# Cryptography Primer

Chester Rebeiro

IIT Madras

# Cryptography

- A crucial component in all security systems
- Fundamental component to achieve
  - **Confidentiality**


Allows only authorized users access to data

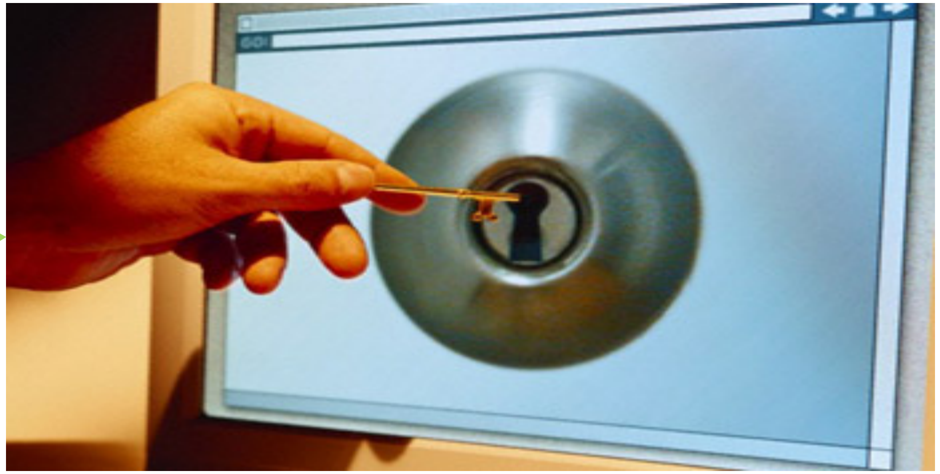
# Cryptography (its use)

- A crucial component in all security systems
- Fundamental component to achieve
  - Confidentiality
  - **Data Integrity**

Cryptography can be used to ensure that only authorized users can make modifications (for instance to a bank account number)

# Cryptography (its use)

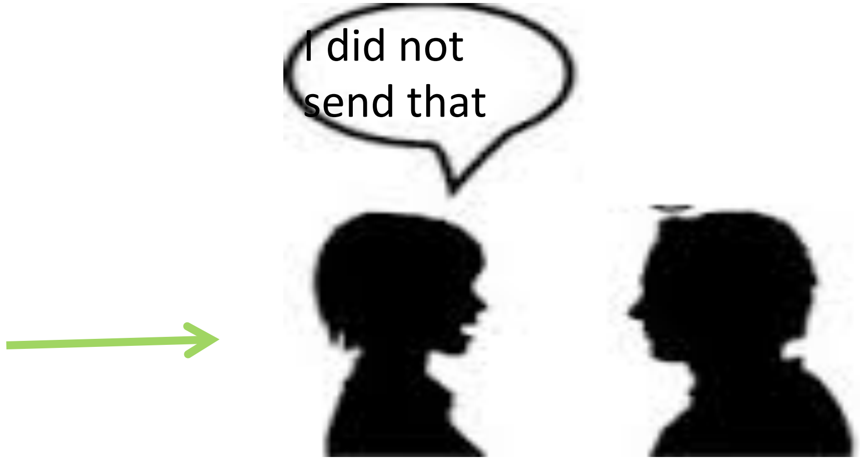
- A crucial component in all security systems
- Fundamental component to achieve
  - Confidentiality
  - Data Integrity
  - **Authentication**




Cryptography helps prove identities

# Cryptography
(its use)

- A crucial component in all security systems
- Fundamental component to achieve
  - Confidentiality
  - Data Integrity
  - Authentication
  - **Non-repudiation**




The sender of a message cannot claim that she did not send it

# Scheme for Confidentiality

Alice

message

Attack at Dawn!!

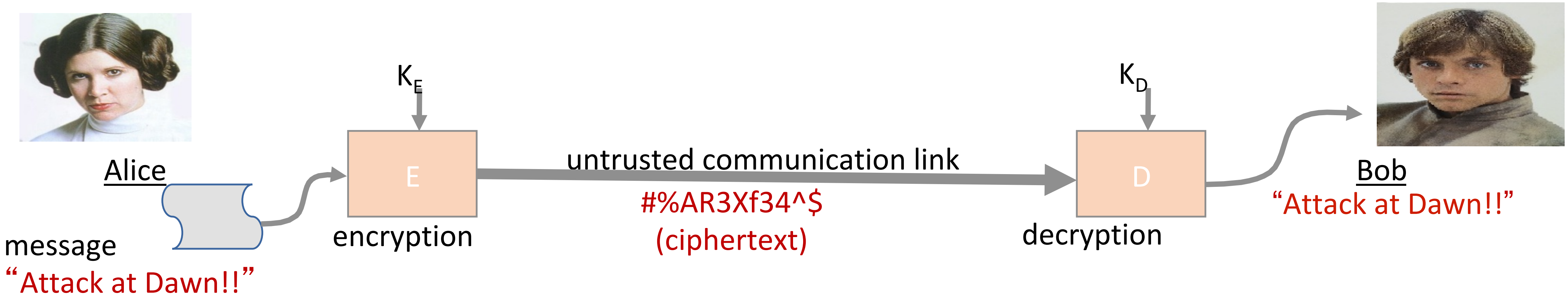
untrusted communication link

Bob

Mallory

**Problem :** Alice wants to send a message to Bob (and only to Bob) through an untrusted communication link

# Encryption



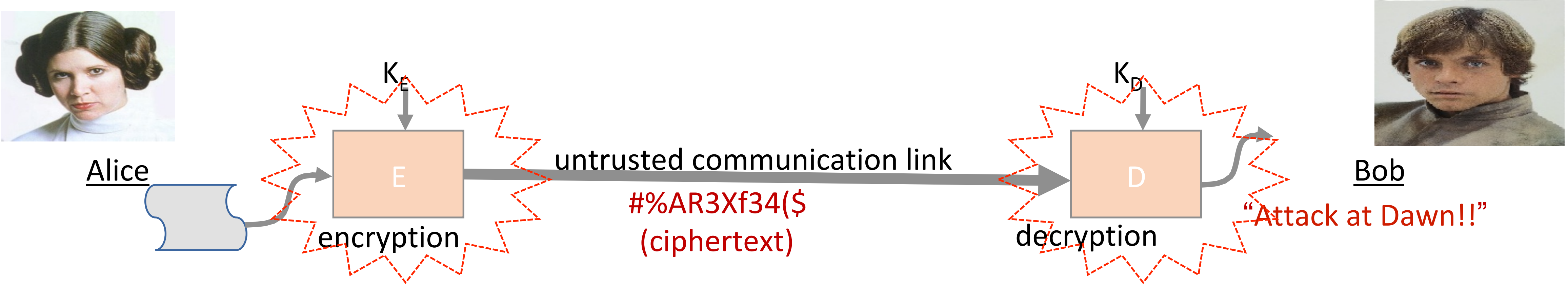

**Secrets**

- Only Alice knows the encryption key $K_E$
- Only Bob knows the decryption key $K_D$

Mallory

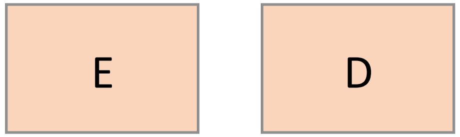
Only sees ciphertext.
cannot get the plaintext message
because she does not know the keys

# Encryption Algorithms




- Should be **easy to compute** for Alice / Bob (who **know the key**)
- Should be **difficult to compute** for Mallory (who **does not know the key**)
- What is **'difficult'** ?
    - **Ideal case :** Prove that the probability of Mallory determining the encryption / decryption key is ***no better than a random guess***
    - **Computationally :** Show that it is ***difficult*** for Mallory to determine the keys even if she has massive computational power

# Ciphers




- Symmetric Algorithms
  - Encryption and Decryption use the same key
  - i.e. $K_E = K_D$
  - Examples:
    - Block Ciphers : DES, AES, PRESENT, etc.
    - Stream Ciphers : A5, Grain, etc.
- Asymmetric Algorithms
  - Encryption and Decryption keys are different
  - $K_E \neq K_D$
  - Examples:
    - RSA
    - ECC

# Encryption Keys


- How are keys managed
  - How does Alice & Bob select the keys?
  - Need algorithms for key exchange

# Algorithmic Attacks

- Can Mallory use tricks to break the algorithm

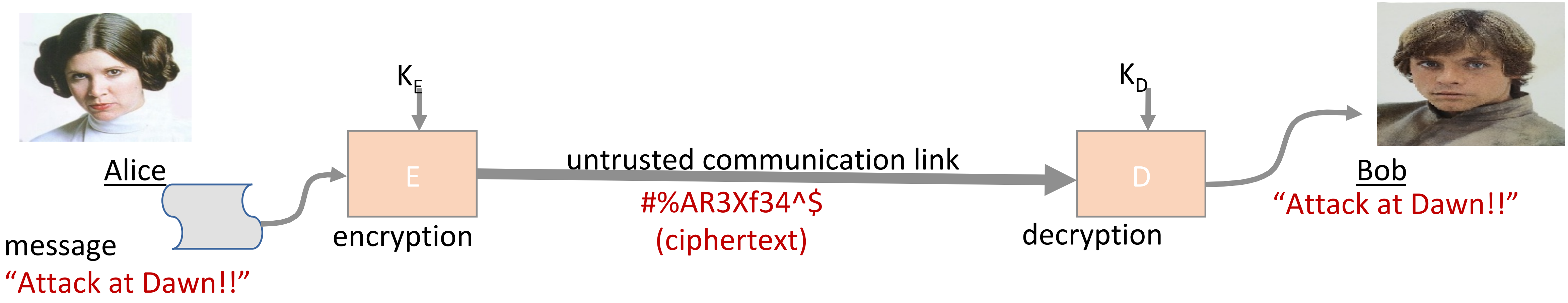

- There by reducing the ‘difficulty’ of getting the key.

