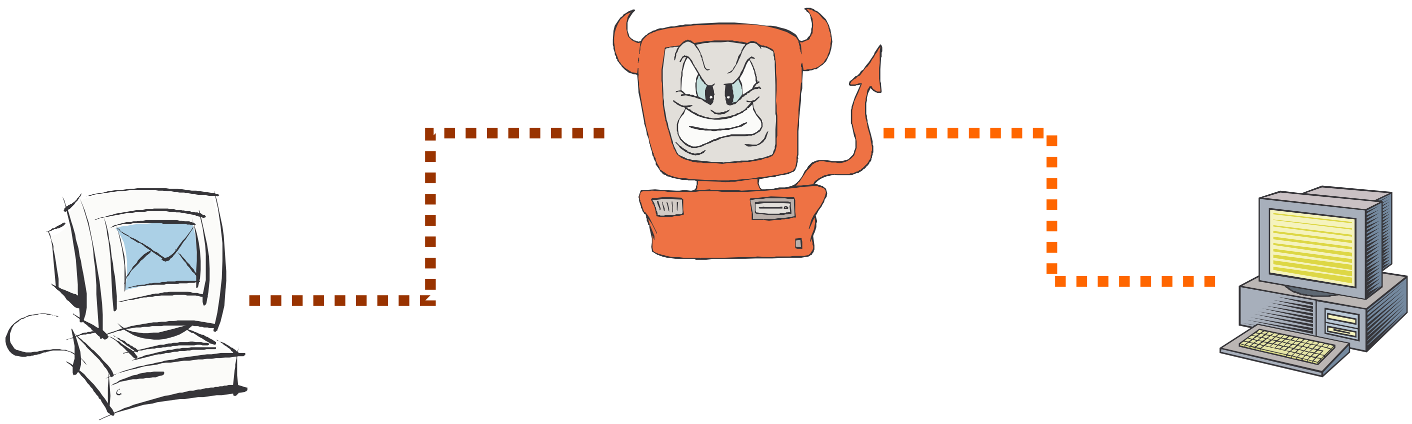
- Identical messages: changing IV or the first plaintext block results in different ciphertext
- Chaining: Ciphertext block $c_j$ depends on $x_j$ and all preceding plaintext blocks (dependency contained in $c_{j-1}$)
- Error propagation: Single bit error on $c_j$ may flip the corresponding bit on $x_{j+1}$, but changes $x_j$ significantly.
- IV need not be secret, but its integrity should be protected

# Counter (CTR)




- Preprocessing possible (inc/decrement and enc/decrypt counter)
- Allows random access

# Data Integrity and Source Authentication



- Encryption does not protect data from modification by another party.
- Need a way to ensure that data arrives at destination in its original form as sent by the sender and it is coming from an authenticated source.

# Cryptographic Hash Functions

- A hash function maps a message of an arbitrary length to a m-bit output
  - output known as the fingerprint or the message digest
  - if the message digest is transmitted securely, then changes to the message can be detected
- A hash is a many-to-one function, so collisions can happen.

# Requirements for Cryptographic Hash Functions

Given a function $h:X \rightarrow Y$, then we say that h is:

- preimage resistant (one-way):

  if given $y \in Y$ it is computationally infeasible to find a value $x \in X$ s.t. $h(x) = y$

- 2-nd preimage resistant (weak collision resistant):

  if given $x \in X$ it is computationally infeasible to find a value $x' \in X$, s.t. $x' \neq x$ and $h(x') = h(x)$

- collision resistant (strong collision resistant):

  if it is computationally infeasible to find two distinct values $x',x \in X$, s.t. $h(x') = h(x)$

# Uses of hash functions

- Message authentication
- Software integrity
- One-time Passwords
- Digital signature
- Timestamping
- Certificate revocation management

# SHA1 (Secure Hash Algorithm)

- SHA was designed by NIST and is the US federal standard for hash functions, specified in FIPS-180 (1993).
- SHA-1, revised version of SHA, specified in FIPS-180-1 (1995) use with Secure Hash Algorithm).
- It produces 160-bit hash values.
- NIST have issued a revision FIPS 180-2 that adds 3 additional hash algorithms:  SHA-256, SHA-384, SHA-512,  designed for compatibility with increased security provided by AES.

# Limitation of Using Hash Functions for Authentication

- Require an authentic channel to transmit the hash of a message
  - anyone can compute the hash value of a message, as the hash function is public
  - not always possible
- How to address this?
  - use more than one hash functions
  - use a key to select which one to use

# Hash Family

- A hash family is a four-tuple ($\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H}$), where
  - $\mathcal{X}$ is a set of possible messages
  - $\mathcal{Y}$ is a finite set of possible message digests
  - $\mathcal{K}$ is the keyspace
  - For each $K \in \mathcal{K}$, there is a hash function $h_K \in \mathcal{H}$ . Each $h_K$: $\mathcal{X} \to \mathcal{Y}$
- Alternatively, one can think of $\mathcal{H}$ as a function $\mathcal{K} \times \mathcal{X} \to \mathcal{Y}$

# Message Authentication Code

- A MAC scheme is a hash family, used for message authentication
- MAC = $C_K(M)$
- The sender and the receiver share K
- The sender sends (M, $C_k(M)$)
- The receiver receives (X,Y) and verifies that $C_K(X)=Y$, if so, then accepts the message as from the sender
- To be secure, an adversary shouldn't be able to come up with (X,Y) such that $C_K(X)=Y$.

