CSE 490K

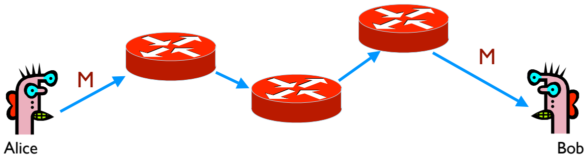
# Cryptography: Symmetric Foundations

Tadayoshi Kohno

Slides derived from Vitaly Shmatikov's

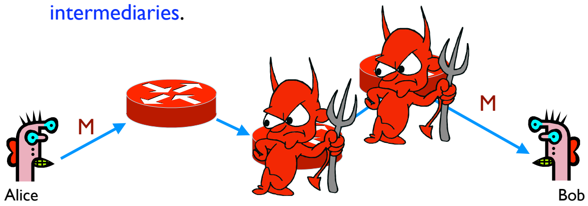
## Basic Problem

Basic Internet model: Communications through untrusted intermediaries.



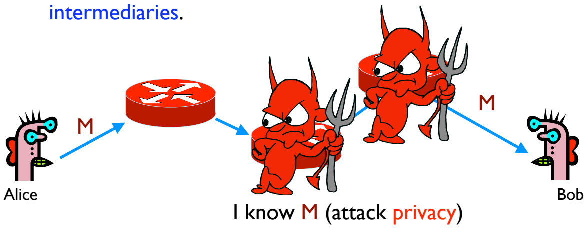
## Basic Problem

Basic Internet model: Communications through untrusted intermediaries.



## Basic Problem

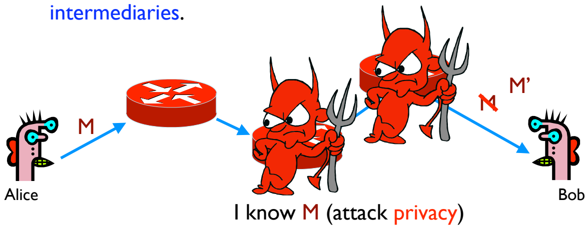
Basic Internet model: Communications through untrusted intermediaries.



I know M (attack privacy)

## Basic Problem

Basic Internet model: Communications through untrusted intermediaries.

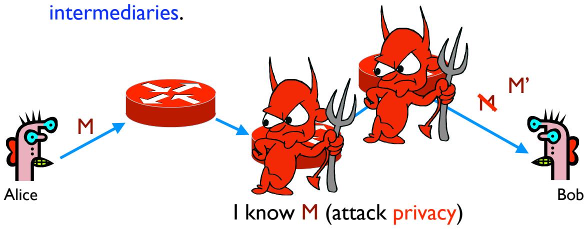


I know M (attack privacy)

I can change M (attack integrity)

## Basic Problem

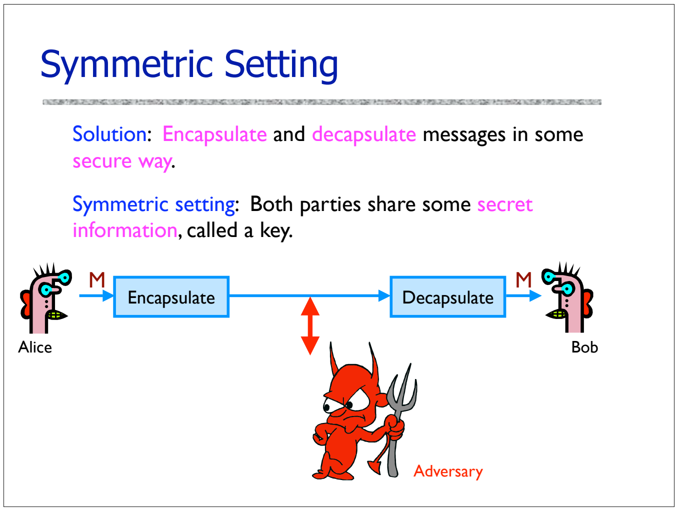
Basic Internet model: Communications through untrusted intermediaries.



I know M (attack privacy)

I can change M (attack integrity)

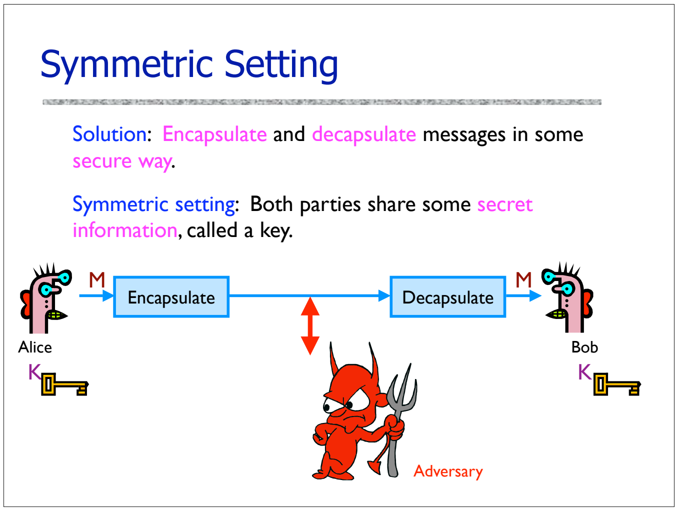
Important for: Secure remote logins, file transfers, web access, ....



## Symmetric Setting

Solution: Encapsulate and decapsulate messages in some secure way.

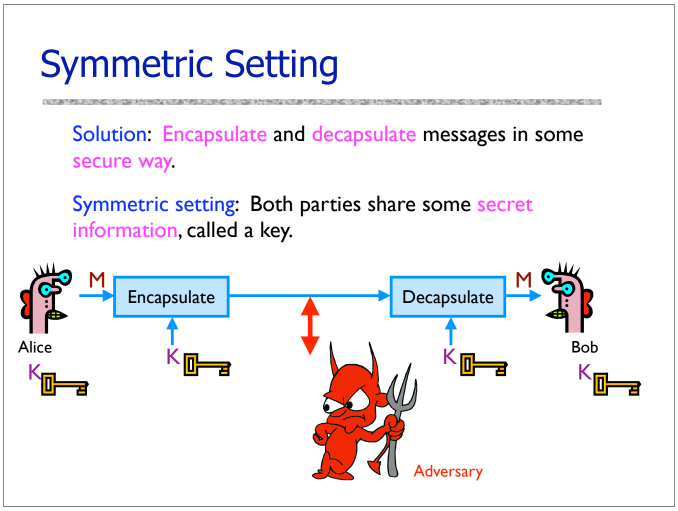
Symmetric setting: Both parties share some secret information, called a key.



## Symmetric Setting

Solution: Encapsulate and decapsulate messages in some secure way.

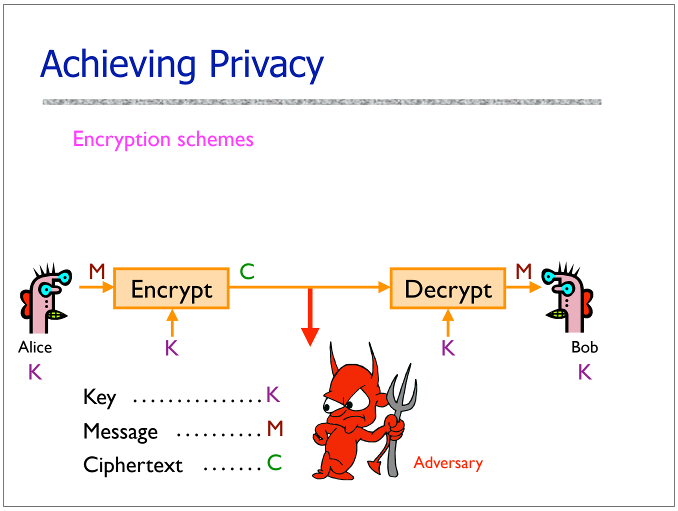
Symmetric setting: Both parties share some secret information, called a key.



## Symmetric Setting

Solution: Encapsulate and decapsulate messages in some secure way.

Symmetric setting: Both parties share some secret information, called a key.



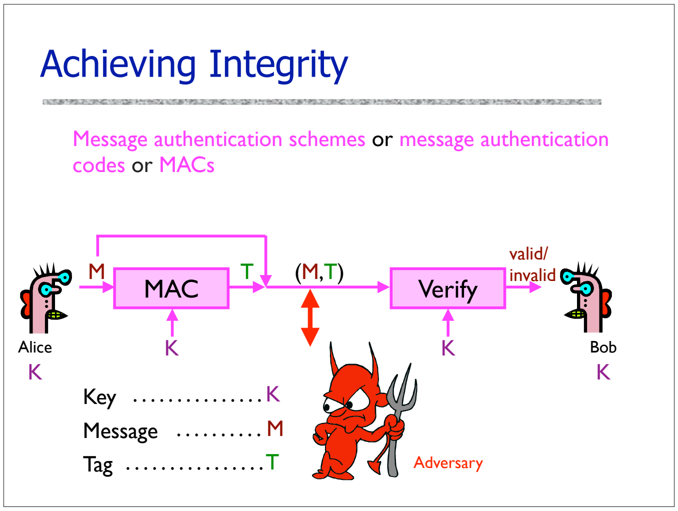
## Achieving Privacy

Encryption schemes

Key ............... K

Message .......... M

Ciphertext ....... C



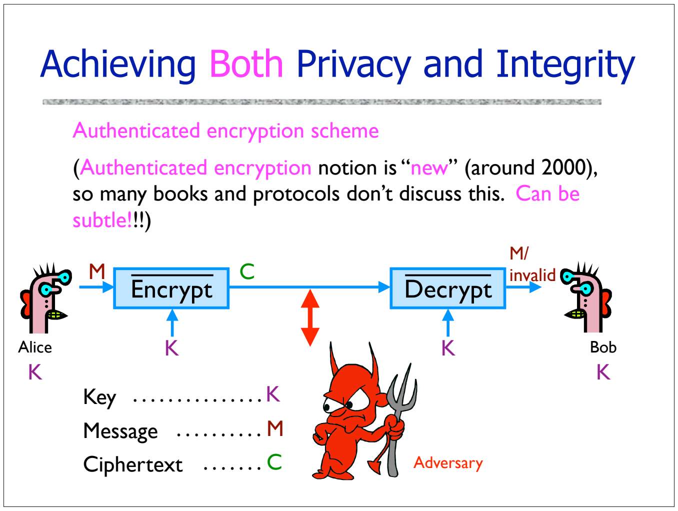
## Achieving Integrity

Message authentication schemes or message authentication codes or MACs

Key ............... K

Message .......... M

Tag ................ T



## Achieving Both Privacy and Integrity

Authenticated encryption scheme

(Authenticated encryption notion is "new" (around 2000), so many books and protocols don't discuss this. Can be subtle!!!)