# Block Ciphers

Chester Rebeiro

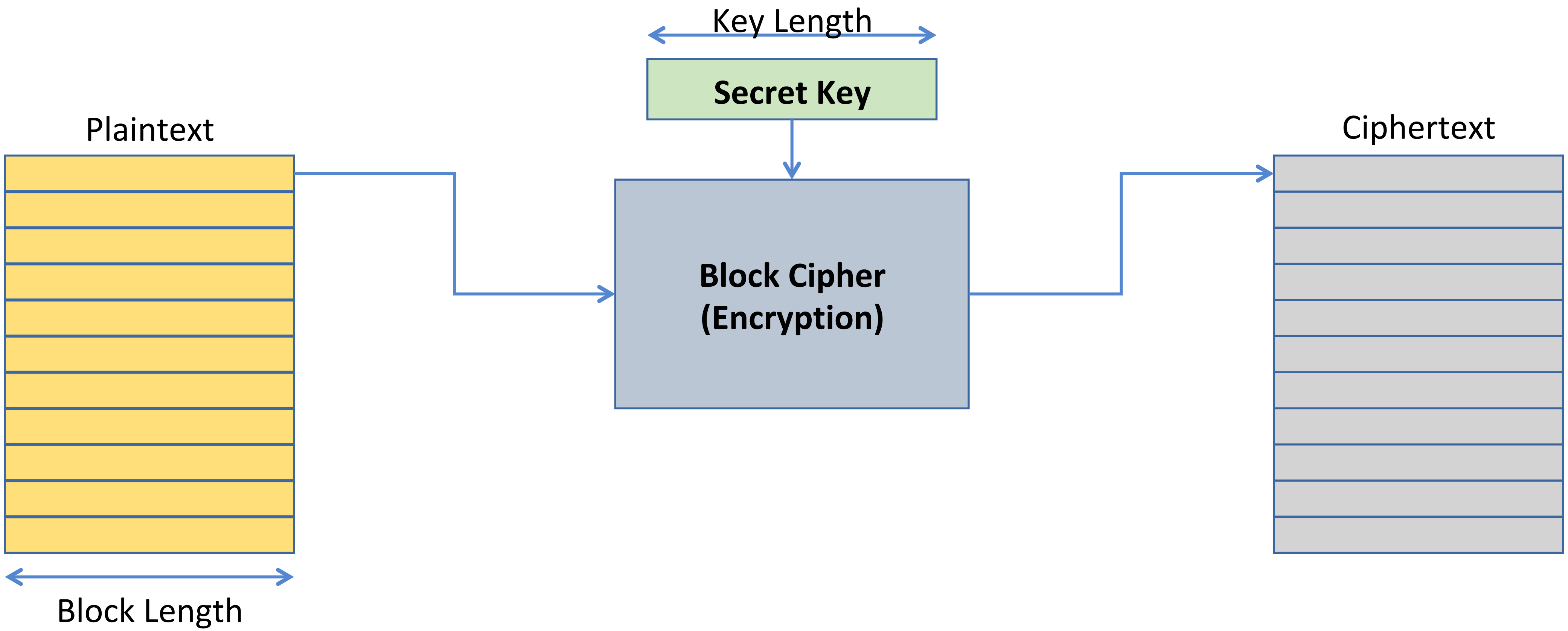
IIT Madras

# Block Cipher



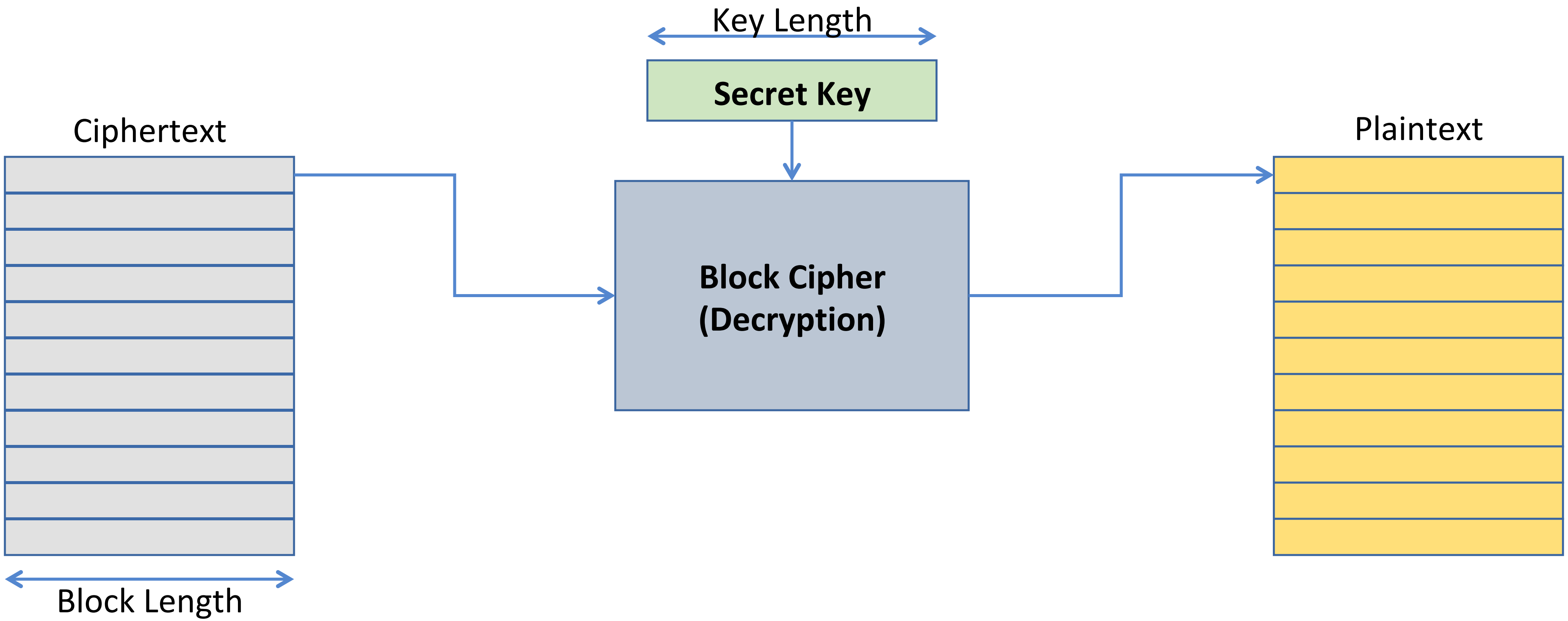

Encryption key is the same as the decryption key ($K_E = K_D$)

# Block Cipher : Encryption



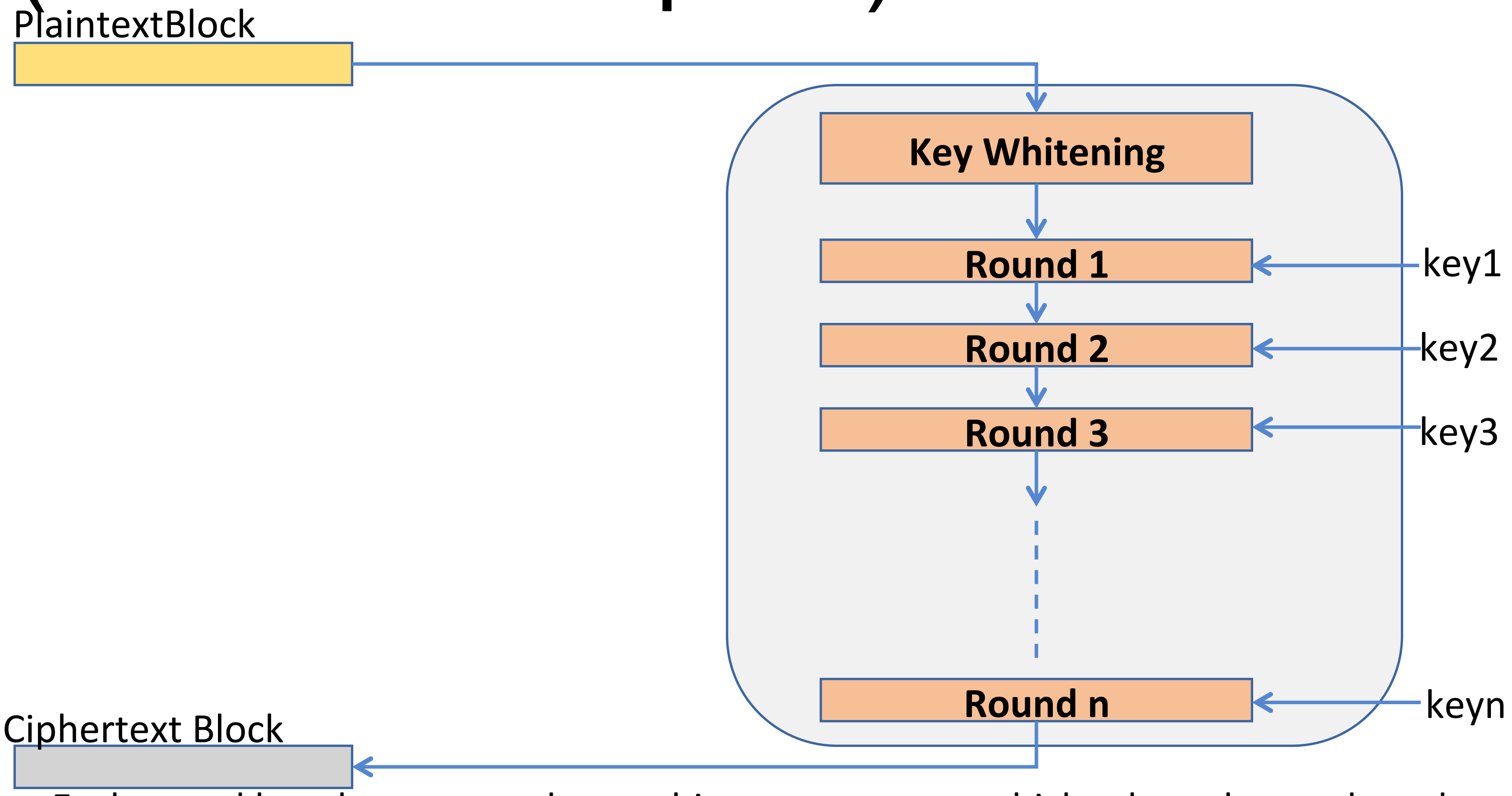

- A block cipher encryption algorithm encrypts n bits of plaintext at a time
- May need to pad the plaintext if necessary
- $y = e_k(x)$

# Block Cipher : Decryption



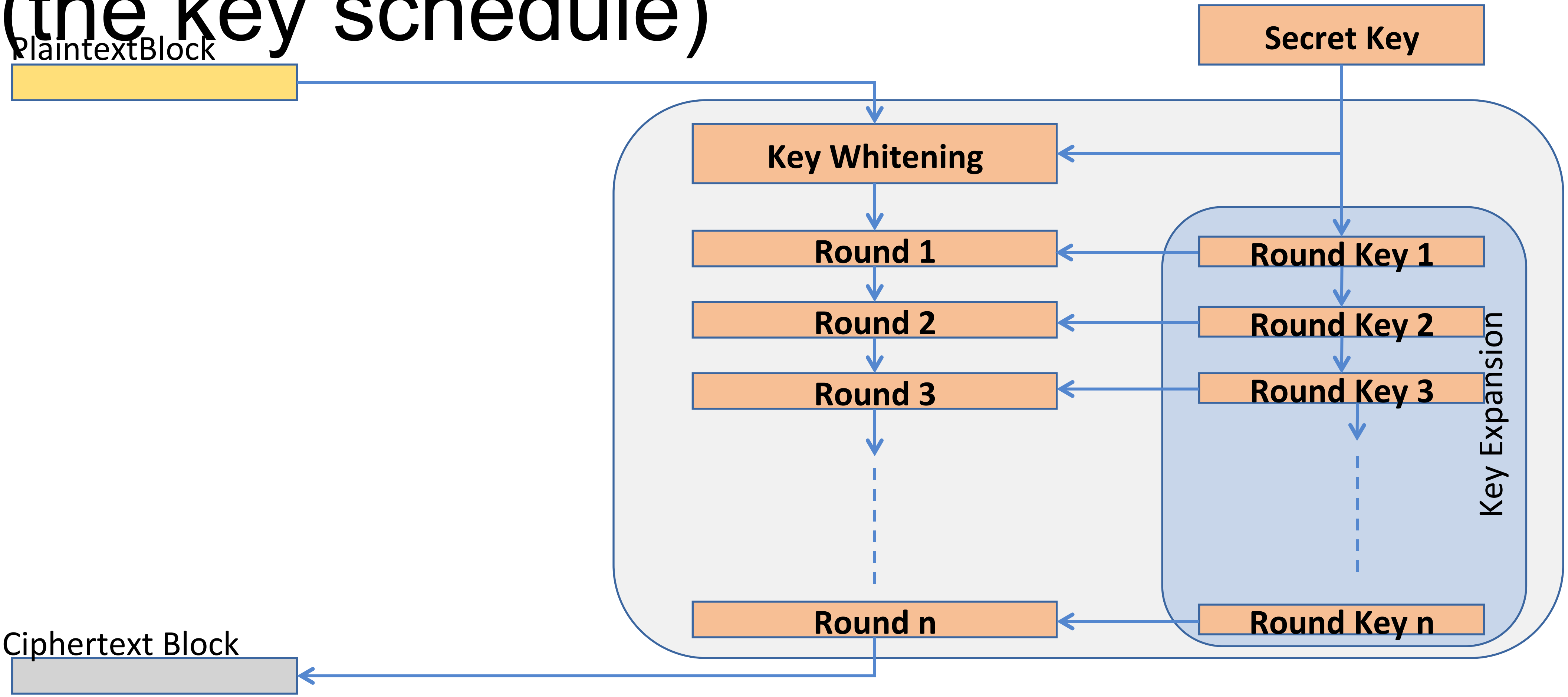

- A block cipher decryption algorithm recovers the plaintext from the ciphertext.
- $x = d_k(y)$

# Inside the Block Cipher (an iterative cipher)



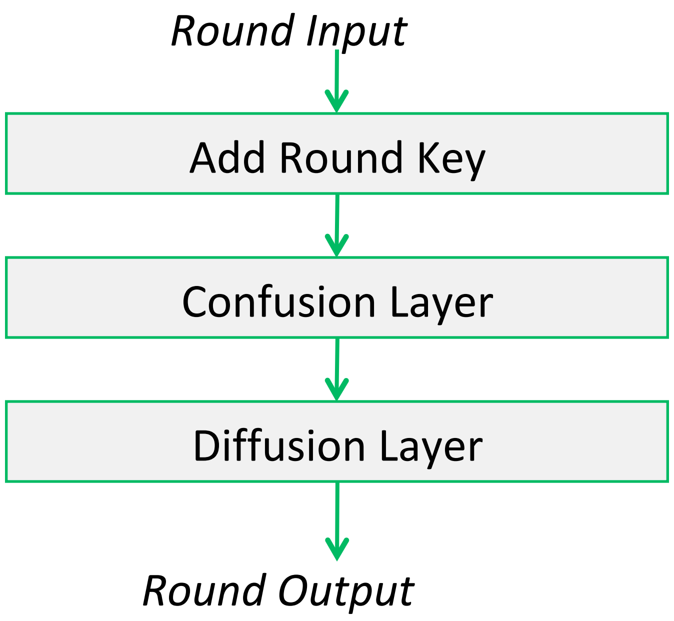

- Each round has the same endomorphic cryptosystem, which takes a key and produces an intermediate ouput
- Size of the key is huge… much larger than the block size.

# Inside the Block Cipher (the key schedule)




- A single secret key of fixed size used to generate 'round keys' for each round

# Inside the Round Function




- ***Add Round key :***

  Mixing operation between the round input
  and the round key.
  typically, an ex-or operation

- ***Confusion layer :***

  Makes the relationship between round
  input and output complex.

- ***Diffusion layer :***
  dissipate the round input.
  *Avalanche effect :* A single bit change in the round input should cause huge changes in the output.