# HMAC Goals

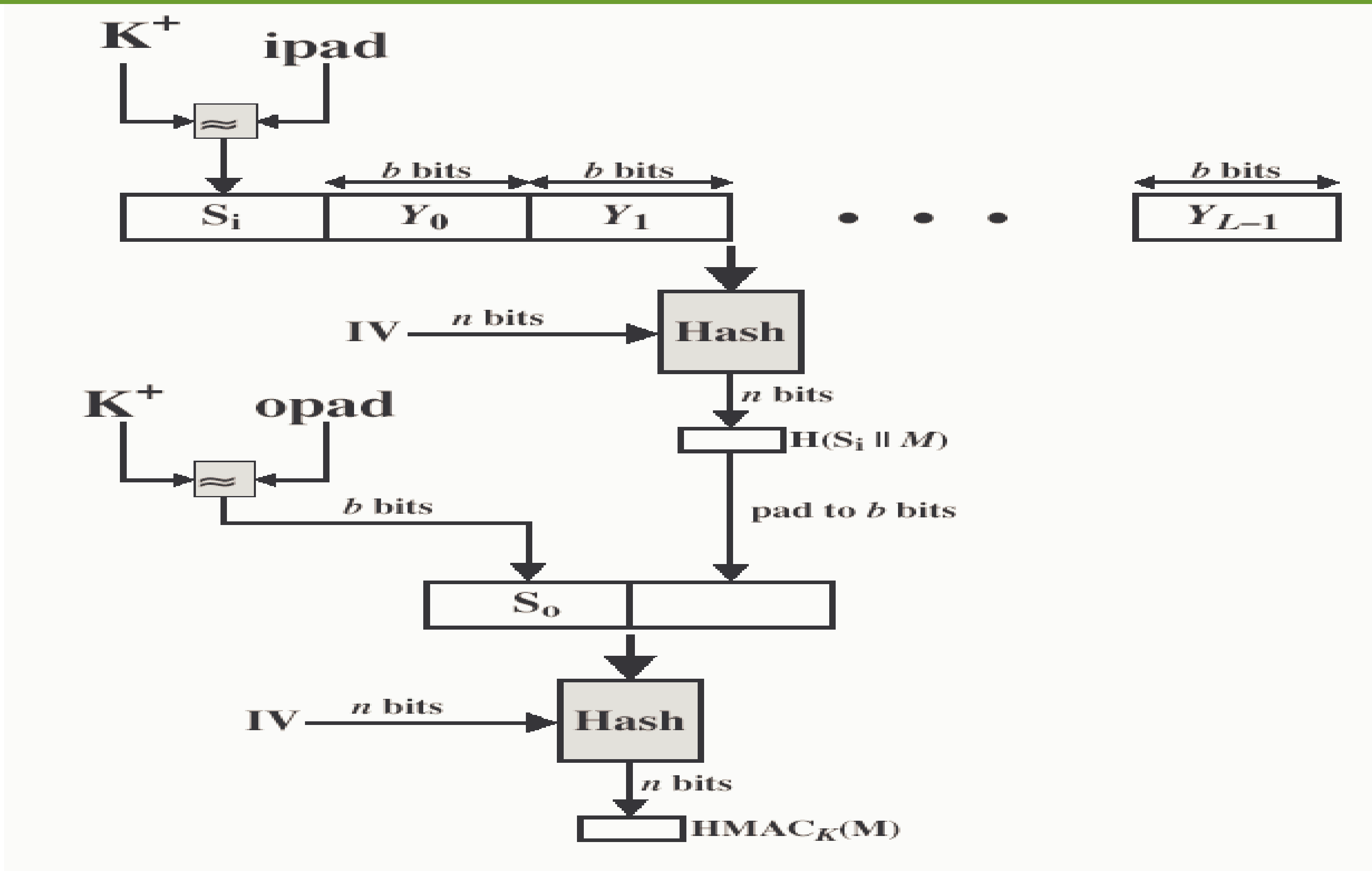
- Use available hash functions without modification.
- Preserve the original performance of the hash function without incurring a significant degradation.
- Use and handle keys in a simple way.
- Allow easy replacement of the underlying hash function in the event that faster or more secure hash functions are later available.
- Have a well-understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions on the underlying hash function.