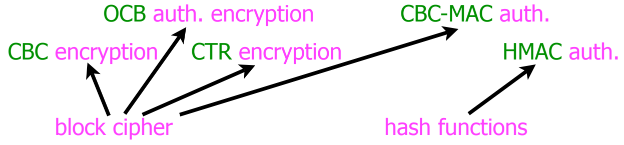
Key ............... K

Message .......... M

Ciphertext ....... C

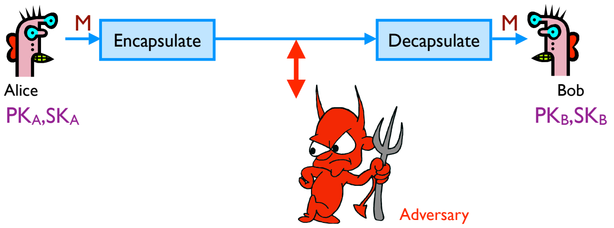
## How this is achieved

- Layered approach:
  - Cryptographic primitives, like block ciphers, stream ciphers, and hash functions
  - Cryptographic protocols, like CBC mode encryption, CTR mode encryption, HMAC message authentication
- Today:
  - Study the above. Basic concepts. Basic pitfalls.



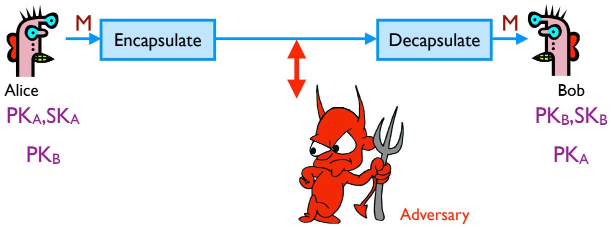
## Asymmetric Setting (NOT today)

Asymmetric setting: Public and Secret keys. (Can help establish shared secret keys K.)



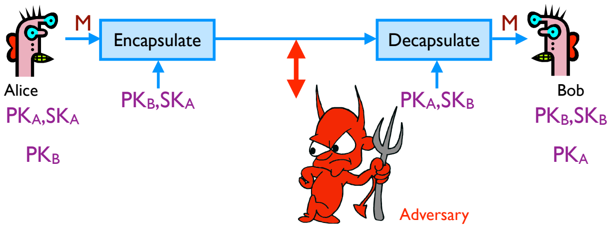
## Asymmetric Setting (NOT today)

Asymmetric setting: Public and Secret keys. (Can help establish shared secret keys K.)

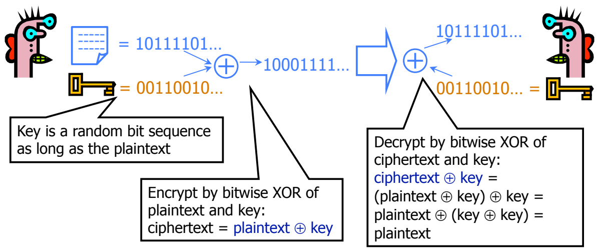


## Asymmetric Setting (NOT today)

Asymmetric setting: Public and Secret keys. (Can help establish shared secret keys K.)



## One-Time Pad



Key is a random bit sequence as long as the plaintext

Encrypt by bitwise XOR of plaintext and key: ciphertext = plaintext ⊕ key

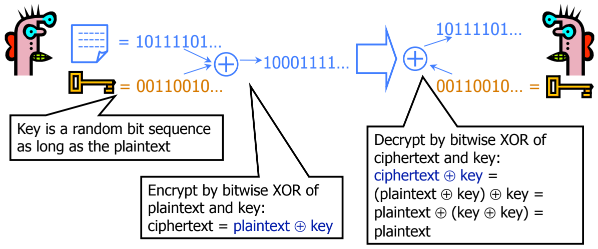
Decrypt by bitwise XOR of ciphertext and key: ciphertext ⊕ key = (plaintext ⊕ key) ⊕ key = plaintext ⊕ (key ⊕ key) = plaintext

Cipher achieves perfect secrecy if and only if there are as many possible keys as possible plaintexts, and every key is equally likely (Claude Shannon)

## Advantages of One-Time Pad

- Easy to compute
  - Encryption and decryption are the same operation
  - Bitwise XOR is very cheap to compute
- As secure as theoretically possible
  - Given a ciphertext, all plaintexts are equally likely, regardless of attacker's computational resources
  - …as long as the key sequence is truly random
    - True randomness is expensive to obtain in large quantities
  - …as long as each key is same length as plaintext
    - But how does the sender communicate the key to receiver?

# Disadvantages



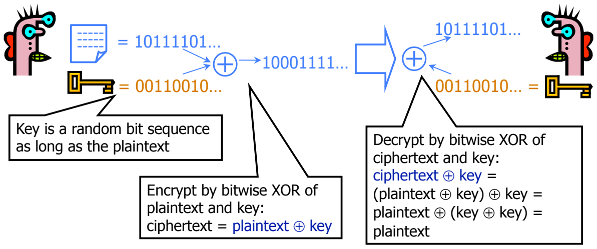
Key is a random bit sequence as long as the plaintext

Encrypt by bitwise XOR of plaintext and key:
ciphertext = plaintext ⊕ key

Decrypt by bitwise XOR of ciphertext and key:
ciphertext ⊕ key =
(plaintext ⊕ key) ⊕ key =
plaintext ⊕ (key ⊕ key) =
plaintext

Disadvantage #1: Keys as long as messages.
Impractical in most scenarios
Still used by intelligence communities

# Disadvantages



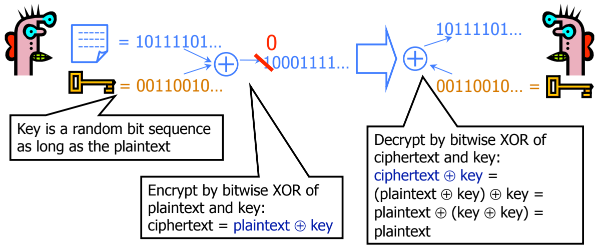
Key is a random bit sequence as long as the plaintext

Encrypt by bitwise XOR of plaintext and key:
ciphertext = plaintext ⊕ key

Decrypt by bitwise XOR of ciphertext and key:
ciphertext ⊕ key =
(plaintext ⊕ key) ⊕ key =
plaintext ⊕ (key ⊕ key) =
plaintext

Disadvantage #2: No integrity protection

# Disadvantages



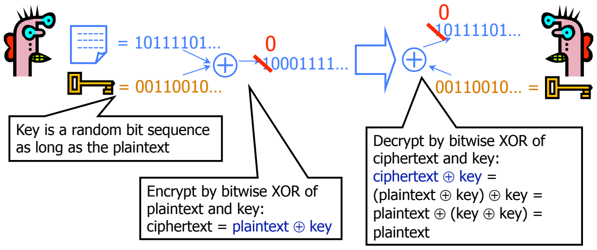
Key is a random bit sequence as long as the plaintext

Encrypt by bitwise XOR of plaintext and key:
ciphertext = plaintext ⊕ key

Decrypt by bitwise XOR of ciphertext and key:
ciphertext ⊕ key =
(plaintext ⊕ key) ⊕ key =
plaintext ⊕ (key ⊕ key) =
plaintext

Disadvantage #2: No integrity protection

# Disadvantages



Key is a random bit sequence as long as the plaintext

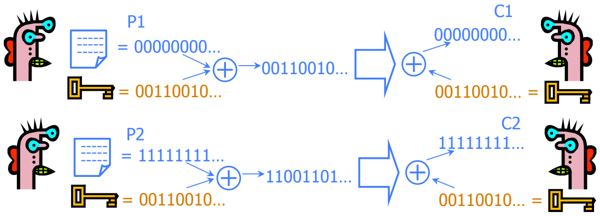
Encrypt by bitwise XOR of plaintext and key:
ciphertext = plaintext ⊕ key

Decrypt by bitwise XOR of ciphertext and key:
ciphertext ⊕ key =
(plaintext ⊕ key) ⊕ key =
plaintext ⊕ (key ⊕ key) =
plaintext

Disadvantage #2: No integrity protection

# Disadvantages

Disadvantage #3: Keys cannot be reused



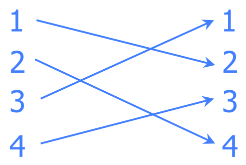
Learn relationship between plaintexts:

$C1 \oplus C2 = (P1 \oplus K) \oplus (P2 \oplus K) = (P1 \oplus P2) \oplus (K \oplus K) = P1 \oplus P2$

# Reducing Keysize

- What do we do when we can't pre-share huge keys?
  - When OTP is unrealistic
- We use special cryptographic primitives
  - Single key can be reused (with some restrictions)
  - But no longer provable secure (in the sense of the OTP)
- Examples: Block ciphers, stream ciphers

## Background: Permutation

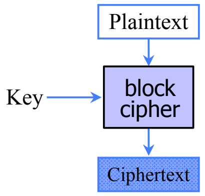


CODE becomes DCEO

- For N-bit input, N! possible permutations
- Idea: split plaintext into blocks, for each block use secret key to pick a permutation, rinse and repeat
  - Without the key, permutation should "look random"