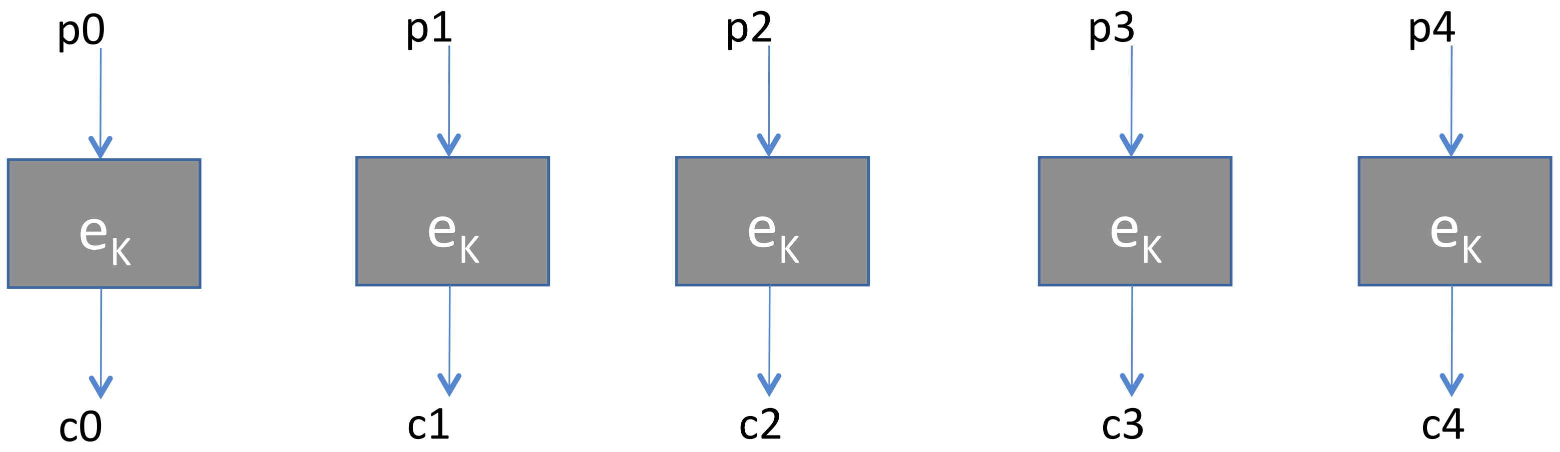
  Makes it difficult for the attacker to pick out some bits over the others (think Hill cipher)

# Modes of Operation

# What are Modes of Operation?

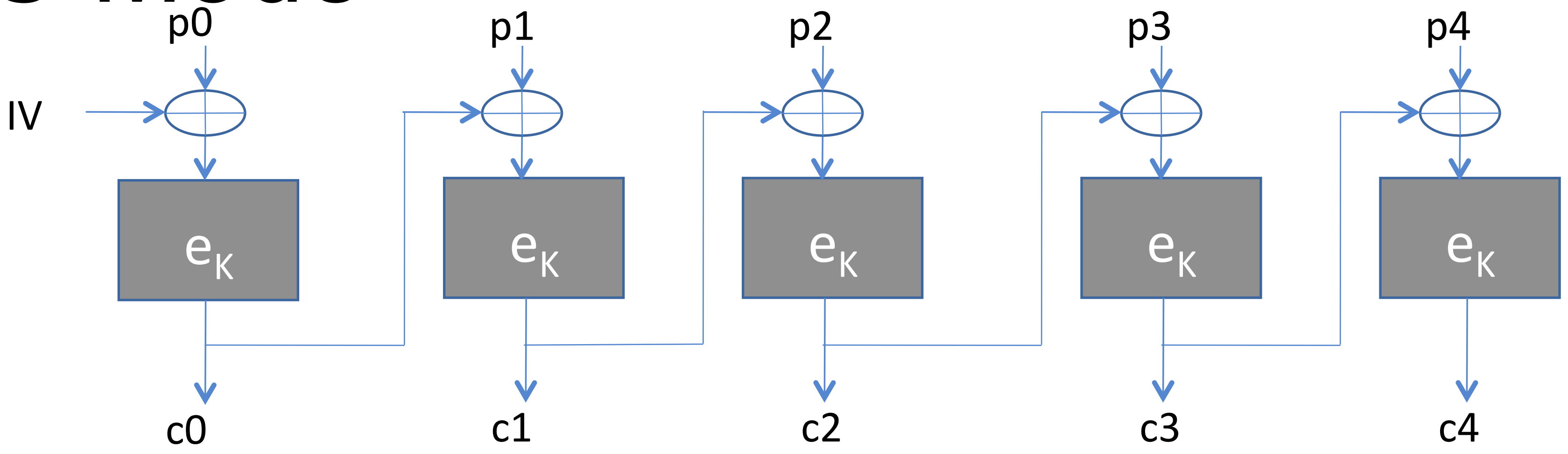
- Block cipher algorithms only encrypt a single block of message
- A mode of operation describes how to repeatedly apply a cipher's single-block operation to securely transform amounts of data larger than a block
- Modes of Operation
  - Electronic code book mode (ECB Mode)
  - Cipher feedback mode (CFB Mode)
  - Cipher block chaining mode (CBC mode)
  - Output feedback mode (OFB mode)
  - Counter mode

# ECB Mode



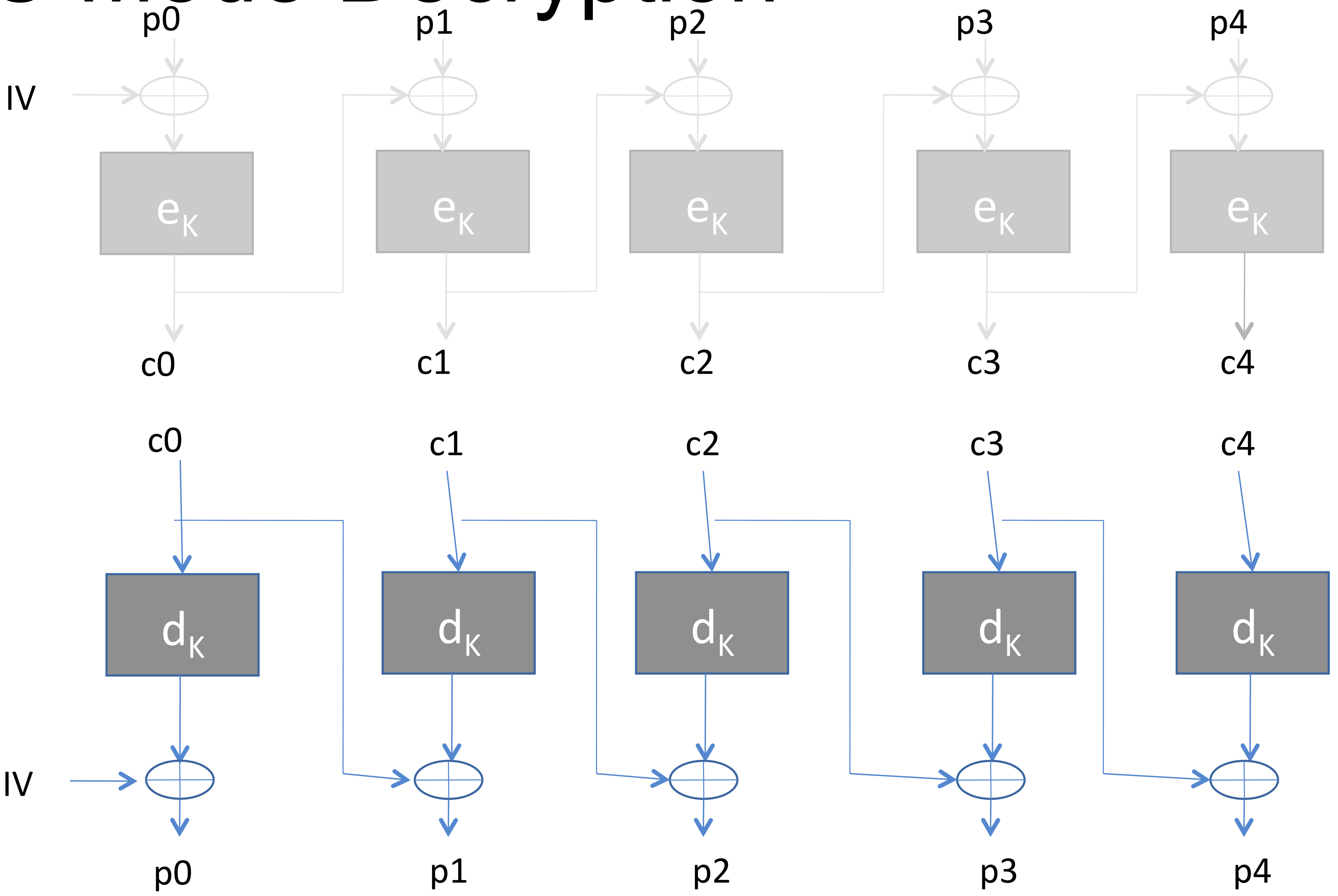

- Every block in the message is encrypted independently with the same key
- Drawback 1 : If $p_i = p_j$ $(i \neq j)$ then $c_i = c_j$
  - Encryption should protect against known plaintext attacks (since the attacker could guess parts of the message..... Like stereotype beginnings)
- Drawback 2 : An interceptor may alter the order of the blocks during transmission
- Not recommended for encryption of more than one block

# CBC Mode




- Cipher Block Chaining
- Advantage 1 : Encryption dependent on the ciphertext of a previous block, therefore
  - $c_i \neq c_j$ $(i \neq j)$ even if $p_i = p_j$
- Advantage 2: Intruder cannot alter the order of the blocks during transmission
- If an error is present in one received block (say $c_i$)
  - Then $c_i$ and $c_{i+1}$ will not be decrypted correctly
  - All remaining blocks will be correctly decrypted

# CBC Mode Decryption

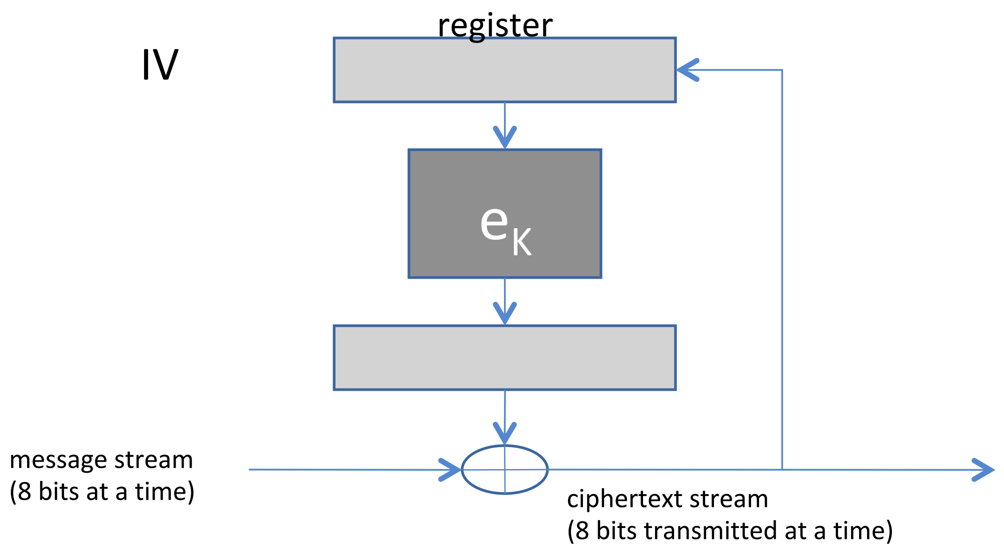


# CFB (Cipher feedback Mode)

Can transform a block cipher into a stream cipher.

- i.e. Each block encrypted with a different key

Uses a shift register that is initialized with an IV

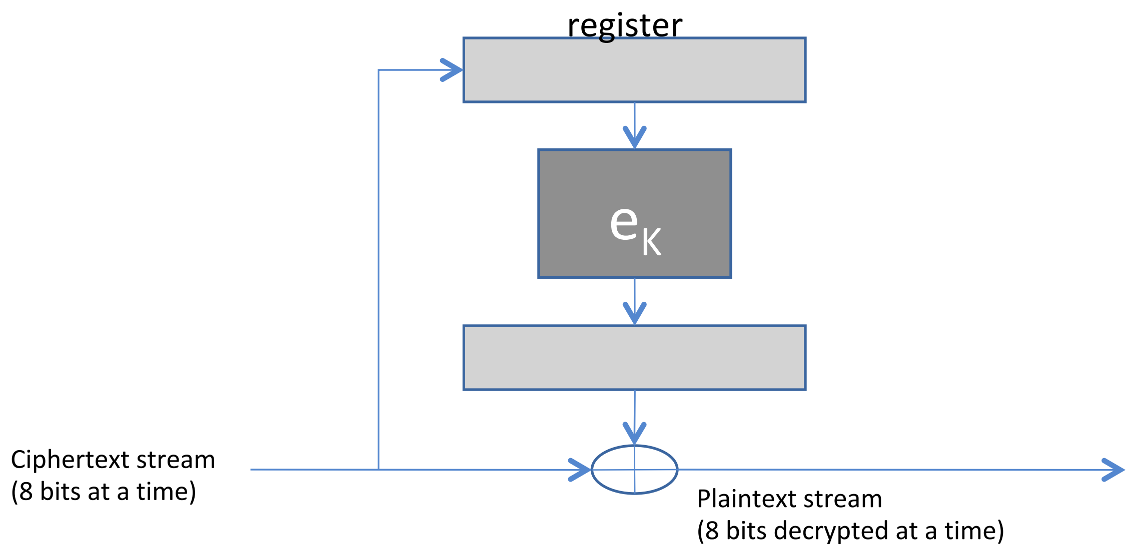



Encryption Scheme

# CFB - Error Propagation

Uses a shift register that is initialized with an IV

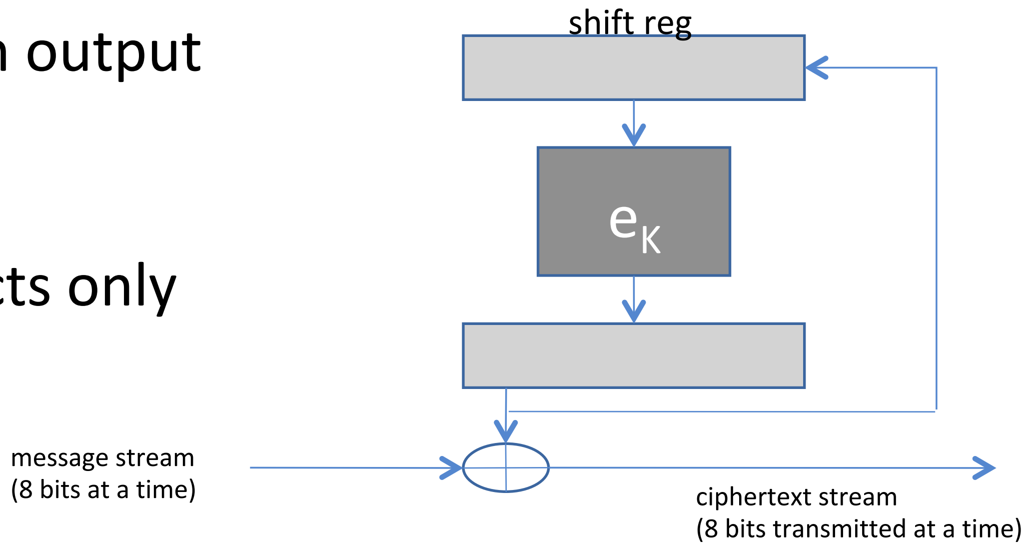
Previous ciphertext block fed into shift register