# HMAC

$$HMAC_K = Hash[(K^+ \oplus opad) \,||\, Hash[(K^+ \oplus ipad)||M)]]$$

- $K^+$ is the key padded out to input block size of the hash function and opad, ipad are specified padding constants
- Key size: $L/2 < K < L$
- MAC size: at least L/2, where L is the hash output

# HMAC Overview



# Limitation of Secret Key (Symmetric) Cryptography

- Secret key cryptography
  - symmetric encryption ⇒ confidentiality (privacy)
  - MAC (keyed hash) ⇒ authentication (integrity)
- Sender and receiver must share the same key
  - needs secure channel for key distribution
  - impossible for two parties having no prior relationship
- Other limitation of authentication scheme
  - cannot authenticate to multiple receivers
  - does not have non-repudiation

# Public Key Cryptography Overview

- Proposed in Diffie and Hellman (1976) “New Directions in Cryptography”
  - public-key encryption schemes
  - public key distribution systems
    - Diffie-Hellman key agreement protocol
  - digital signature
- Public-key encryption was proposed in 1970 by James Ellis
  - in a classified paper made public in 1997 by the British Governmental Communications Headquarters
- Diffie-Hellman key agreement and concept of digital signature are still due to Diffie & Hellman

# Public Key Encryption

- Public-key encryption
  - each party has a PAIR $(K, K^{-1})$ of keys: K is the **public** key and $K^{-1}$ is the **secret** key, such that
    $\mathbf{D}_{K^{-1}}[\mathbf{E}_K[M]] = M$
  - Knowing the public-key and the cipher, it is computationally infeasible to compute the private key
  - Public-key crypto system is thus known to be *asymmetric* crypto systems
  - The public-key K may be made publicly available, e.g., in a publicly available directory
  - Many can encrypt, only one can decrypt

# Public Key Cryptography Overview

- Public key distribution systems
  - two parties who do not share any private information through communications arrive at some secret not known to any eavesdroppers
- Authentication with public keys: Digital Signature
  - the authentication tag of a message can only be computed by one user, but can be verified by many
  - called one-way message authentication in [Diffie & Hellman, 1976]

# Digital Signatures: The Problem

- Consider the real-life example where a person pays by credit card and signs a bill; the seller verifies that the signature on the bill is the same with the signature on the card
- Contracts, they are valid if they are signed.
- Can we have a similar service in the electronic world?

# Digital Signatures

- Digital Signature: a data string which associates a message with some originating entity.
- Digital Signature Scheme: for each key, there is a SECRET signature generation algorithm and a PUBLIC verification algorithm.
- Services provided:
  - Authentication
  - Data integrity
  - Non-Repudiation (MAC does not provide this.)

# RSA Signature

**Key generation (as in RSA encryption):**

- Select 2 large prime numbers of about the same size, p and q
- Compute $n = pq$, and $\Phi = (q - 1)(p - 1)$
- Select a random integer e, $1 < e < \Phi$, s.t. $\gcd(e, \Phi) = 1$
- Compute d, $1 < d < \Phi$ s.t. $ed \equiv 1 \bmod \Phi$

**Public key: (e, n)**

**Secret key: d, p and q must also remain secret**

# RSA Signature (cont.)

**Signing message M**

- M must verify $0 < M < n$
- Use private key (d)
- compute $S = M^d \bmod n$

**Verifying signature S**

- Use public key (e, n)
- Compute $S^e \bmod n = (M^d \bmod n)^e \bmod n = M$

Note: in practice, a hash of the message is signed and not the message itself.

# Implementing Cryptosystems is Hard

- Crypto is not easy !
- Simple changes in the algorithm could make the underlying system insecure !
- CryptoSystems usually fail because of implementation.
- Unlike theory, in practice cryptosystems do not work in isolation.

# Possible Implementation Pitfalls

- Not using publicly tested algorithms
  - Do not use any algorithm that has not been tested by the crypto community extensively.
  - Remember what happened to original DVD encryption
- Not using algorithms correctly
  - I.e., Using AES in ECB mode or RSA function directly.
- Not generating randomness correctly.
  - Note that CBC mode could be insecure if the IV is not generated randomly.

# More on Random Number Generation

- Generic pseudo-random number generation is not secure.

```
procedure srand(seed)    | function rand()
    state = seed;        |     state = ((state * 1103515245) + 12345)
                         |         mod 2147483648;
                         |     return state
```

- Must use provably-secure pseudo-random number generators (see the Anderson book for details.)

# Issues Related to Key Management

- Secret keys should be generated randomly.
- Secret keys should be protected.
  – Your implementation should not leave keys in memory.
  – Need to consider the trust model carefully.
    - i.e., can someone easily access the secret key files?
    - What happens if you have trojan on your computer?
    - What happens if there is a system failure?

# Weakest Link: Users

- Users choose easy to guess passwords.
  - Always make sure that chosen passwords are strong.
- They can be easily tricked into revealing passwords
  - Consider two, three factor authentication methods.