## Block Ciphers

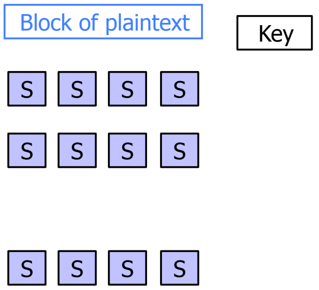
- Operates on a single chunk ("block") of plaintext
  - For example, 64 bits for DES, 128 bits for AES
  - Same key is reused for each block (can use short keys)



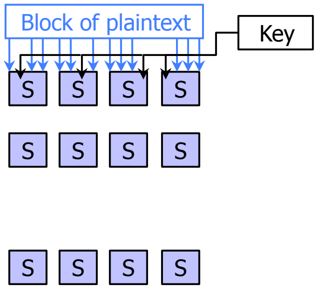
## Block Cipher Security

- Result should look like a random permutation
  - "As if" plaintext bits were randomly shuffled
- Only computational guarantee of secrecy
  - Not impossible to break, just very expensive
    - If there is no efficient algorithm (unproven assumption!), then can only break by brute-force, try-every-possible-key search
  - Time and cost of breaking the cipher exceed the value and/or useful lifetime of protected information

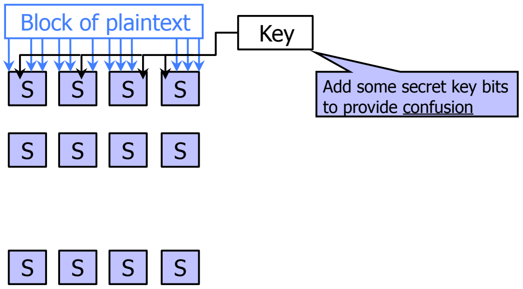
## Block Cipher Operation (Simplified)



## Block Cipher Operation (Simplified)

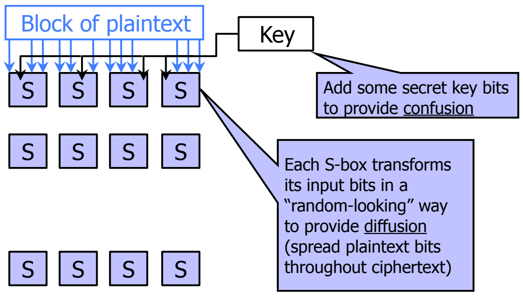


## Block Cipher Operation (Simplified)



Add some secret key bits to provide confusion

# Block Cipher Operation (Simplified)



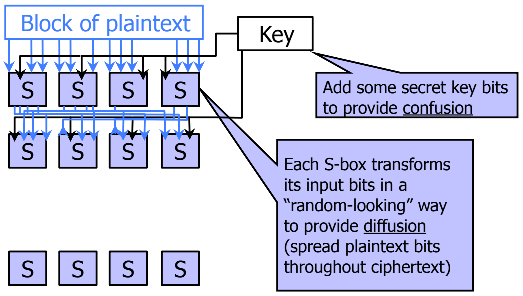
Block of plaintext

Key

Add some secret key bits to provide confusion

Each S-box transforms its input bits in a "random-looking" way to provide diffusion (spread plaintext bits throughout ciphertext)

# Block Cipher Operation (Simplified)



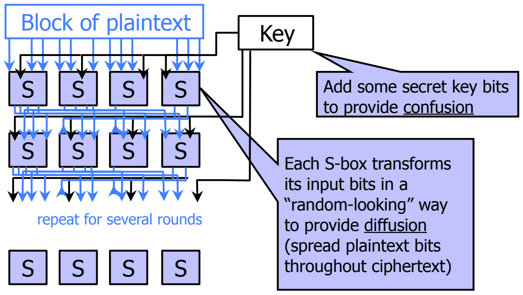
Block of plaintext

Key

Add some secret key bits to provide confusion

Each S-box transforms its input bits in a "random-looking" way to provide diffusion (spread plaintext bits throughout ciphertext)

# Block Cipher Operation (Simplified)



Block of plaintext

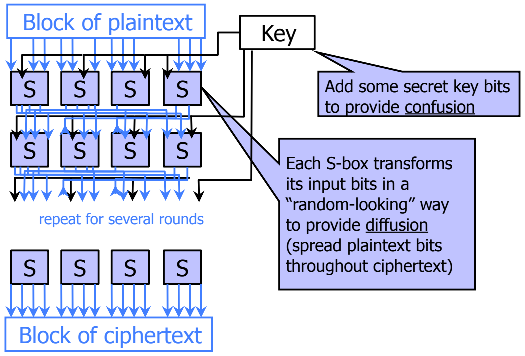
Key

Add some secret key bits to provide confusion

Each S-box transforms its input bits in a "random-looking" way to provide diffusion (spread plaintext bits throughout ciphertext)

repeat for several rounds

# Block Cipher Operation (Simplified)



Block of plaintext

Key

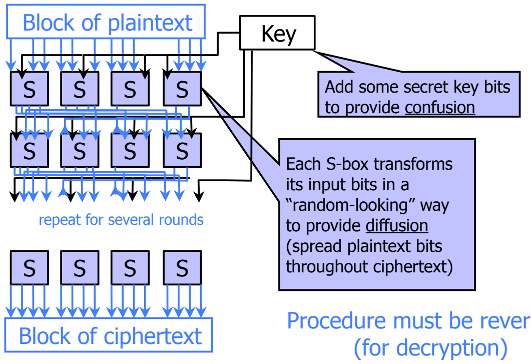
Add some secret key bits to provide confusion

Each S-box transforms its input bits in a "random-looking" way to provide diffusion (spread plaintext bits throughout ciphertext)

repeat for several rounds

Block of ciphertext

# Block Cipher Operation (Simplified)



Block of plaintext

Key

Add some secret key bits to provide confusion

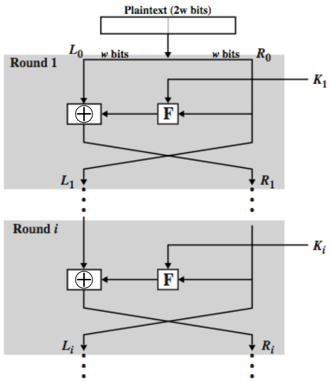
Each S-box transforms its input bits in a "random-looking" way to provide diffusion (spread plaintext bits throughout ciphertext)

repeat for several rounds

Block of ciphertext

Procedure must be reversible
(for decryption)

# Feistel Structure (Stallings Fig 2.2)

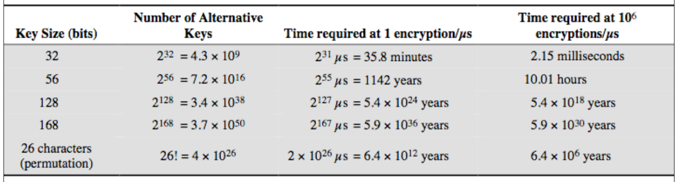


# DES

- Feistel structure
  - "Ladder" structure: split input in half, put one half through the round and XOR with the other half
  - After 3 random rounds, ciphertext indistinguishable from a random permutation (Luby & Rackoff)
- DES: Data Encryption Standard
  - Feistel structure
  - Invented by IBM, issued as federal standard in 1977
  - 64-bit blocks, 56-bit key + 8 bits for parity

# DES and 56 bit keys (Stallings Tab 2.2)

- 56 bit keys are quite short



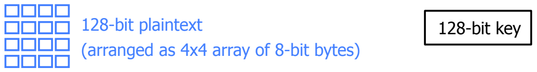
| Key Size (bits) | Number of Alternative Keys | Time required at 1 encryption/$\mu$s | Time required at $10^6$ encryptions/$\mu$s |
|---|---|---|---|
| 32 | $2^{32} = 4.3 \times 10^9$ | $2^{31}\ \mu s = 35.8$ minutes | 2.15 milliseconds |
| 56 | $2^{56} = 7.2 \times 10^{16}$ | $2^{55}\ \mu s = 1142$ years | 10.01 hours |
| 128 | $2^{128} = 3.4 \times 10^{38}$ | $2^{127}\ \mu s = 5.4 \times 10^{24}$ years | $5.4 \times 10^{18}$ years |
| 168 | $2^{168} = 3.7 \times 10^{50}$ | $2^{167}\ \mu s = 5.9 \times 10^{36}$ years | $5.9 \times 10^{30}$ years |
| 26 characters (permutation) | $26! = 4 \times 10^{26}$ | $2 \times 10^{26}\ \mu s = 6.4 \times 10^{12}$ years | $6.4 \times 10^6$ years |

- 1999: EFF DES Crack + distibuted machines
  - < 24 hours to find DES key
- DES ---> 3DES
  - 3DES: DES + inverse DES + DES (with 2 or 3 diff keys)

# Advanced Encryption Standard (AES)

- New federal standard as of 2001
- Based on the Rijndael algorithm
- 128-bit blocks, keys can be 128, 192 or 256 bits
- Unlike DES, does not use Feistel structure
  - The entire block is processed during each round
- Design uses some very nice mathematics

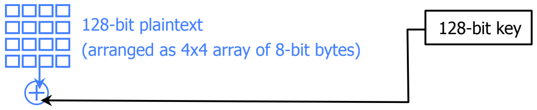
# Basic Structure of Rijndael



128-bit plaintext
(arranged as 4x4 array of 8-bit bytes)

128-bit key

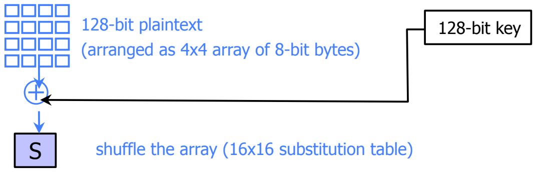
# Basic Structure of Rijndael



128-bit plaintext
(arranged as 4x4 array of 8-bit bytes)