Decryption Scheme

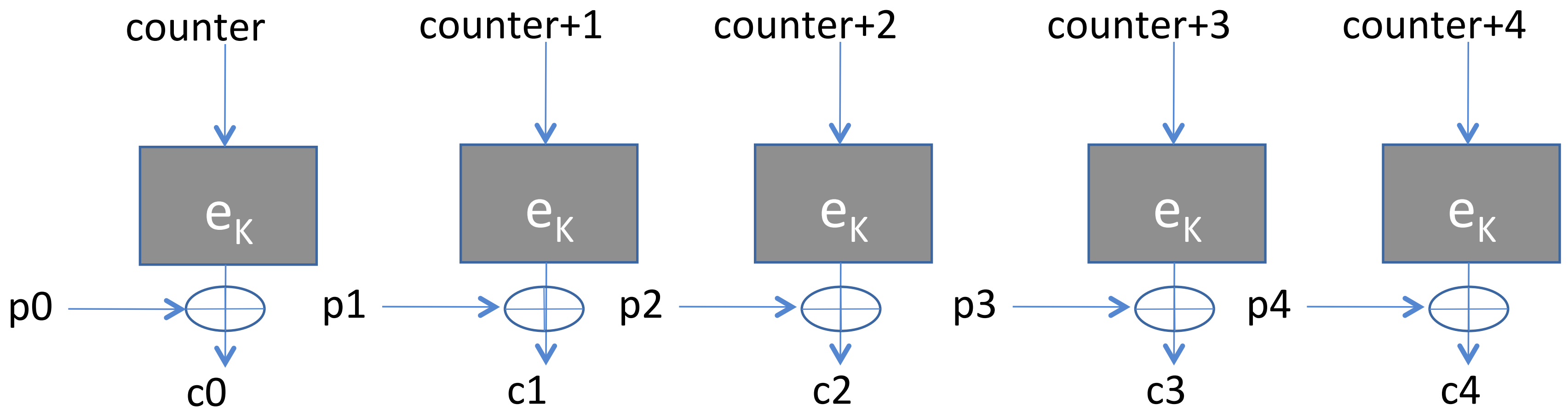
# Output Feedback Mode (OFB)

- Very similar to CFB but feedback taken from output of $e_k$
- An error in one byte of the ciphertexts affects only one decryption




Encryption Scheme
(Decryption scheme is similar)

# Counter Mode




- A randomly initialized counter is incremented with every encryption
- Can be parallelized
  - Ie. Multiple encryption engines can simultaneously run
- As with OFB, an error in a single ciphertext block affects only one decrypted plaintext

# Cryptographic Hash Functions

# Issues with Integrity

Alice

Message

"Attack at Dawn!!"

unsecure channel

Bob

"Attack at Dusk!!"

Change 'Dawn' to 'Dusk'

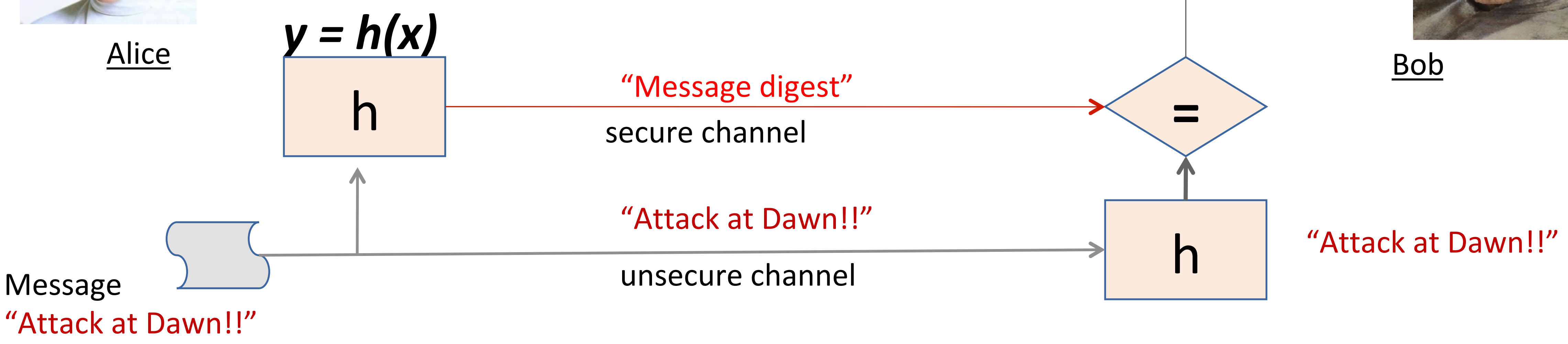
**How can Bob ensure that Alice's message has not been modified?**

**Note…. We are not concerned with confidentiality here**

# Hashes

Alice

Bob




**Alice passes the message through a hash function, which produces a fixed length message digest.**

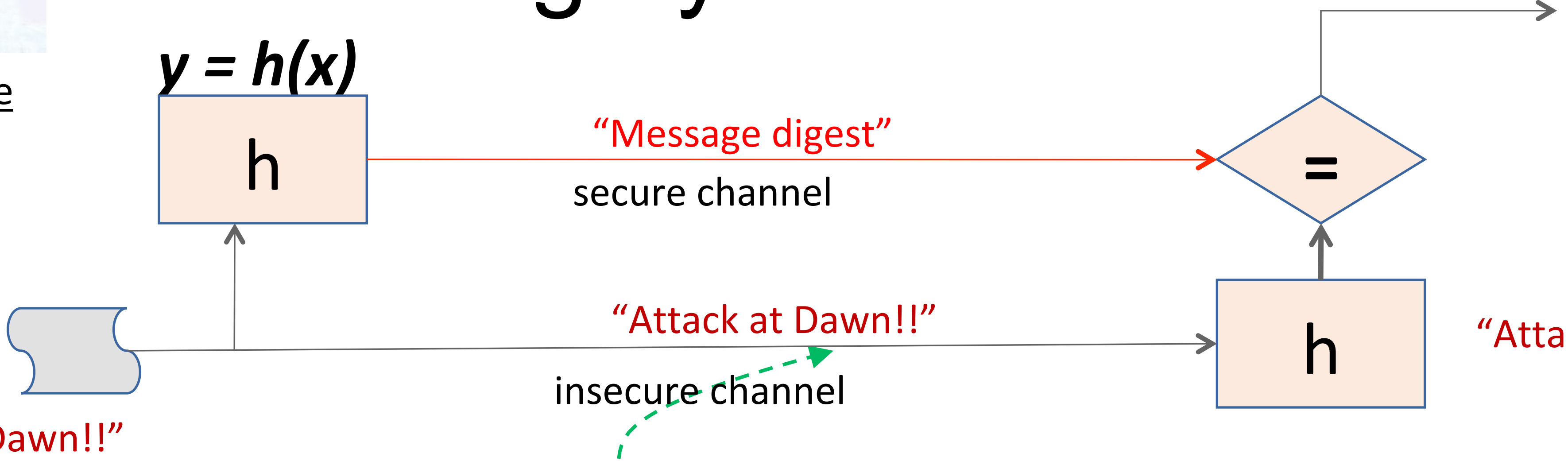
- **The message digest is representative of Alice's message.**
- **Even a small change in the message will result in a completely new message digest**
- **Typically of 160 bits, irrespective of the message size.**

**Bob re-computes a message hash and verifies the digest with Alice's message digest.**

# Integrity with Hashes

Alice

Bob




"Attack at Dawn!!"

Message
"Attack at Dawn!!"

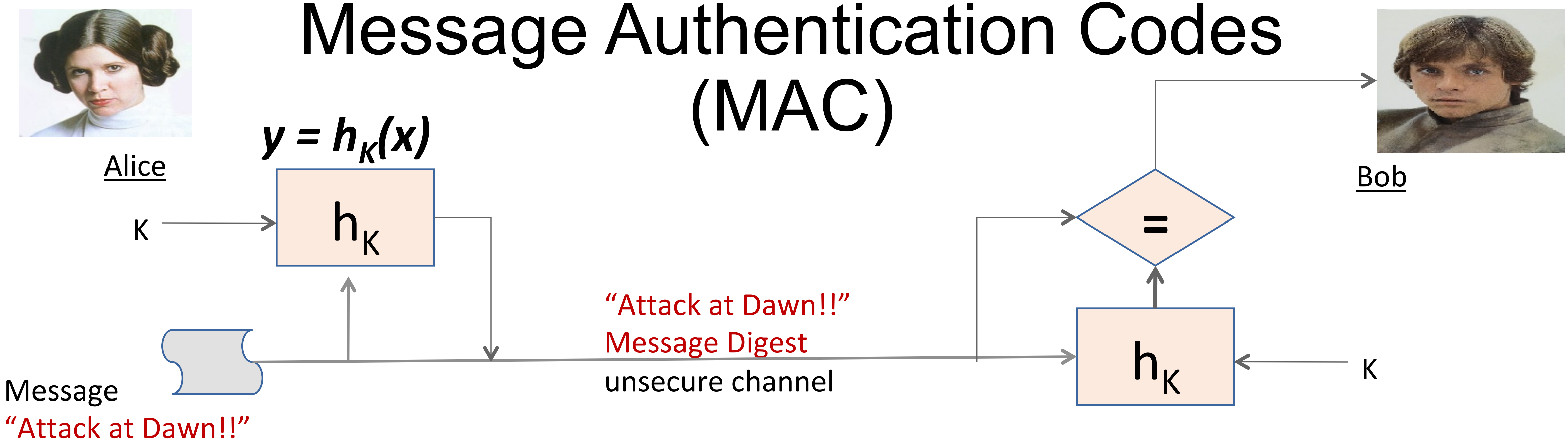
***y = h(x)***
***y = h(x')***

Mallory does not have access to the digest y.
Her task (to modify Alice's message) is much more difficult.

If she modifies x to x', the modification can be detected unless h(x) = h(x')

**Hash functions are specially designed to resist such collisions**

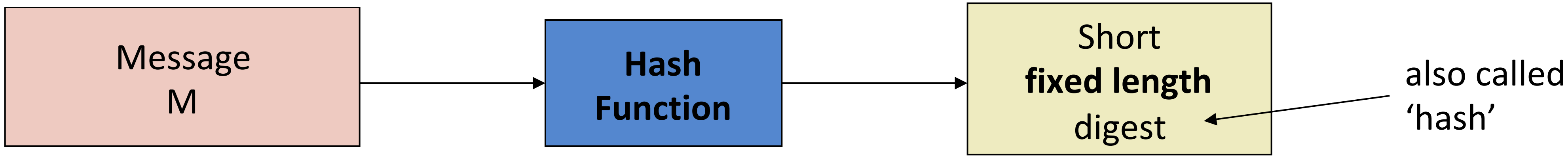
# Message Authentication Codes (MAC)




**MACs allow the message and the digest to be sent over an insecure channel**

**However, it requires Alice and Bob to share a common key**

# Avalanche Effect




Hash functions provide unique digests with high probability.

Even a small change in **M** will result in a new digest

SHA256("short sentence")

0x 0acdf28f4e8b00b399d89ca51f07fef34708e729ae15e85429c5b0f403295cc9

SHA256("The quick brown fox jumps over the lazy **dog**")

0x d7a8fbb307d7809469ca9abcb0082e4f8d5651e46d3cdb762d02d0bf37c9e592

SHA256("The quick brown fox jumps over the lazy **dog.**")

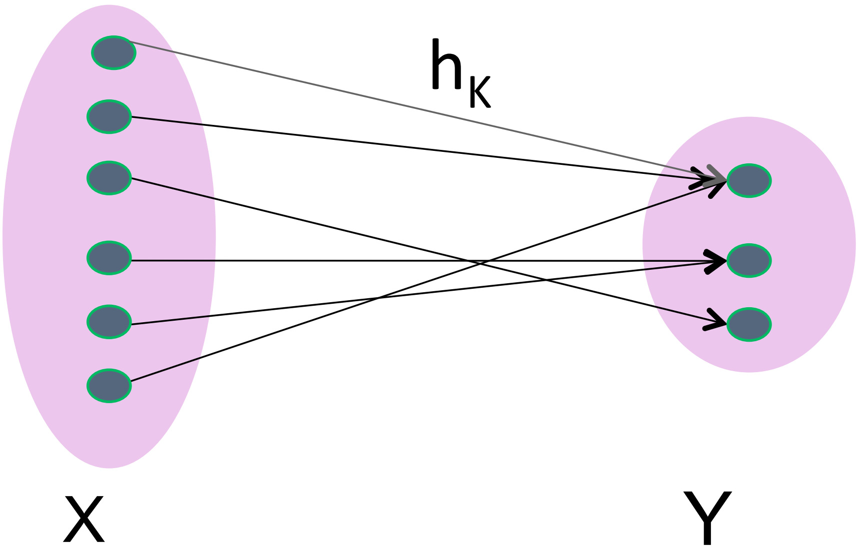
**(extra period added)**

0x ef537f25c895bfa782526529a9b63d97aa631564d5d789c2b765448c8635fb6c

# Hash functions in Security

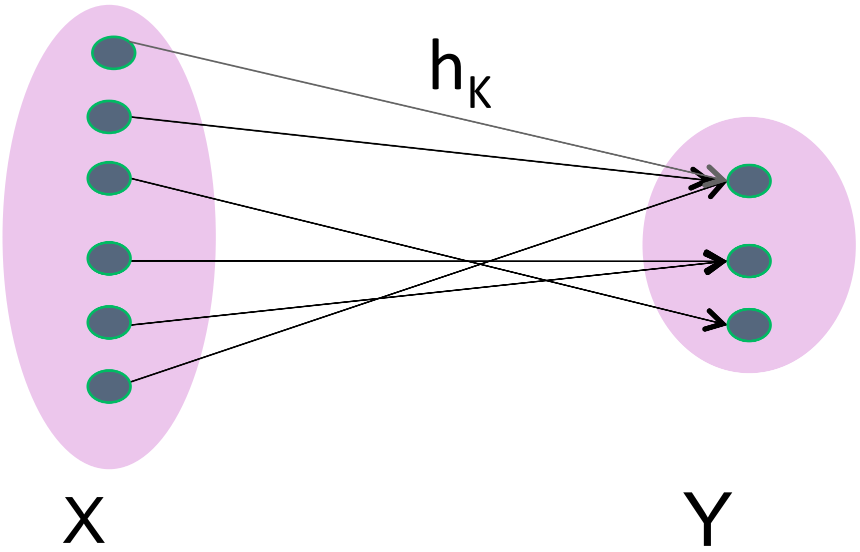
- Digital signatures
- Random number generation
- Key updates and derivations
- One way functions
- MAC
- Detect malware in code
- User authentication (storing passwords)

# Hash Family



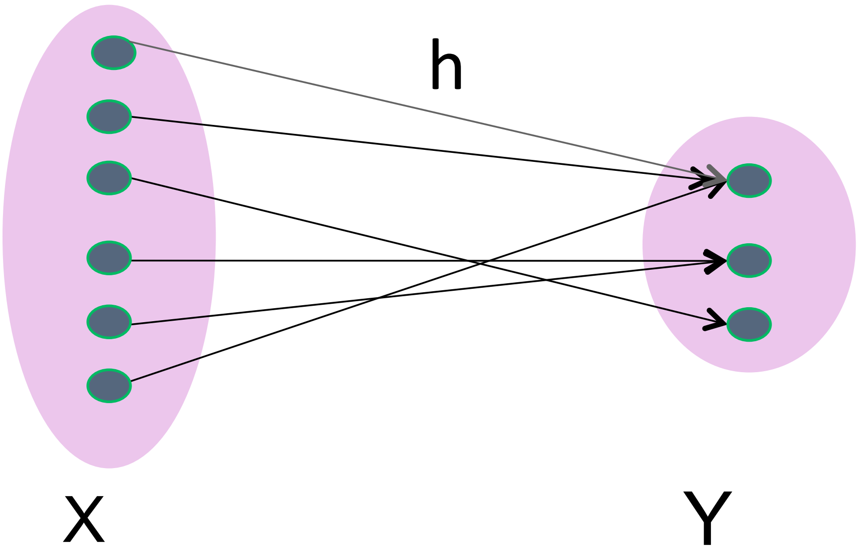

- The hash family is a 4-tuple defined by (X,Y,K,H)
- X is a set of messages (may be infinite)
- Y is a finite set of message digests (aka authentication tags)
- K is a finite set of keys
- Each K Ɛ K, defines a keyed hash function $h_K$ Ɛ H

# Hash Family : some definitions




- Valid pair under K : (x,y) Ɛ Xxy such that, x = $h_K(y)$
- Size of the hash family: is the number of functions possible from set X to set Y; |Y| = M and |X| = N then the number of mappings possible is $M^N$
- The collection of all such mappings are termed (N,M)-hash mapping.

# Unkeyed Hash Function




- The hash family is a 4-tuple defined by (X,Y,K,H)
- X is a set of messages
  (may be infinite, we assume the minimum size is at least 2|Y| )
- Y is a finite set of message digests
- In an unkeyed hash function : |K | = 1
- We thus have only one mapping function in the family

# Security Aspects of Unkeyed Hash Functions

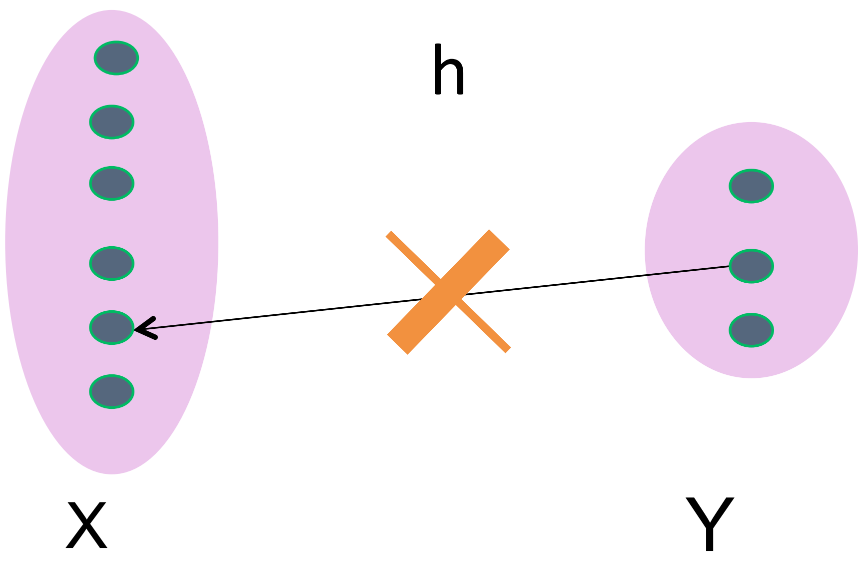
h = X → Y

y = h(x) -----> no shortcuts in computing. The only valid way if computing y is to invoke the hash function h on x

- Three problems that define security of a hash function
  * Preimage Resistance
  * Second Preimage Resistance
  * Collision Resistance

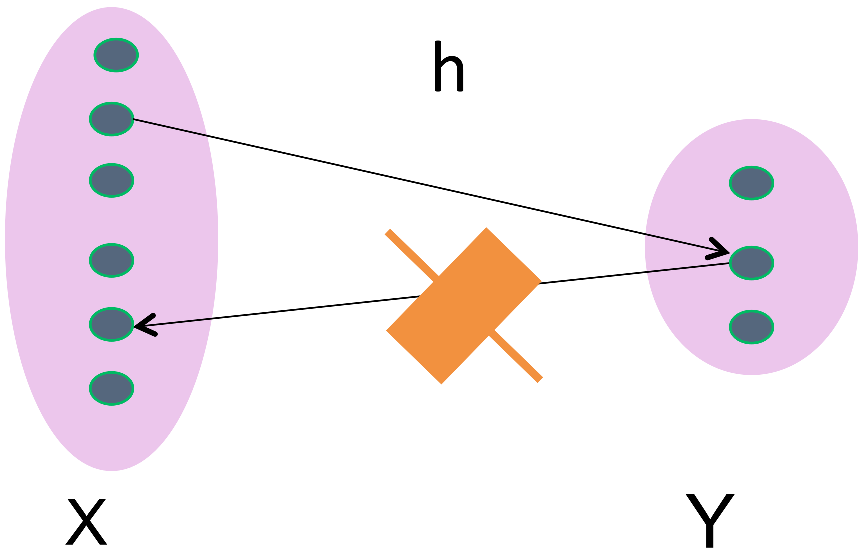
# Hash function Requirement 1
# Preimage Resistant

- Also know as **one-wayness problem**
- If Mallory happens to know the message digest, she should not be able to determine the message
- Given a hash function h : X →Y and an element y Ɛ Y. Find any x Ɛ X such that, h(x) = y



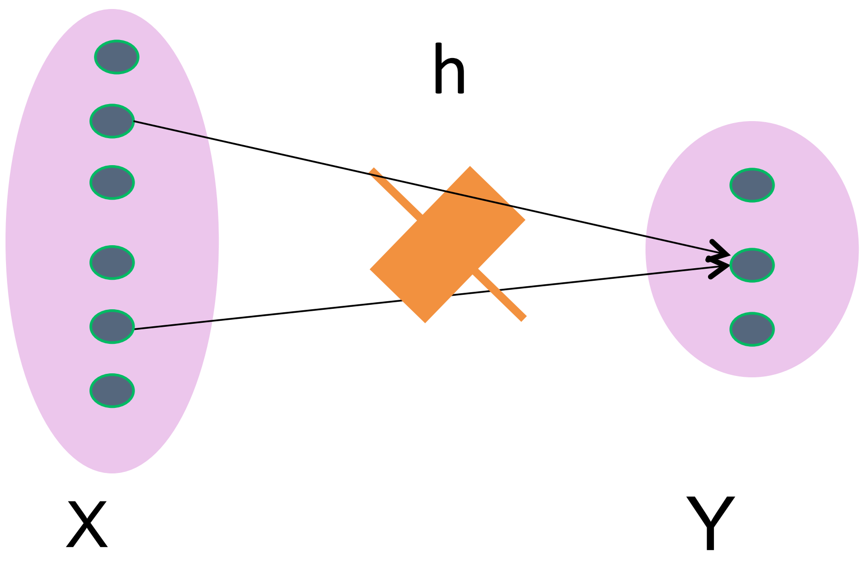
# Hash function Requirement 2 (Second Preimage)

- Mallory has x and can compute h(x), she should not be able to find another message x' which produces the same hash.
  - It would be easy to forge new digital signatures from old signatures if the hash function used weren't second preimage resistant
- Given a hash function h : X →Y and an element x Ɛ X, find, x' Ɛ X such that, h(x) = h(x')



# Hash Function Requirement (Collision Resistant)

- Mallory should not be able to find two messages x and x’ which produce the same hash
- Given a hash function h : X →Y and an element x Ɛ X, find, x, x’ Ɛ X and x ≠x’ such that, h(x) = h(x’)




There is no collision Free hash Function but hash functions can be designed so that collisions are difficult to find.

# Finding Collisions

```
Find_Collisions(h,  Q){
    choose Q distinct values from X (say x_1, x_2, …., x_Q)
     for(i=1; i<=Q; ++i) y_i = h(x_i)
     if  there exists (y_j == y_k) for j ≠k then return (x_j, x_k)
    return FAIL
}
```

$$Success\Pr obability\,(\varepsilon)\,is\,\varepsilon = 1 - \prod_{i=1}^{Q-1}\left(1 - \frac{i}{M}\right)$$

# Birthday Paradox

- Find the probability that at-least two people in a room have the same birthday

$$Event\ A: atleast\ two\ people\ in\ the\ room\ have\ the\ same\ birthday$$

$$Event\ A': no\ two\ people\ in\ the\ room\ have\ the\ same\ birthday$$

$$\Pr[A] = 1 - \Pr[A']$$

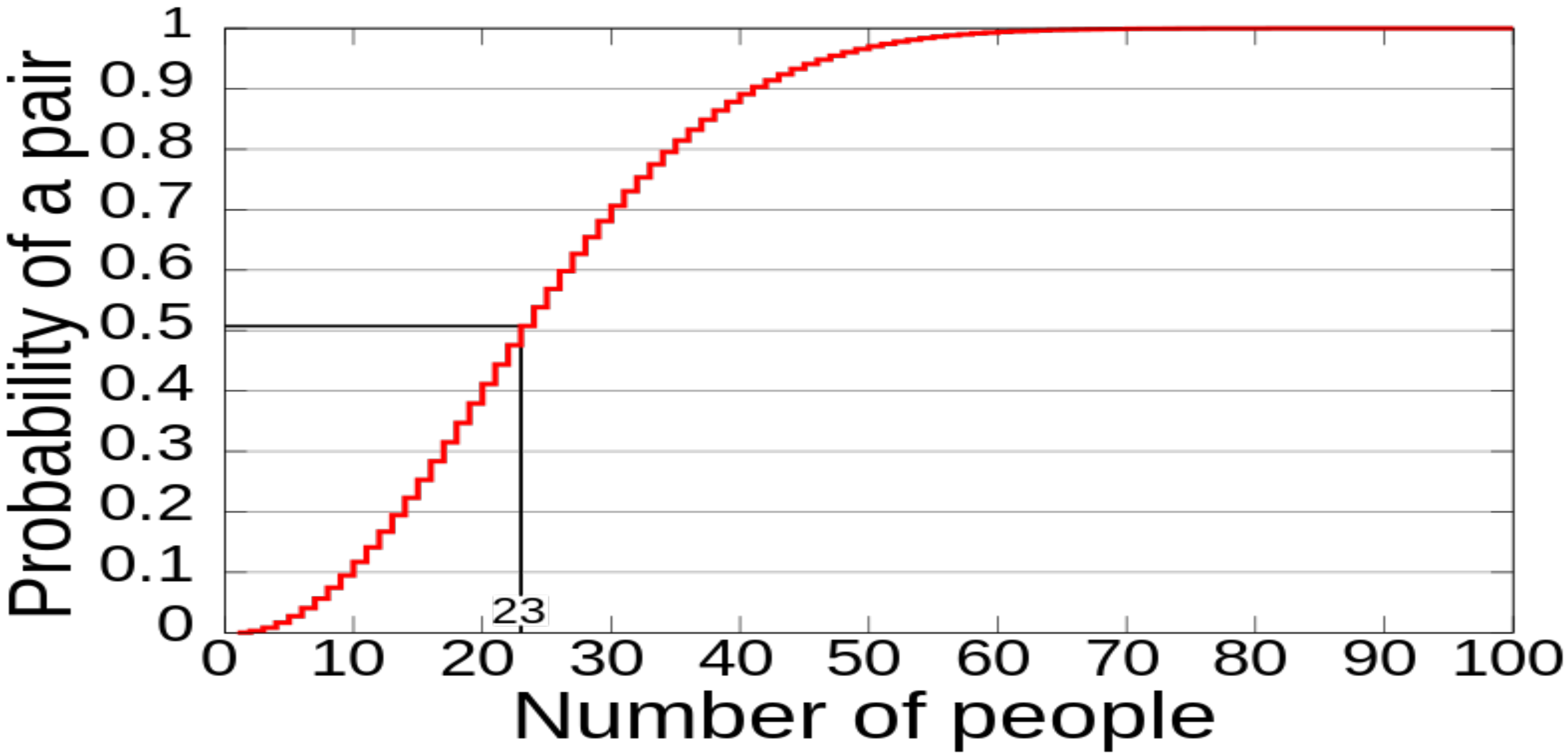
$$\Pr[A'] = 1 \times \left(1 - \frac{1}{365}\right) \times \left(1 - \frac{2}{365}\right) \times \left(1 - \frac{3}{365}\right) \cdots\cdots \left(1 - \frac{Q-1}{365}\right)$$

$$= \prod_{i=1}^{Q-1} \left(1 - \frac{i}{365}\right)$$

$$\Pr[A] = 1 - \prod_{i=1}^{Q-1} \left(1 - \frac{i}{365}\right)$$

# Birthday Paradox

- If there are 23 people in a room, then the probability that two birthdays collide is 1/2