128-bit key

# Basic Structure of Rijndael

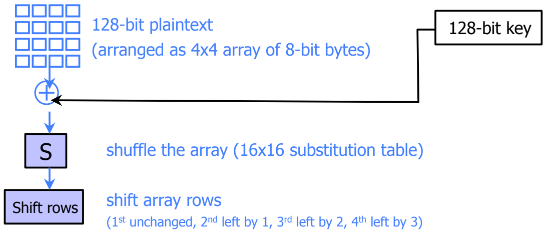


128-bit plaintext
(arranged as 4x4 array of 8-bit bytes)

128-bit key

shuffle the array (16x16 substitution table)

## Basic Structure of Rijndael

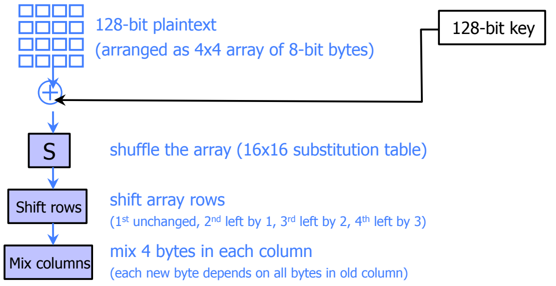


128-bit plaintext
(arranged as 4x4 array of 8-bit bytes)

shuffle the array (16x16 substitution table)

shift array rows
(1st unchanged, 2nd left by 1, 3rd left by 2, 4th left by 3)

## Basic Structure of Rijndael



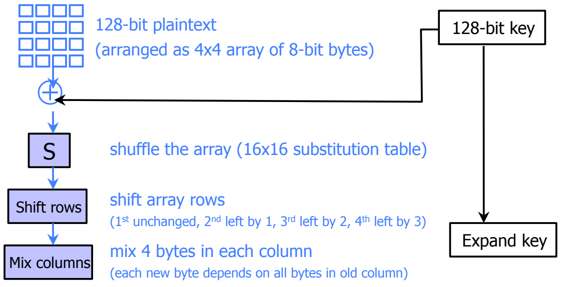
128-bit plaintext
(arranged as 4x4 array of 8-bit bytes)

shuffle the array (16x16 substitution table)

shift array rows
(1st unchanged, 2nd left by 1, 3rd left by 2, 4th left by 3)

mix 4 bytes in each column
(each new byte depends on all bytes in old column)

## Basic Structure of Rijndael



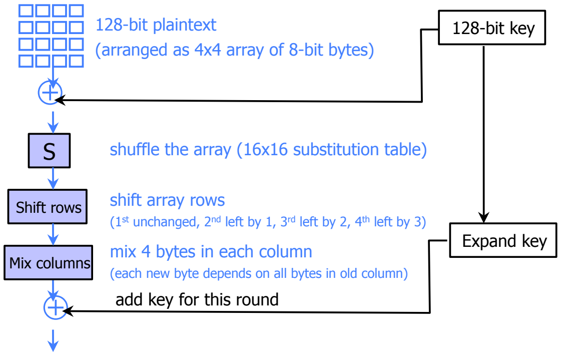
128-bit plaintext
(arranged as 4x4 array of 8-bit bytes)

shuffle the array (16x16 substitution table)

shift array rows
(1st unchanged, 2nd left by 1, 3rd left by 2, 4th left by 3)

mix 4 bytes in each column
(each new byte depends on all bytes in old column)

## Basic Structure of Rijndael



128-bit plaintext
(arranged as 4x4 array of 8-bit bytes)

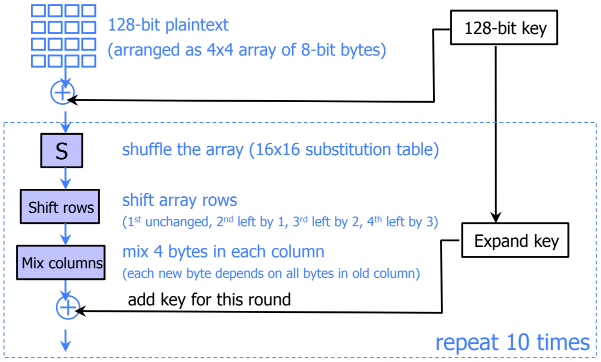
shuffle the array (16x16 substitution table)

shift array rows
(1st unchanged, 2nd left by 1, 3rd left by 2, 4th left by 3)

mix 4 bytes in each column
(each new byte depends on all bytes in old column)

add key for this round

## Basic Structure of Rijndael



128-bit plaintext
(arranged as 4x4 array of 8-bit bytes)

shuffle the array (16x16 substitution table)

shift array rows
(1st unchanged, 2nd left by 1, 3rd left by 2, 4th left by 3)

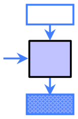
mix 4 bytes in each column
(each new byte depends on all bytes in old column)

add key for this round

repeat 10 times

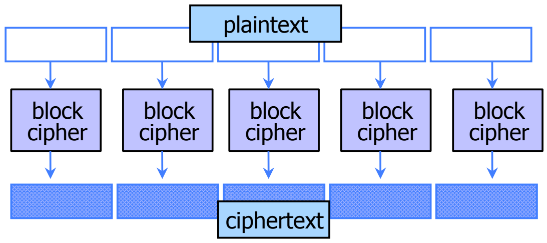
## Encrypting a Large Message

- So, we've got a good block cipher, but our plaintext is larger than 128-bit block size
- Electronic Code Book (ECB) mode
  - Split plaintext into blocks, encrypt each one separately using the block cipher

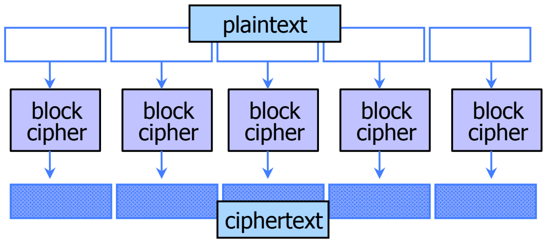


- Cipher Block Chaining (CBC) mode
  - Split plaintext into blocks, XOR each block with the result of encrypting previous blocks
- Counter (CTR) mode
  - Use block cipher to generate keystream, like a stream cipher
- …

## ECB Mode

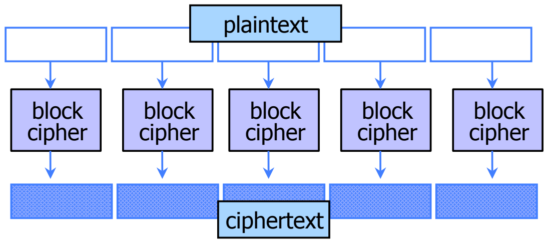


## ECB Mode



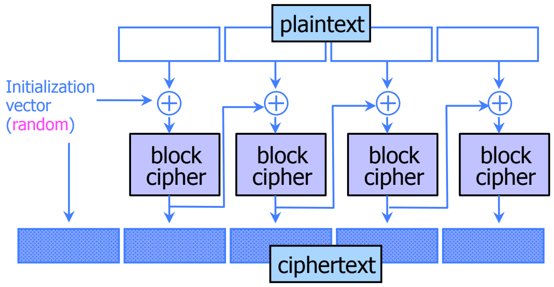
- Identical blocks of plaintext produce identical blocks of ciphertext

## ECB Mode



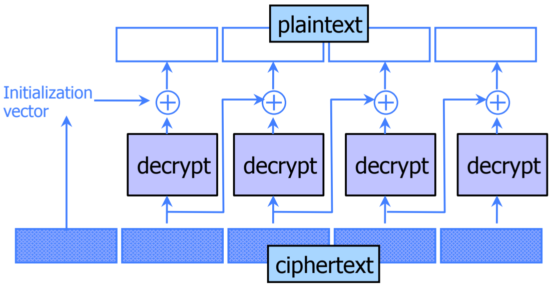
- Identical blocks of plaintext produce identical blocks of ciphertext
- No integrity checks: can mix and match blocks

## CBC Mode: Encryption

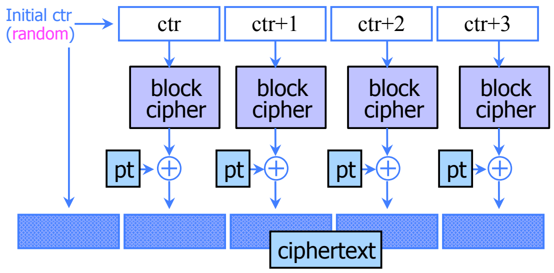


- Identical blocks of plaintext encrypted differently
- Last cipherblock depends on entire plaintext
  - Still does not guarantee integrity

## CBC Mode: Decryption

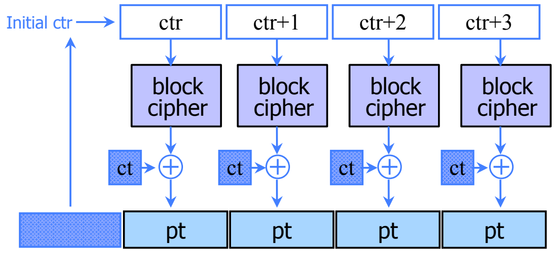


## CTR Mode: Encryption



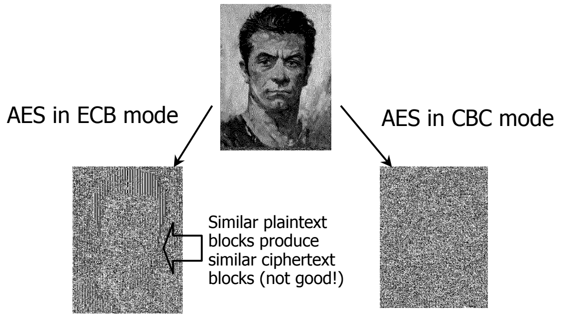
- Identical blocks of plaintext encrypted differently
- Still does not guarantee integrity

## CTR Mode: Decryption



## ECB vs. CBC

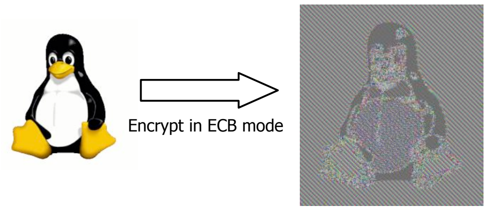
[Picture due to Bart Preneel]



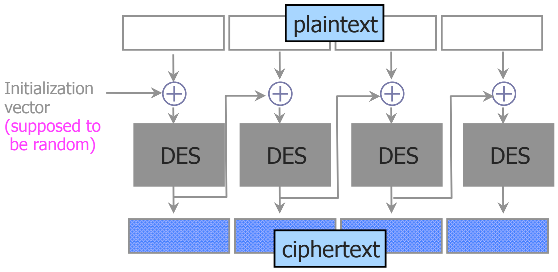
Similar plaintext blocks produce similar ciphertext blocks (not good!)