# Birthday Attacks and Message Digests

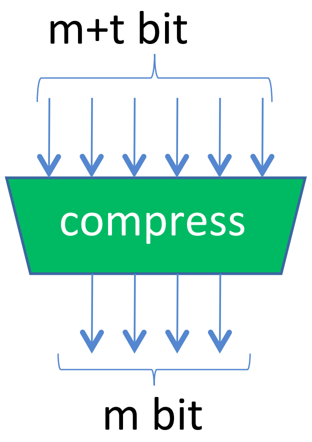
$$Q \approx 1.17\sqrt{M}$$

- If the size of a message digest is 40 bits
- $M = 2^{40}$
- A birthday attack would require $2^{20}$ queries

- Thus to achieve 128 bit security against collision attacks, hashes of length at-least 256 is required

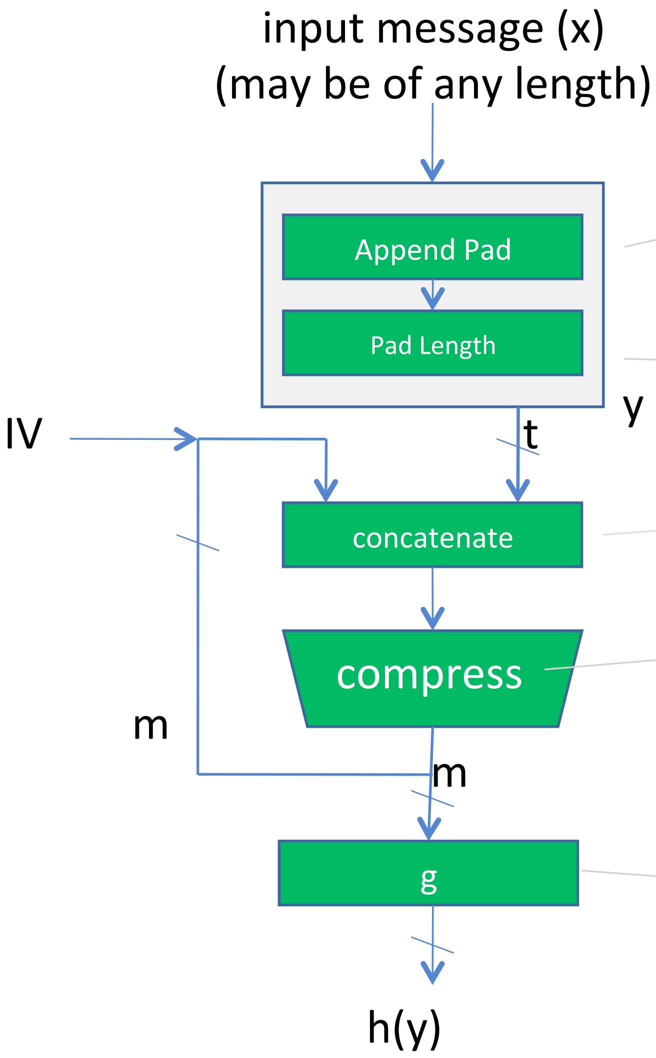
# Iterated Hash Functions

- So far, we've looked at hash functions where the message was picked from a finite set X
- What if the message is of an infinite size?
  - We use an iterated hash function
- The core in an iterated hash function is a function called compress
  - Compress, hashes from m+t bit to m bit




$$compress : \{0,1\}^{m+t} \longrightarrow \{0,1\}^m$$
$$t \geq 1$$

# Iterated Hash Function (given m and t)



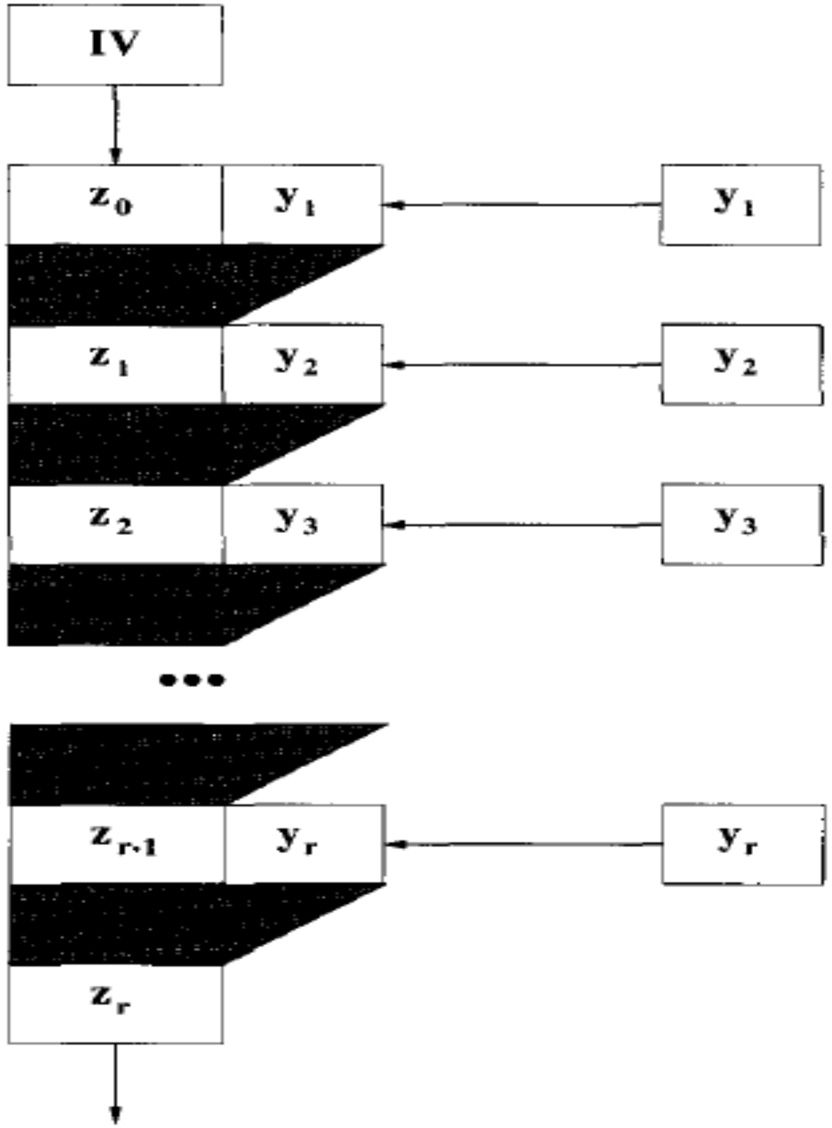

- must be at-least m+t+1 in length

- Input message is padded so that its length is a multiple of t

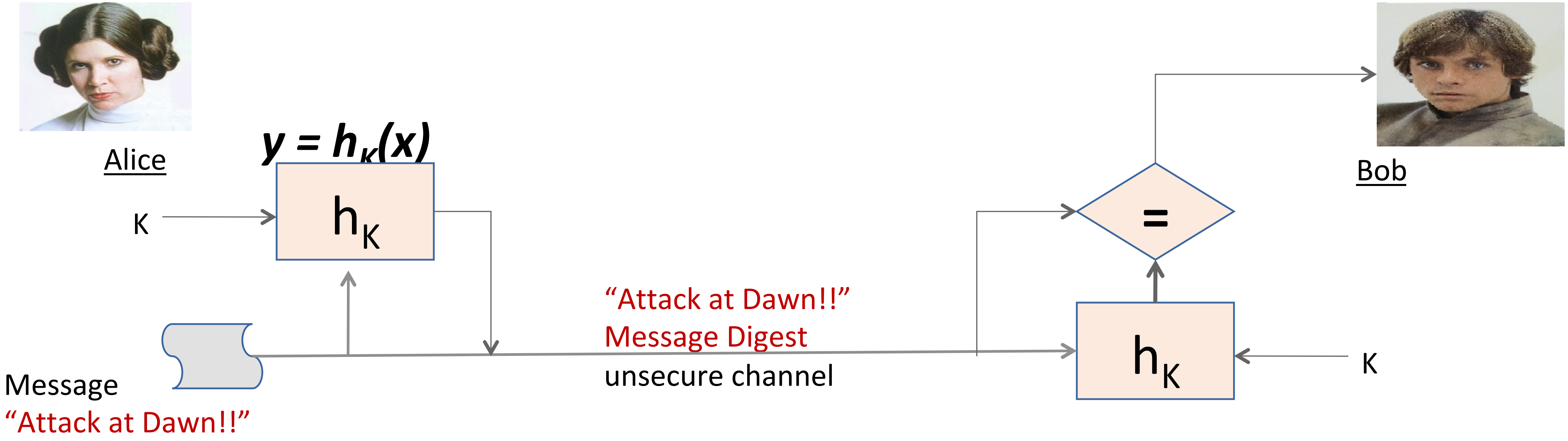
- Number of bits in the pad appended
- Concatenate previous m bit output with next t bit block (IV used only during initialization)
- The compress function is invoked iteratively for each t bit block in the message. For the first operation, an initialization vector is used
- After all t bit blocks are processed, there is a post processing step, and finally the hash is obtained.
  This step is optional.

# Iterated Hash Function (Principle)

- Another perspective



# Message Authentication Codes (Keyed Hash Functions)



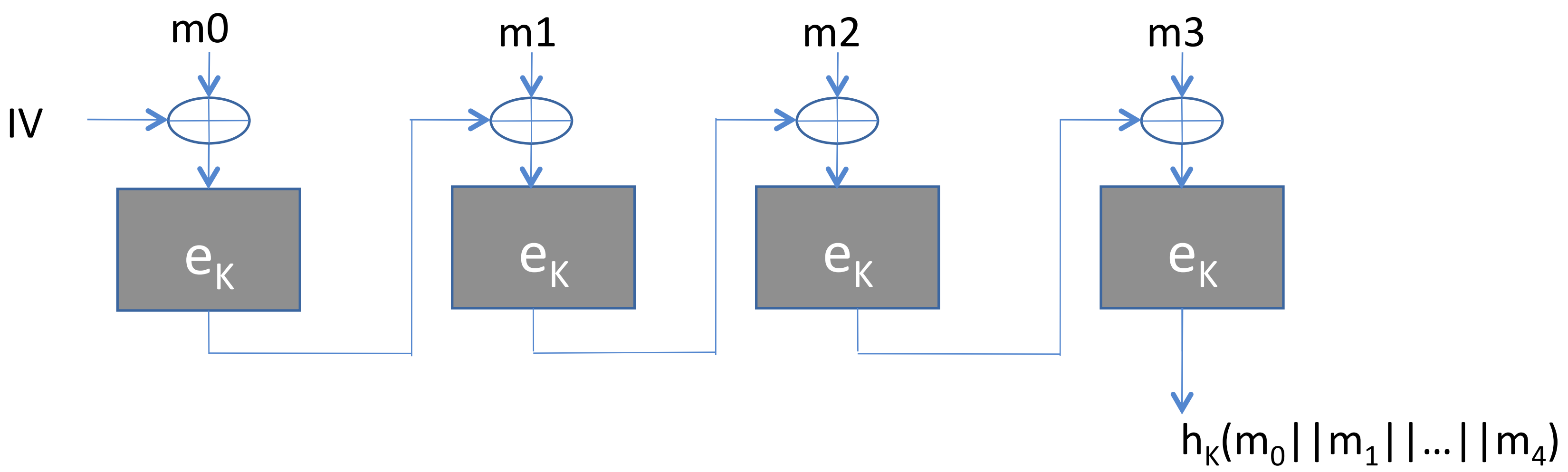

Provides Integrity and Authenticity

Integrity : Messages are not tampered

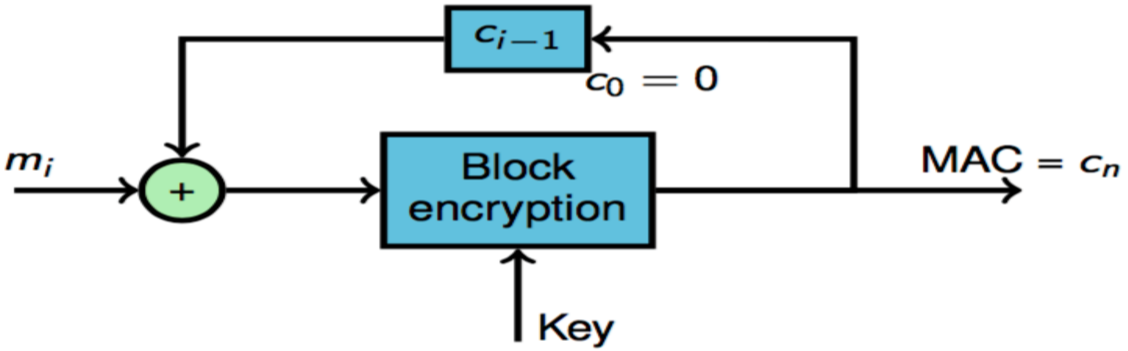
Authenticity : Bob can verify that the message came from Alice