## Information Leakage in ECB Mode

[Wikipedia]



## CBC and Electronic Voting



Found in the source code for Diebold voting machines:

```
DesCBCEncrypt((des_c_block*)tmp, (des_c_block*)record.m_Data,
              totalSize, DESKEY, NULL, DES_ENCRYPT)
```

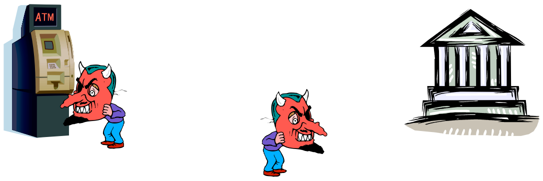
## When Is a Cipher "Secure"?

- Hard to recover the key?
  - What if attacker can learn plaintext without learning the key?
- Hard to recover plaintext from ciphertext?
  - What if attacker learns some bits or some function of bits?
- Fixed mapping from plaintexts to ciphertexts?
  - What if attacker sees two identical ciphertexts and infers that the corresponding plaintexts are identical?
  - Implication: encryption must be randomized or stateful

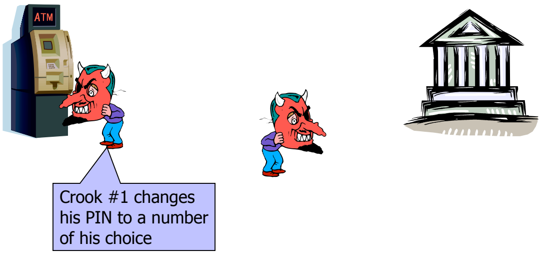
## How Can a Cipher Be Attacked?

- Assume that the attacker knows the encryption algorithm and wants to decrypt some ciphertext
- Main question: what else does attacker know?
  - Depends on the application in which cipher is used!
- Ciphertext-only attack
- Known-plaintext attack (stronger)
  - Knows some plaintext-ciphertext pairs
- Chosen-plaintext attack (even stronger)
  - Can obtain ciphertext for any plaintext of his choice
- Chosen-ciphertext attack (very strong)
  - Can decrypt any ciphertext <u>except</u> the target
  - Sometimes very realistic model

## Chosen-Plaintext Attack

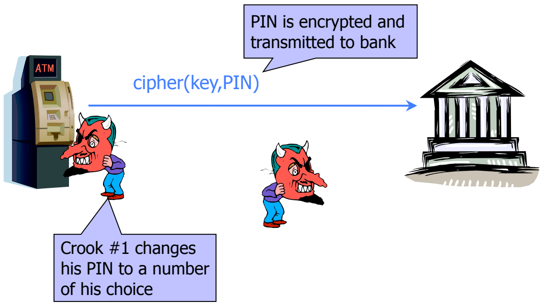


## Chosen-Plaintext Attack



Crook #1 changes his PIN to a number of his choice

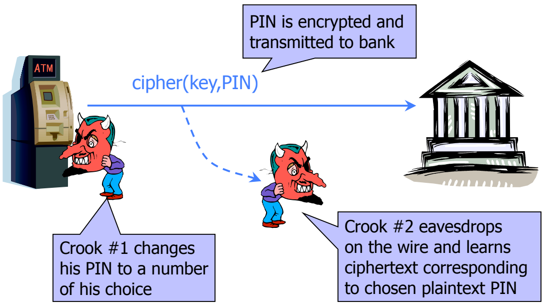
## Chosen-Plaintext Attack



PIN is encrypted and transmitted to bank

Crook #1 changes his PIN to a number of his choice

## Chosen-Plaintext Attack

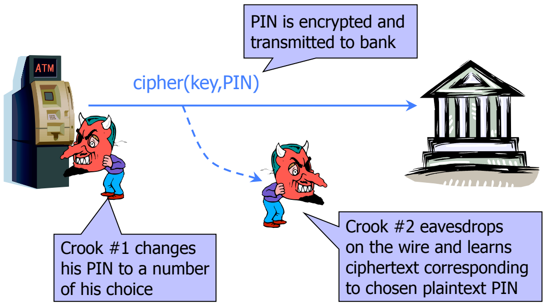


PIN is encrypted and transmitted to bank

Crook #1 changes his PIN to a number of his choice

Crook #2 eavesdrops on the wire and learns ciphertext corresponding to chosen plaintext PIN

## Chosen-Plaintext Attack



PIN is encrypted and transmitted to bank

Crook #1 changes his PIN to a number of his choice

Crook #2 eavesdrops on the wire and learns ciphertext corresponding to chosen plaintext PIN

… repeat for any PIN value

## The Chosen-Plaintext Game

- Attacker does not know the key
- He chooses as many plaintexts as he wants, and learns the corresponding ciphertexts
- When ready, he picks two plaintexts $M_0$ and $M_1$
  - He is even allowed to pick plaintexts for which he previously learned ciphertexts!
- He receives either a ciphertext of $M_0$, or a ciphertext of $M_1$
- He wins if he guesses correctly which one it is

# Defining Security

- Idea: attacker should not be able to learn even a single bit of the encrypted plaintext
- Define Enc($M_0$,$M_1$,b) to be a function that returns encrypted $M_b$

0 or 1

  - Given two plaintexts, Enc returns a ciphertext of one or the other depending on the value of bit b
  - Think of Enc as a magic box that computes ciphertexts on attacker’s demand. He can obtain a ciphertext of any plaintext M by submitting $M_0$=$M_1$=M, or he can try to learn even more by submitting $M_0 \neq M_1$.
- Attacker’s goal is to learn just one bit b

# Why Hide Everything?

- Leaking even a little bit of information about the plaintext can be disastrous
- Electronic voting
  - 2 candidates on the ballot (1 bit to encode the vote)
  - If ciphertext leaks the parity bit of the encrypted plaintext, eavesdropper learns the entire vote
- D-Day: Pas-de-Calais or Normandy?
  - Allies convinced Germans that invasion will take place at Pas-de-Calais
    - Dummy landing craft, feed information to double spies
  - Goal: hide a 1-bit secret
- Also, want a strong definition, that implies others

# Chosen-Plaintext Security

- Consider two experiments (A is the attacker)

| Experiment 0 | Experiment 1 |
|---|---|
| A interacts with Enc(-,-,0)<br>and outputs bit d | A interacts with Enc(-,-,1)<br>and outputs bit d |

  - Identical except for the value of the secret bit
  - d is attacker’s guess of the secret bit
- Attacker’s advantage is defined as

If A “knows” secret bit, he should be able to make his output depend on it

| Prob(A outputs 1 in Exp0) - Prob(A outputs 1 in Exp1)) |

- Encryption scheme is chosen-plaintext secure if this advantage is negligible for any efficient A

# Simple Example

# Simple Example

- Any deterministic, stateless symmetric encryption scheme is insecure

# Simple Example

- Any deterministic, stateless symmetric encryption scheme is insecure
  - Attacker can easily distinguish encryptions of different plaintexts from encryptions of identical plaintexts

## Simple Example

- Any deterministic, stateless symmetric encryption scheme is insecure
  - Attacker can easily distinguish encryptions of different plaintexts from encryptions of identical plaintexts
  - This includes ECB mode of common block ciphers!

## Simple Example

- Any deterministic, stateless symmetric encryption scheme is insecure
  - Attacker can easily distinguish encryptions of different plaintexts from encryptions of identical plaintexts
  - This includes ECB mode of common block ciphers!

Attacker A interacts with Enc(-,-,b)

## Simple Example

- Any deterministic, stateless symmetric encryption scheme is insecure
  - Attacker can easily distinguish encryptions of different plaintexts from encryptions of identical plaintexts
  - This includes ECB mode of common block ciphers!

Attacker A interacts with Enc(-,-,b)

Let X,Y be any two different plaintexts

## Simple Example

- Any deterministic, stateless symmetric encryption scheme is insecure
  - Attacker can easily distinguish encryptions of different plaintexts from encryptions of identical plaintexts
  - This includes ECB mode of common block ciphers!

Attacker A interacts with Enc(-,-,b)

Let X,Y be any two different plaintexts

$C_1 \leftarrow \text{Enc}(X,Y,b)$; $C_2 \leftarrow \text{Enc}(Y,Y,b)$;

## Simple Example

- Any deterministic, stateless symmetric encryption scheme is insecure
  - Attacker can easily distinguish encryptions of different plaintexts from encryptions of identical plaintexts
  - This includes ECB mode of common block ciphers!

Attacker A interacts with Enc(-,-,b)

Let X,Y be any two different plaintexts

$C_1 \leftarrow \text{Enc}(X,Y,b)$; $C_2 \leftarrow \text{Enc}(Y,Y,b)$;

If $C_1=C_2$ then b=1 else say b=0

## Simple Example

- Any deterministic, stateless symmetric encryption scheme is insecure
  - Attacker can easily distinguish encryptions of different plaintexts from encryptions of identical plaintexts
  - This includes ECB mode of common block ciphers!

Attacker A interacts with Enc(-,-,b)

Let X,Y be any two different plaintexts

$C_1 \leftarrow \text{Enc}(X,Y,b)$; $C_2 \leftarrow \text{Enc}(Y,Y,b)$;

If $C_1=C_2$ then b=1 else say b=0

- The advantage of this attacker A is 1

## Simple Example

- Any deterministic, stateless symmetric encryption scheme is insecure
  - Attacker can easily distinguish encryptions of different plaintexts from encryptions of identical plaintexts
  - This includes ECB mode of common block ciphers!

Attacker A interacts with Enc(-,-,b)

Let X,Y be any two different plaintexts

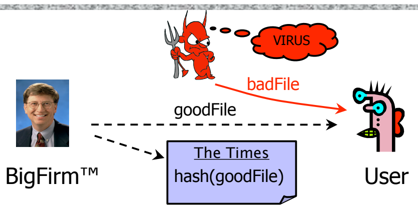
$C_1 \leftarrow Enc(X,Y,b)$; $C_2 \leftarrow Enc(Y,Y,b)$;

If $C_1=C_2$ then b=1 else say b=0

- The advantage of this attacker A is 1

Prob(A outputs 1 if b=0)=0 Prob(A outputs 1 if b=1)=1

## Integrity



Software manufacturer wants to ensure that the executable file is received by users without modification.

It sends out the file to users and publishes its hash in NY Times.

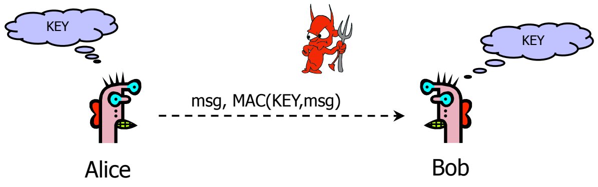
The goal is integrity, not secrecy

Idea: given goodFile and hash(goodFile), very hard to find badFile such that hash(goodFile)=hash(badFile)

## Integrity vs. Secrecy

- Integrity: attacker cannot tamper with message
- Encryption does not always guarantee integrity
  - Intuition: attacker may able to modify message under encryption without learning what it is
    - One-time pad: given key K, encrypt M as M⊕K
    - This guarantees perfect secrecy, but attacker can easily change unknown M under encryption to M⊕M' for any M'
    - Online auction: halve competitor's bid without learning its value
  - This is recognized by industry standards (e.g., PKCS)
    - "RSA encryption is intended primarily to provide confidentiality… It is not intended to provide integrity" (from RSA Labs Bulletin)

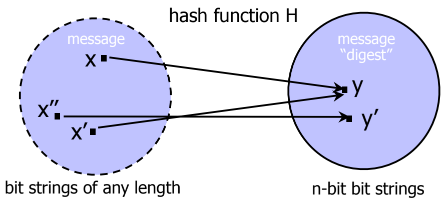
## Motivation: Authentication



Alice wants to make sure that nobody modifies message in transit

Idea: given msg, very hard to compute MAC(KEY,msg) without KEY; very easy with KEY

## Hash Functions: Main Idea



- H is a lossy compression function
  - Collisions: h(x)=h(x') for distinct inputs x, x'
  - Result of hashing should "look random" (make this precise later)
    - Intuition: half of digest bits are "1"; any bit in digest is "1" half the time
- Cryptographic hash function needs a few properties…

## One-Way

- Intuition: hash should be hard to invert
  - "Preimage resistance"
  - Let $h(x')=y\in\{0,1\}^n$ for a random x'
  - Given y, it should be hard to find any x such that h(x)=y
- How hard?
  - Brute-force: try every possible x, see if h(x)=y
  - SHA-1 (common hash function) has 160-bit output
    - Suppose have hardware that'll do $2^{30}$ trials a pop
    - Assuming $2^{34}$ trials per second, can do $2^{89}$ trials per year
    - Will take around $2^{71}$ years to invert SHA-1 on a random image

## Collision Resistance

- Should be hard to find distinct x, x′ such that h(x)=h(x′)
  - Brute-force collision search is only $O(2^{n/2})$, not $O(2^n)$
  - For SHA-1, this means $O(2^{80})$ vs. $O(2^{160})$
- Birthday paradox (informal)
  - Let t be the number of values x,x′,x″… we need to look at before finding the first pair x,x′ s.t. h(x)=h(x′)
  - What is probability of collision for each pair x,x′?
  - How many pairs would we need to look at before finding the first collision?
  - How many pairs x,x′ total?
  - What is t?

## Collision Resistance

- Should be hard to find distinct x, x′ such that h(x)=h(x′)
  - Brute-force collision search is only $O(2^{n/2})$, not $O(2^n)$
  - For SHA-1, this means $O(2^{80})$ vs. $O(2^{160})$
- Birthday paradox (informal)
  - Let t be the number of values x,x′,x″… we need to look at before finding the first pair x,x′ s.t. h(x)=h(x′)
  - What is probability of collision for each pair x,x′? $1/2^n$
  - How many pairs would we need to look at before finding the first collision?
  - How many pairs x,x′ total?
  - What is t?

## Collision Resistance

- Should be hard to find distinct x, x′ such that h(x)=h(x′)
  - Brute-force collision search is only $O(2^{n/2})$, not $O(2^n)$
  - For SHA-1, this means $O(2^{80})$ vs. $O(2^{160})$
- Birthday paradox (informal)
  - Let t be the number of values x,x′,x″… we need to look at before finding the first pair x,x′ s.t. h(x)=h(x′)
  - What is probability of collision for each pair x,x′? $1/2^n$
  - How many pairs would we need to look at before finding the first collision? $O(2^n)$
  - How many pairs x,x′ total?
  - What is t?

## Collision Resistance

- Should be hard to find distinct x, x′ such that h(x)=h(x′)
  - Brute-force collision search is only $O(2^{n/2})$, not $O(2^n)$
  - For SHA-1, this means $O(2^{80})$ vs. $O(2^{160})$
- Birthday paradox (informal)
  - Let t be the number of values x,x′,x″… we need to look at before finding the first pair x,x′ s.t. h(x)=h(x′)
  - What is probability of collision for each pair x,x′? $1/2^n$
  - How many pairs would we need to look at before finding the first collision? $O(2^n)$
  - How many pairs x,x′ total? Choose(2,t)=t(t-1)/2 ~ $O(t^2)$
  - What is t?

## Collision Resistance

- Should be hard to find distinct x, x′ such that h(x)=h(x′)
  - Brute-force collision search is only $O(2^{n/2})$, not $O(2^n)$
  - For SHA-1, this means $O(2^{80})$ vs. $O(2^{160})$
- Birthday paradox (informal)
  - Let t be the number of values x,x′,x″… we need to look at before finding the first pair x,x′ s.t. h(x)=h(x′)
  - What is probability of collision for each pair x,x′? $1/2^n$
  - How many pairs would we need to look at before finding the first collision? $O(2^n)$
  - How many pairs x,x′ total? Choose(2,t)=t(t-1)/2 ~ $O(t^2)$
  - What is t? $2^{n/2}$

## One-Way vs. Collision Resistance

## One-Way vs. Collision Resistance

- One-wayness does not imply collision resistance
  - Suppose g is one-way
  - Define h(x) as g(x′) where x′ is x except the last bit
    - h is one-way (to invert h, must invert g)
    - Collisions for h are easy to find: for any x, h(x0)=h(x1)

## One-Way vs. Collision Resistance

- One-wayness does not imply collision resistance
  - Suppose g is one-way
  - Define h(x) as g(x′) where x′ is x except the last bit
    - h is one-way (to invert h, must invert g)
    - Collisions for h are easy to find: for any x, h(x0)=h(x1)
- Collision resistance does not imply one-wayness
  - Suppose g is collision-resistant
  - Define h(x) to be 0x if x is n-bit long, 1g(x) otherwise
    - Collisions for h are hard to find: if y starts with 0, then there are no collisions, if y starts with 1, then must find collisions in g
    - h is not one way: half of all y's (those whose first bit is 0) are easy to invert (how?); random y is invertible with probab. 1/2

## Weak Collision Resistance

- Given randomly chosen x, hard to find x′ such that h(x)=h(x′)
  - Attacker must find collision for a specific x. By contrast, to break collision resistance, enough to find any collision.
  - Brute-force attack requires $O(2^n)$ time
  - AKA second-preimage collision resistance
- Weak collision resistance does not imply collision resistance

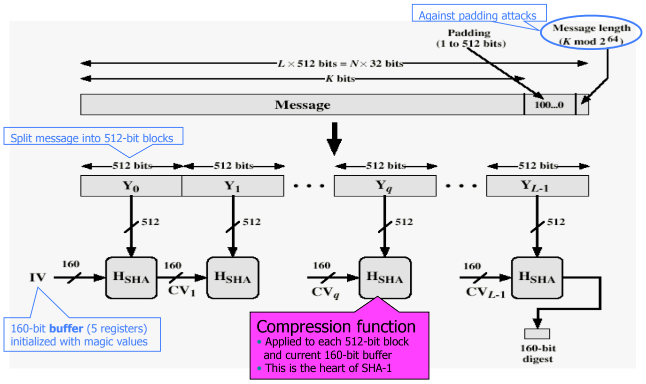
## Which Property Do We Need?