(Does not provide non-repudiation)

# CBC-MAC



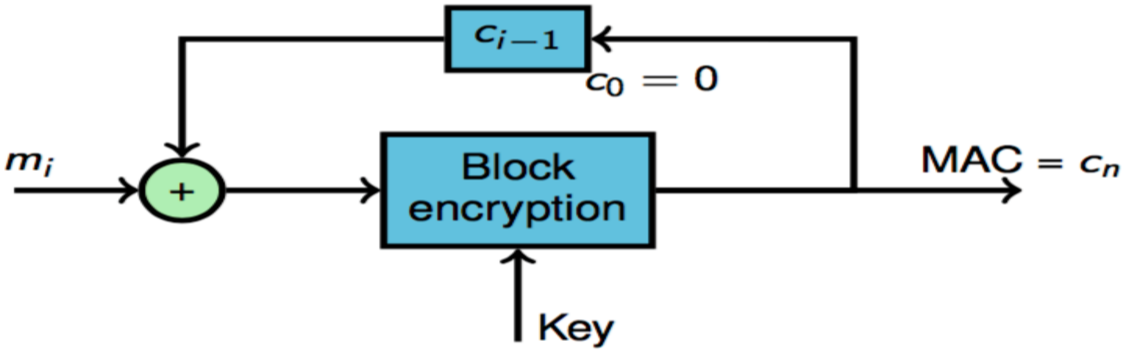
# Birthday Attack on CBC MAC




By Birthday paradox, in $2^{64}$ steps (assuming a 128 bit cipher), a collision will arise.
Let's assume that the collision occurs in the a-th and b-th step.

$$c_a = c_b$$
$$E_k(m_a \oplus c_{a-1}) = E_k(m_b \oplus c_{b-1})$$
$$thus$$
$$m_a \oplus c_{a-1} = m_b \oplus c_{b-1}$$
$$m_a \oplus m_b = c_{a-1} \oplus c_{b-1}$$

# Birthday Attack on CBC MAC




By Birthday paradox, in $2^{64}$ steps (assuming a 128 bit cipher), a collision will arise.
Let's assume that the collision occurs in the a-th and b-th step.

$$
\begin{aligned}
& c_a = c_b \\
& E_k(m_a \oplus c_{a-1}) = E_k(m_b \oplus c_{b-1}) \\
& thus \\
& m_a \oplus c_{a-1} = m_b \oplus c_{b-1} \\
& m_a \oplus m_b = c_{a-1} \oplus c_{b-1}
\end{aligned}
$$

$$
\begin{aligned}
& M_1 = m_1 \| m_2 \| \ldots \| m_i \| \ldots \| m_n \\
& M_2 = m_1 \| m_2 \| \ldots \| (m_i \oplus c_{a-1} \oplus c_{a-2}) \| \ldots \| m_n
\end{aligned}
$$

# HMAC

- FIPS standard for MAC
- Based on unkeyed hash function (SHA-1)

$$HMAC_k(x) = SHA1((K \oplus opad) \,||\, SHA1(K \oplus ipad) \,||\, x))$$

Ipad and opad are predefined constants

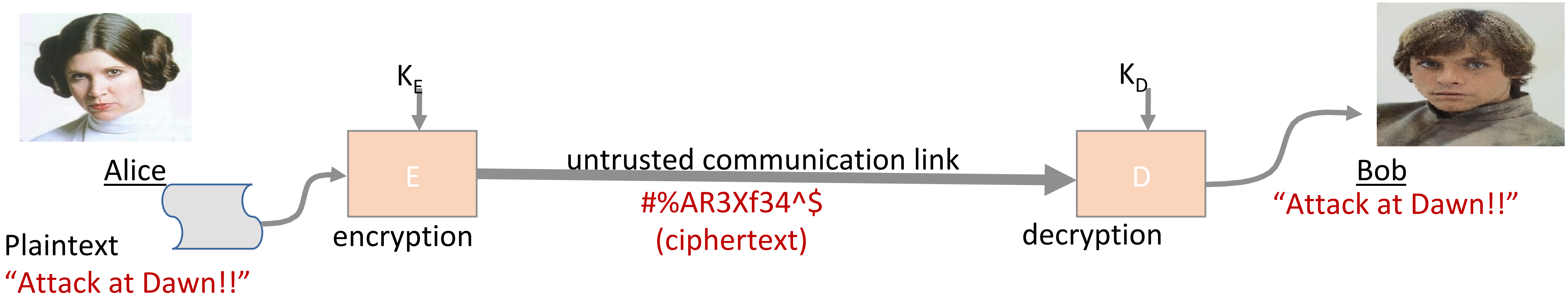
# RSA and Public Key Cryptography

STINSON : chapter 5, 6

# Ciphers

- Symmetric Algorithms
  - Encryption and Decryption use the same key
  - i.e. $K_E = K_D$
  - Examples:
    - Block Ciphers : DES, AES, PRESENT, etc.
    - Stream Ciphers : A5, Grain, etc.
- Asymmetric Algorithms
  - Encryption and Decryption keys are different
  - $K_E \neq K_D$
  - Examples:
    - RSA
    - ECC

# Asymmetric Key Algorithms




**The Key K is a secret**

**Encryption Key $K_E$ not same as decryption key $K_D$**

**$K_E$ known as Bob's public key;**
**$K_D$ is Bob's private key**

**Asymmetric key algorithms based on trapdoor one-way functions**

Advantage : No need of secure key exchange between Alice and Bob

# One Way Functions

- Easy to compute in one direction
- Once done, it is difficult to inverse

**Press to lock**
**(can be easily done)**

**Once locked it is**
**difficult to unlock**
**without a key**

# Trapdoor One Way Function

- One way function with a trapdoor
- Trapdoor is a special function that if possessed can be used to easily invert the one way

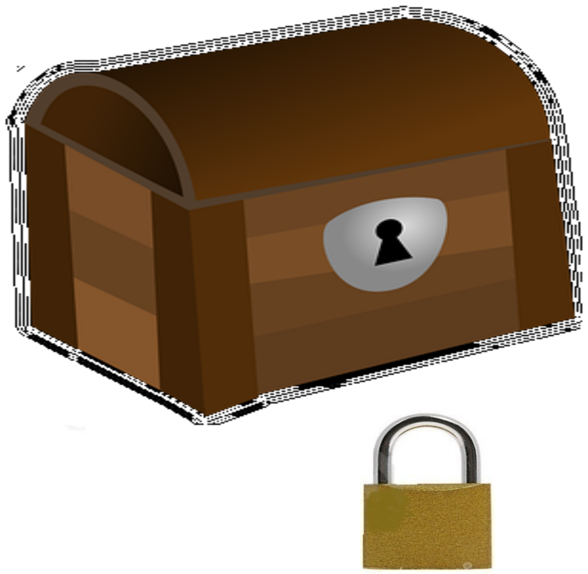
Locked
(difficult to unlock)

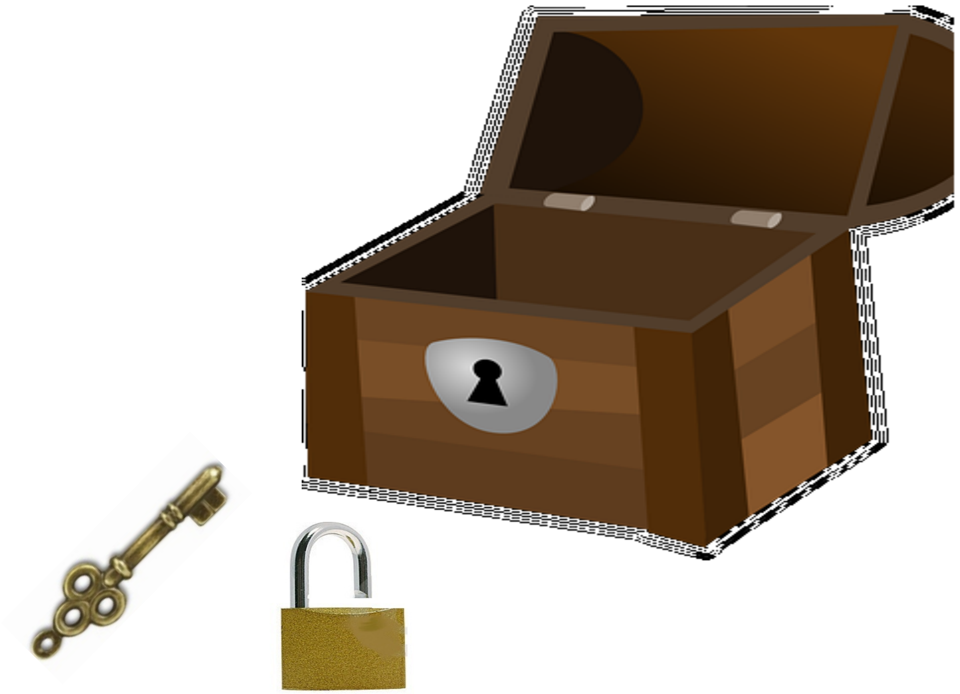
trapdoor

Easily Unlocked

# Public Key Cryptography (An Anology)

- Alice puts message into box and locks it
- Only Bob, who has the key to the lock can open it and read the message





# Mathematical Trapdoor One way functions

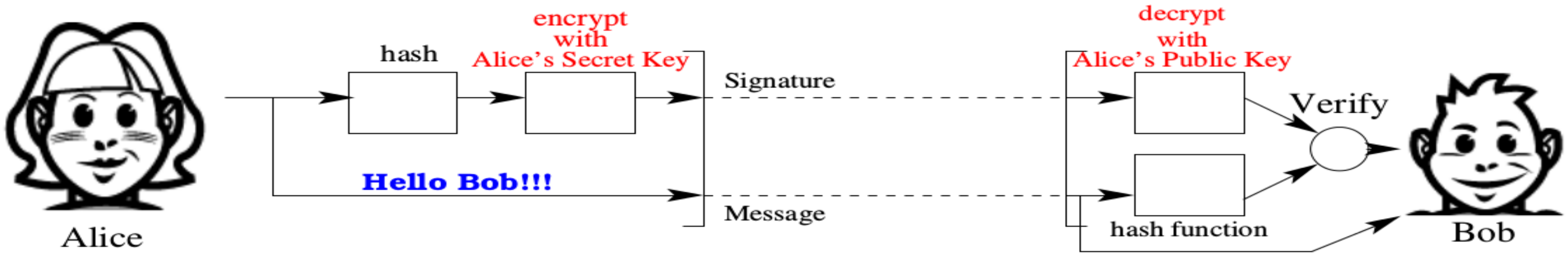
- Examples
  - Integer Factorization (in NP, maybe NP-complete)
    - Given P, Q are two primes
    - and N = P * Q
      - It is easy to compute N
      - However given N it is difficult to factorize into P and Q
    - Used in cryptosystems like RSA
  - Discrete Log Problem (in NP)
    - Consider b and g are elements in a finite group and $b^k = g$, for some k
    - Given b and k it is easy to compute g
    - Given b and g it is difficult to determine k
    - Used in cryptosystems like Diffie-Hellman
    - A variant used in ECC based crypto-systems

# Applications of Public key Cryptography

- **Encryption**
- **Digital Signature :**

  "Is this message really from Alice?"