- UNIX passwords stored as hash(password)
  - One-wayness: hard to recover password
- Integrity of software distribution
  - Weak collision resistance
  - But software images are not really random… maybe need full collision resistance
- Auction bidding
  - Alice wants to bid B, sends H(B), later reveals B
  - One-wayness: rival bidders should not recover B
  - Collision resistance: Alice should not be able to change her mind to bid B′ such that H(B)=H(B′)

## Common Hash Functions

- MD5
  - 128-bit output
  - Designed by Ron Rivest, used very widely
  - Collision-resistance broken (summer of 2004)
- RIPEMD-160
  - 160-bit variant of MD5
- SHA-1 (Secure Hash Algorithm)
  - 160-bit output
  - US government (NIST) standard as of 1993-95
    - Also the hash algorithm for Digital Signature Standard (DSS)

## Basic Structure of SHA-1



Against padding attacks

Split message into 512-bit blocks

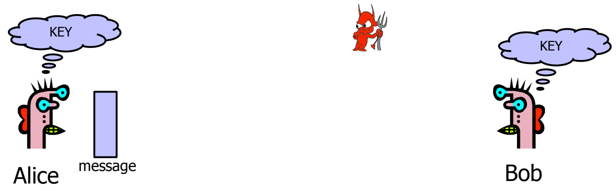
160-bit **buffer** (5 registers) initialized with magic values

Compression function
- Applied to each 512-bit block and current 160-bit buffer
- This is the heart of SHA-1

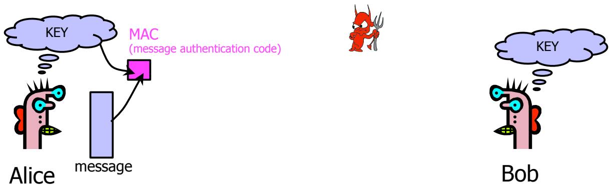
## How Strong Is SHA-1?

- Every bit of output depends on every bit of input
  - Very important property for collision-resistance
- Brute-force inversion requires $2^{160}$ ops, birthday attack on collision resistance requires $2^{80}$ ops
- Some very recent weaknesses (2005)
  - Collisions can be found in $2^{63}$ ops

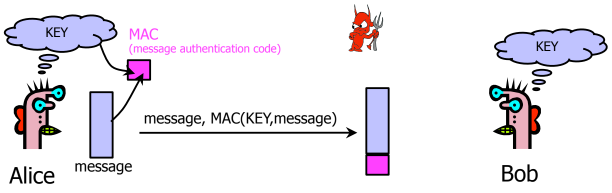
## Authentication Without Encryption



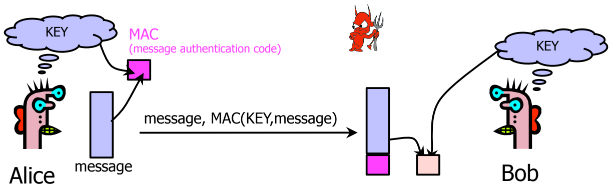
## Authentication Without Encryption



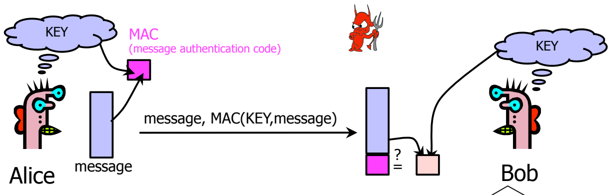
## Authentication Without Encryption



## Authentication Without Encryption

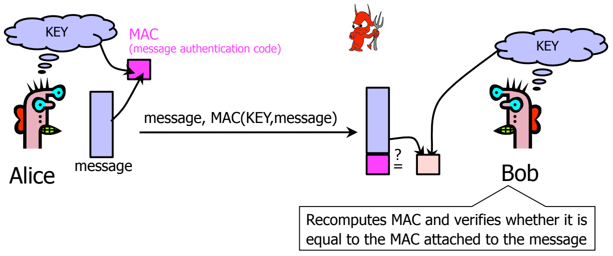


## Authentication Without Encryption



Recomputes MAC and verifies whether it is equal to the MAC attached to the message

# Authentication Without Encryption



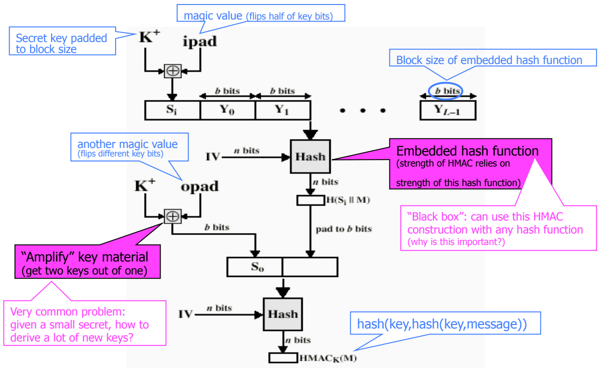
Recomputes MAC and verifies whether it is equal to the MAC attached to the message

Integrity and authentication: only someone who knows KEY can compute MAC for a given message

# HMAC

- Construct MAC by applying a cryptographic hash function to message and key
  - Could also use encryption instead of hashing, but…
  - Hashing is faster than encryption in software
  - Library code for hash functions widely available
  - Can easily replace one hash function with another
  - There used to be US export restrictions on encryption
- Invented by Bellare, Canetti, and Krawczyk (1996)
  - HMAC strength established by cryptographic analysis
- Mandatory for IP security, also used in SSL/TLS

# Structure of HMAC



magic value (flips half of key bits)

Secret key padded to block size

Block size of embedded hash function

another magic value (flips different key bits)

Embedded hash function (strength of HMAC relies on strength of this hash function)

"Black box": can use this HMAC construction with any hash function (why is this important?)

"Amplify" key material (get two keys out of one)

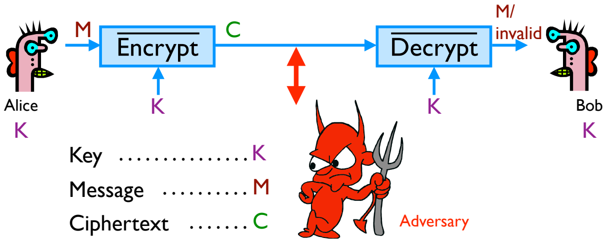
Very common problem: given a small secret, how to derive a lot of new keys?

hash(key,hash(key,message))

# Achieving Both Privacy and Integrity

Authenticated encryption scheme

Recall: Often desire both privacy and integrity. (For SSH, SSL, IPsec, etc.)



# Some subtleties! Encrypt-and-MAC

Natural approach for authenticated encryption: Combine an encryption scheme and a MAC.

# Some subtleties! Encrypt-and-MAC

Natural approach for authenticated encryption: Combine an encryption scheme and a MAC.

$\bar{E}_{Ke,Km}$ $\bar{D}_{Ke,Km}$

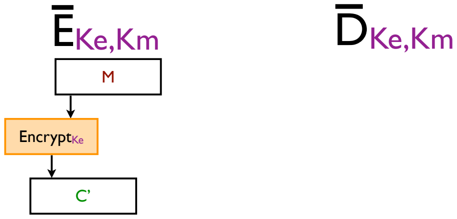
## Some subtleties! Encrypt-and-MAC

Natural approach for authenticated encryption: Combine an encryption scheme and a MAC.

$\bar{E}_{Ke,Km}$
M
$\bar{D}_{Ke,Km}$

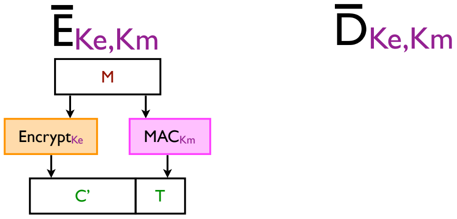
## Some subtleties! Encrypt-and-MAC

Natural approach for authenticated encryption: Combine an encryption scheme and a MAC.



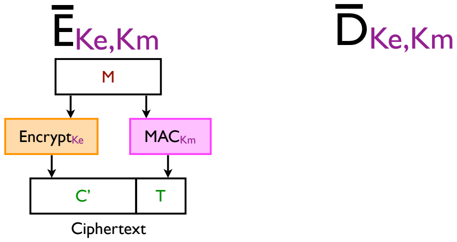
## Some subtleties! Encrypt-and-MAC

Natural approach for authenticated encryption: Combine an encryption scheme and a MAC.



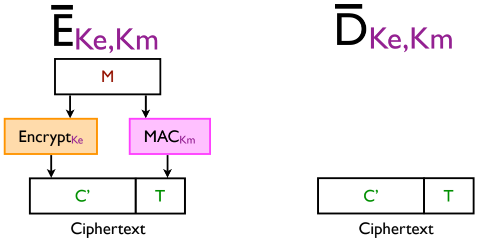
## Some subtleties! Encrypt-and-MAC

Natural approach for authenticated encryption: Combine an encryption scheme and a MAC.



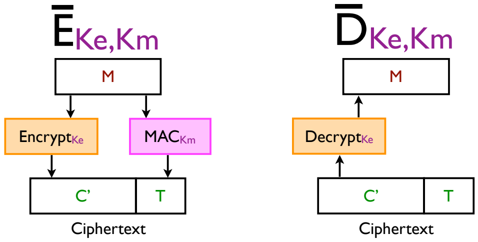
## Some subtleties! Encrypt-and-MAC

Natural approach for authenticated encryption: Combine an encryption scheme and a MAC.



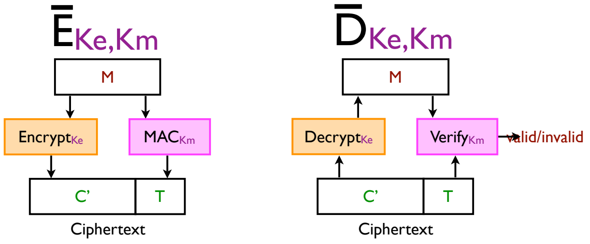
## Some subtleties! Encrypt-and-MAC

Natural approach for authenticated encryption: Combine an encryption scheme and a MAC.



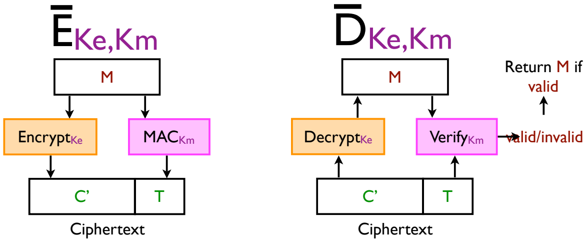
# Some subtleties! Encrypt-and-MAC

Natural approach for authenticated encryption: Combine an encryption scheme and a MAC.