  - Alice signs by 'encrypting' with private key
  - Anyone can verify signature by 'decrypting' with Alice's public key
  - Why it works?
    - Only Alice, who owns the private key could have signed



# Applications of Public key Cryptography

Diffie-Hellman Key Exchange

- **Key Establishment :** "Alice and Bob want to use a block cipher for encryption. How do they agree upon the secret key"

Alice and Bob agree upon a prime **p** and a generator **g**.
This is public information

choose a secret **a**
compute **A** = $g^a$ **mod p**

choose a secret **b**
compute **B** = $g^b$ **mod p**

**B**

**A**

Compute K = $B^a$ **mod p**

Compute K = $A^b$ **mod p**

$$A^b \bmod p = (g^a)^b \bmod p = (g^b)^a \bmod p = B^a \bmod p$$

# RSA

Shamir, Rivest, Adleman (1977)

# RSA : Key Generation

Bob first creates a pair of keys (one public the other private)

$$1.\ \text{Generate two large primes } p, q \ (p \neq q)$$

$$2.\ \text{Compute } n = p \times q \ \text{ and } \phi(n) = (p-1)(q-1)$$

$$3.\ \text{Choose a random } b \ (1 < b < \phi(n)) \text{ and } \gcd(b, \phi(n)) = 1$$

$$4.\ \text{Compute } a = b^{-1} \bmod(\phi(n))$$

$$Bob's\ public\ key\ is\ (n, b)$$

$$Bob's\ private\ key\ is\ (p, q, a)$$

Given the private key it is easy to compute the public key

Given the public key it is difficult to derive the private key

# RSA Encryption & Decryption

Encryption

$$e_K(x) = y = x^b \bmod n$$
$$where\; x \in Z_n$$

Decryption

$$d_K(x) = y^a \bmod n$$

# RSA Example

1. Take two primes $p = 653$ and $q = 877$
2. $n = 653 \times 877 = 572681$; $\phi(n) = 652 \times 876 = 571152$
3. Choose public key $b = 13$; note that $\gcd(13, 571152) = 1$
4. Private key $a = 395413 = 13^{-1} \bmod 571152$

$Message\ x = 12345$

$encryption: y = 12345^{13} \bmod 572681 \equiv 536754$

$\text{decryption}: x = 536754^{395413} \bmod 572681 \equiv 12345$

# Correctness

$$when \;\; x \in Z_n \;\; and \; \gcd(x, n) = 1$$

Encryption

$$e_K(x) = y = x^b \bmod n$$
$$where \; x \in Z_n$$

Decryption

$$d_K(x) = y^a \bmod n$$

$$\begin{aligned} y^a &\equiv (x^b)^a \bmod n \\ &\equiv (x^{ab}) \bmod n \\ &\equiv (x^{t\phi(n)+1}) \bmod n \\ &\equiv (x^{t\phi(n)} x) \bmod n \\ &\equiv x \end{aligned}$$

$$\begin{aligned} ab &\equiv 1 \bmod \varphi(n) \\ ab - 1 &= t\varphi(n) \\ ab &= t\varphi(n) + 1 \end{aligned}$$

From Fermat's theorem

# Correctness

$$when\ \ x \in Z_n\ \ and\ \gcd(x,n) \neq 1$$

$$Since\ n = pq, \gcd(x,n) = p\ or\ \gcd(x,n) = q$$

$$If$$

$$x \equiv x^{ab} \bmod p$$

$$x \equiv x^{ab} \bmod q$$

$$=\triangleright\ x \equiv x^{ab} \bmod n$$

$$(by\ CRT)$$

$$Assume\ \gcd(n,x) = p$$

$$=\triangleright\ p \mid x =\triangleright\ pk = x$$

$$LHS : x \bmod p \equiv pk \bmod p \equiv 0$$

$$RHS : x^{ab} \bmod p \equiv 0$$

$$\because \gcd(p,x) = p\ it\ implies\ \gcd(q,x) = 1$$

$$\begin{aligned} x^{ab} \bmod q &\equiv x^{t\phi(n)+1} \bmod q \\ &\equiv x^{t\phi(p)\phi(q)+1} \bmod q \\ &\equiv (x^{\phi(q)})^{t\varphi(p)} \cdot x \bmod q \\ &\equiv (1)^{t\varphi(p)} \cdot x \bmod q \equiv x \end{aligned}$$

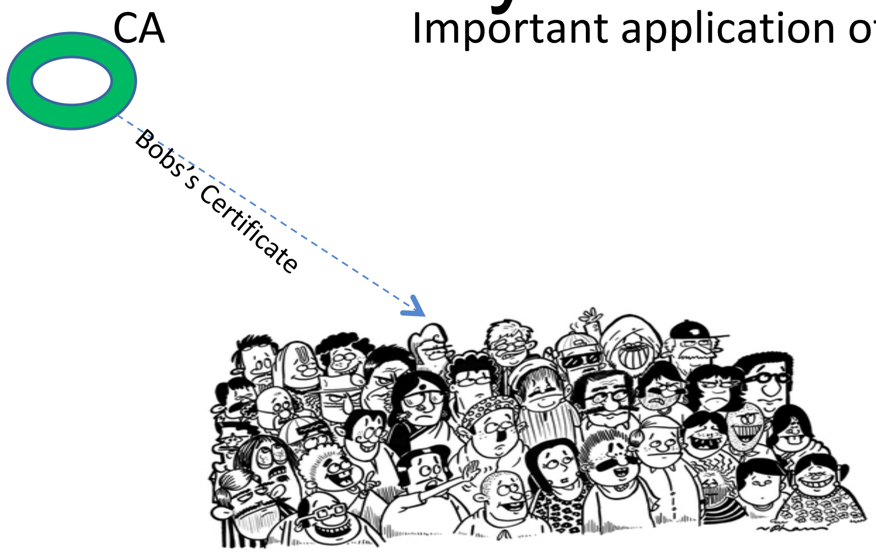
# Signature Schemes

# Digital Signatures

- A token sent along with the message that achieves
  - Authentication
  - Non-repudiation
  - Integrity
- Based on public key cryptography

# Public key Certificates

Important application of digital signatures

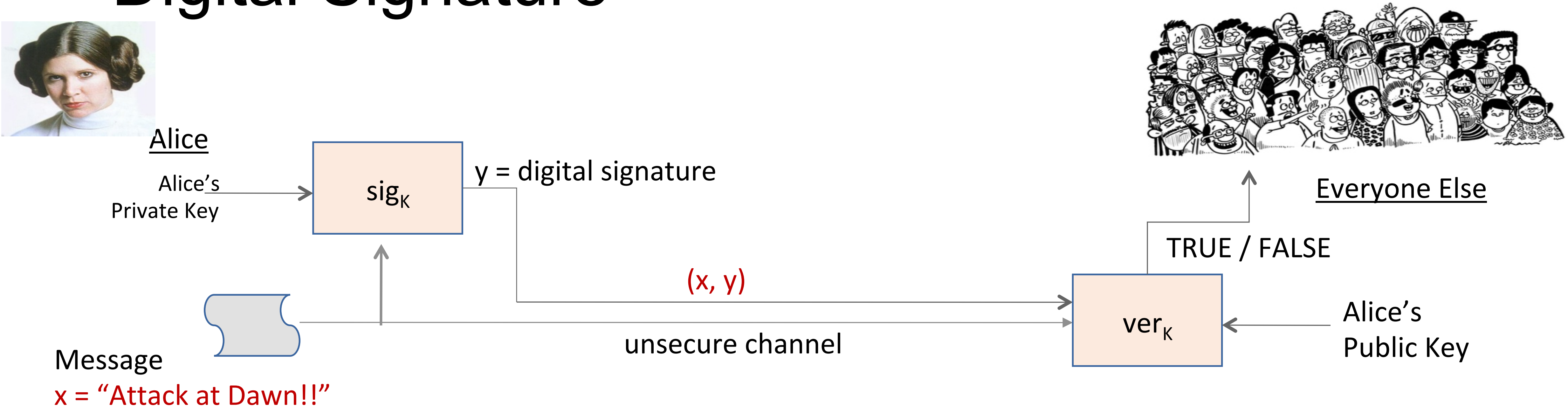



Bob's Certificate{
    Bob's public key in plaintext
    Signature of the certifying authority
    other information
}

To communicate with Bob, Alice gets his public key from a certifying authority (CA)
A trusted authority could be a Government agency, Verisign, etc.

A signature from the CA, ensures that the public key is authentic.

# Digital Signature




**Signing Function**

**y = $sig_a$(x)**

**Input :** Message (x) and Alice's private key
**Output:** Digital Signature of Message

**Verifying Function**

**$ver_b$(x, y)**

**Input :** digital signature, message
**Output :** true or false
true if signature valid
false otherwise

# Digital Signatures (Formally)

**Definition** : A *signature scheme* is a five-tuple $(\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}, \mathcal{V})$, where the following conditions are satisfied:

1. $\mathcal{P}$ is a finite set of possible *messages*
2. $\mathcal{A}$ is a finite set of possible *signatures*
3. $\mathcal{K}$, the *keyspace*, is a finite set of possible *keys*
4. For each $K \in \mathcal{K}$, there is a *signing algorithm* $\mathbf{sig}_K \in \mathcal{S}$ and a corresponding *verification algorithm* $\mathbf{ver}_K \in \mathcal{V}$. Each $\mathbf{sig}_K : \mathcal{P} \to \mathcal{A}$ and $\mathbf{ver}_K : \mathcal{P} \times \mathcal{A} \to \{true, false\}$ are functions such that the following equation is satisfied for every message $x \in \mathcal{P}$ and for every signature $y \in \mathcal{A}$:

$$\mathbf{ver}_K(x, y) = \begin{cases} true & \text{if } y = \mathbf{sig}_K(x) \\ false & \text{if } y \neq \mathbf{sig}_K(x). \end{cases}$$

A pair $(x, y)$ with $x \in \mathcal{P}$ and $y \in \mathcal{A}$ is called a *signed message*.

y

Mallory

Forgery Algorithm

digital signature

(x, y)

unsecure channel

TRUE

Everyone Else

$ver_K$

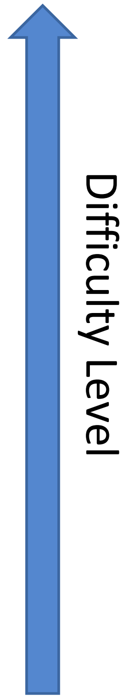
Alice's Public Key

**If Mallory can create a valid digital signature such that $ver_K(x, y)$ = TRUE for a message not previously signed by Alice, then the pair (x, y) forms a forgery**

# Security Models for Digital Signatures

Assumptions | **Goals of Attacker**

- **Total break:**
  Mallory can determine Alice's private key
  (therefore can generate any number of signed messages)
- **Selective forgery:**
  Given a message x, Mallory can determine y, such that (x, y) is a valid signature from Alice
- **Existential forgery:**
  Mallory is able to create y for some x, such that (x, y) is a valid signature from Alice



# Security Models for Digital Signatures

**Assumptions** Goals of Attacker

Weak
(needs a strong attacker)

- **Key-only attack :**
  Mallory only has Alice's public key
  (i.e. only has access to the verification function, *ver*)

- **Known-message attack :**
  Mallory only has a list of messages signed by Alice
  $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), \ldots$..

- **Chosen-message attack :**

  Mallory chooses messages $x_1, x_2, x_3, \ldots\ldots$.. and tricks
  Alice into providing the corresponding signatures $y_1, y_2, y_3$ (resp.)

Strong

# First Attempt making a digital signature (using RSA)

$b, n$ public

$a, p, q$ private

$n = pq;\ a \equiv b^{-1} \bmod \phi(n)$

$sig(x)\{$

$\quad y \equiv x^a \bmod n$

$\quad return\ (x, y)$

$\}$

$(x, y)$ →

$ver(x, y)\{$

$\quad if\,(x \equiv y^b \bmod n)\ \ return\ TRUE$

$\quad else\ return\ \ FALSE$

$\}$

x is the message here
and (x, y) the signature

# A Forgery for the RSA signature (First Forgery)

$b, n$ public

$a, p, q$ private

$n = pq;\; a \equiv b\text{-}1 \bmod \varphi(n)$

$sig(x)\{$

$\quad y \equiv x^a \bmod n$

$\quad return\;\; (x, y)$

$\}$

$(x, y)$

$ver_K(x, y)\{$

$\quad if\,(x \equiv y^b \bmod n)\;\; return\;\; TRUE$

$\quad else\; return\;\; FALSE$

$\}$

$forgery()\{$

$\quad$ select a random $y$

$\quad compute\;\; x \equiv y^b \bmod n$

$\quad return\;\; (x, y)$

$\}$

Key only, existential forgery

# Second Forgery

Suppose Alice creates signatures of two messages $x_1$ and $x_2$

$$y_1 = sig(x_1) \longrightarrow y_1 \equiv x_1^a \bmod n \qquad (x_1, y_1)$$
$$y_2 = sig(x_2) \longrightarrow y_2 \equiv x_2^a \bmod n \qquad (x_2, y_2)$$

Mallory can use the **multiplicative property of RSA** to create a forgery

$$(x_1 x_2 \bmod n, y_1 y_2 \bmod n) \quad \textit{is a forgery}$$
$$y_1 y_2 \equiv x_1^a x_2^a \bmod n$$

Known message, existential forgery

# RSA Digital Signatures

Incorporate a hash function in the scheme to prevent forgery

$$\begin{array}{ll} b, n & \text{public} \\ a, p, q & \text{private} \end{array}$$

$$\begin{aligned} &sig(x)\{ \\ &\quad z = h(x) \\ &\quad y \equiv z^a \bmod n \\ &\quad return\ \ (x, y) \\ &\} \end{aligned}$$

$\xrightarrow{(x, y)}$

$$\begin{aligned} &ver_K(x, y)\{ \\ &\quad z = h(x) \\ &\quad if\,(z \equiv y^b \bmod n)\ \ return\ TRUE \\ &\quad else\ return\ \ FALSE \\ &\} \end{aligned}$$

x is the message here, (x, y) the signature
and h is a hash function

# How does the hash function help?

Preventing the First Forgery

$$
\begin{aligned}
&forgery()\{\\
&\quad \text{select a random } y\\
&\quad compute\ z' \equiv y^b \bmod n\\
&\quad compute\ I^{st}\ preimage:\ x\ st.\ z' = h(x)\\
&\quad return\ (x, y)\\
&\}
\end{aligned}
$$

Forgery becomes equivalent to the first preimage attack on the hash function

# How does the hash function help?

Preventing the Second Forgery

$$(x_1x_2 \bmod n, y_1y_2 \bmod n) \quad \textit{is difficult}$$

$$y_1y_2 \equiv h(x_1)^a h(x_2)^a \bmod n$$

$$\not\equiv x_1{}^a x_2{}^a \bmod n$$

creating such a forgery is unlikely

# How does the hash function help?

Another Forgery prevented

$$
\begin{aligned}
&forgery(x, y)\{\\
&\quad compute\ \ h(x)\\
&\quad compute\ II^{nd}\ preimage:\ \ find\ \ x'\,s.t.\ \ h(x) = h(x')\ \ and\ \ x \neq x'\\
&\quad return\ \ (x', y)\\
&\}
\end{aligned}
$$

Given a valid signature (x,y) find (x',y)

creating such a forgery is equivalent to solving the 2nd preimage problem of the hash functionw