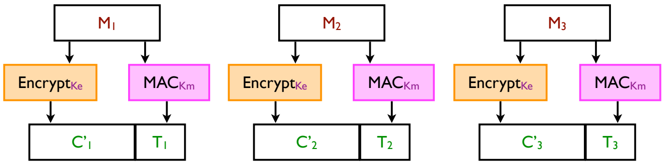
# Some subtleties! Encrypt-and-MAC

Natural approach for authenticated encryption: Combine an encryption scheme and a MAC.



# But insecure! [BN, Kra]

Assume Alice sends messages:



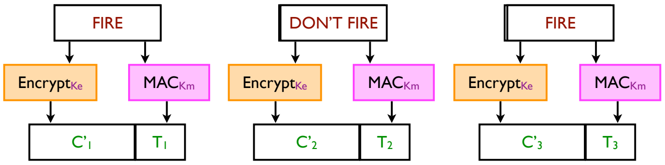
If $T_i = T_j$ then $M_i = M_j$

Adversary learns whether two plaintexts are equal.

Especially problematic when $M_1$, $M_2$, ... take on only a small number of possible values.

# But insecure! [BN, Kra]

Assume Alice sends messages:



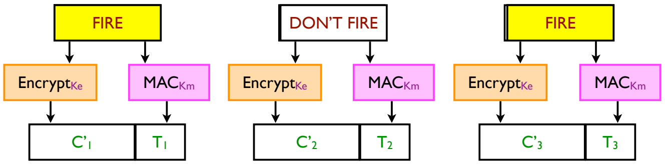
If $T_i = T_j$ then $M_i = M_j$

Adversary learns whether two plaintexts are equal.

Especially problematic when $M_1$, $M_2$, ... take on only a small number of possible values.

# But insecure! [BN, Kra]

Assume Alice sends messages:



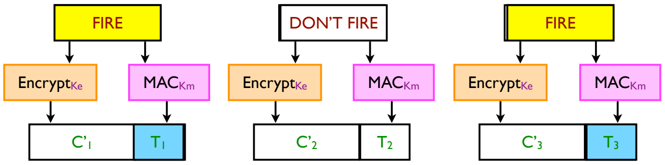
If $T_i = T_j$ then $M_i = M_j$

Adversary learns whether two plaintexts are equal.

Especially problematic when $M_1$, $M_2$, ... take on only a small number of possible values.

# But insecure! [BN, Kra]

Assume Alice sends messages:



If $T_i = T_j$ then $M_i = M_j$

Adversary learns whether two plaintexts are equal.

Especially problematic when $M_1$, $M_2$, ... take on only a small number of possible values.

The Secure Shell (SSH) protocol is designed to provide:

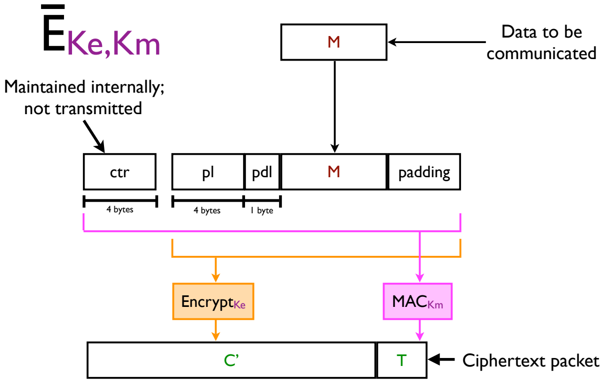
- Secure remote logins.
- Secure file transfers.

Where security includes:

- Protecting the privacy of users' data.
- Protecting the integrity of users' data.

OpenSSH is included in the default installations of OS X and many Linux distributions.

# Authenticated encryption in SSH



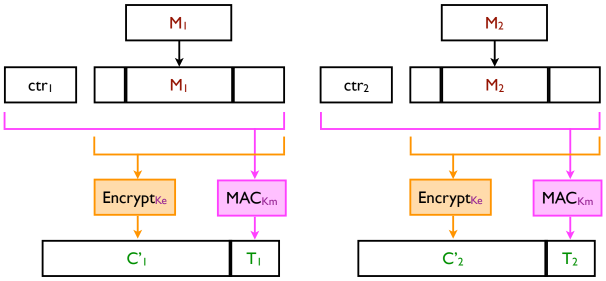
# What's different about SSH?

Assume Alice sends messages $M_1$ and $M_2$ that are the same.

Then the tags $T_1$ and $T_2$ will be different with high probability.

# What's different about SSH?

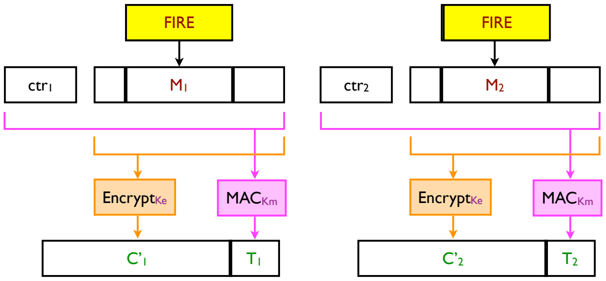
Assume Alice sends messages $M_1$ and $M_2$ that are the same.



Then the tags $T_1$ and $T_2$ will be different with high probability.

# What's different about SSH?

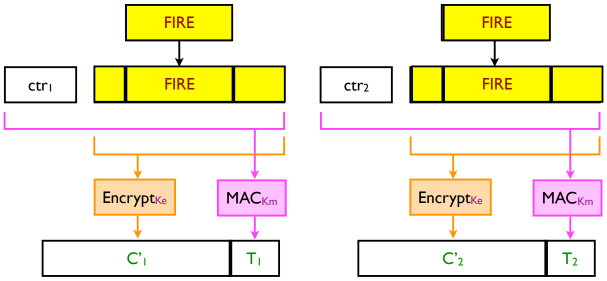
Assume Alice sends messages $M_1$ and $M_2$ that are the same.



Then the tags $T_1$ and $T_2$ will be different with high probability.

# What's different about SSH?

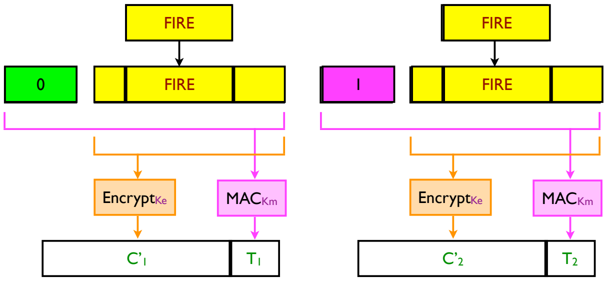
Assume Alice sends messages $M_1$ and $M_2$ that are the same.



Then the tags $T_1$ and $T_2$ will be different with high probability.

# What's different about SSH?

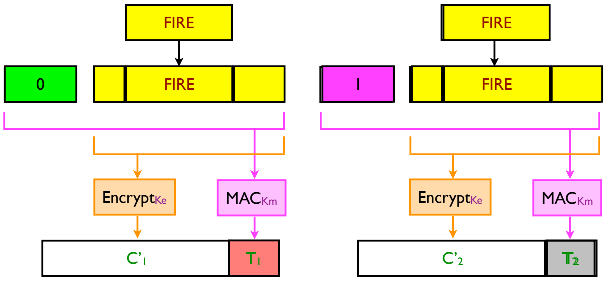
Assume Alice sends messages $M_1$ and $M_2$ that are the same.



Then the tags $T_1$ and $T_2$ will be different with high probability.

# What's different about SSH?

Assume Alice sends messages $M_1$ and $M_2$ that are the same.



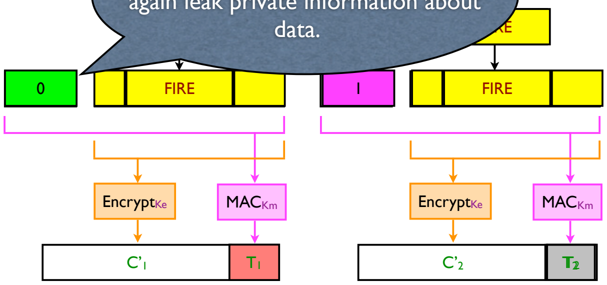
Then the tags $T_1$ and $T_2$ will be different with high probability.

# What's different about SSH?

Assume A

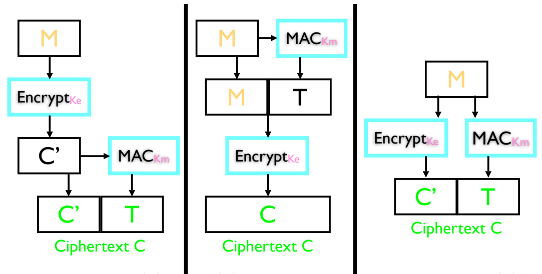
ame.

But if counters repeat, tags may once again leak private information about data.



Then the tags $T_1$ and $T_2$ will be different with high probability.

# Results of [BN00,Kra01]



| | Encrypt-then-MAC | MAC-then-Encrypt | Encrypt-and-MAC |
|---|---|---|---|
| Privacy | Strong (CCA) | Weak (CPA) | Insecure |
| Integrity | Strong (CTXT) | Weak (PTXT) | Weak (PTXT) |

# Provable security

To prove that a scheme X is secure using reductions [GM]: Show that

- if one can compromise the security of X efficiently,
- then one can compromise the security of Y efficiently,

where Y is believed to be secure.

If Y is secure, an efficient adversary against X cannot exist.

# Security Evaluations

- First one out today
- Due next Tuesday

- Consider the security of the U.S. telecommunications system
- (Much like in-class study last week.)

## Project 1

- Out today
- Part 1: Due next Thursday (April 19, 11:59pm)
- Part 2: Due following Thursday (April 26, 11:59pm)

- Topic: Buffer overflow, format string, and double free vulnerabilities
- Seven vulnerable programs
- Your job: Attack them and obtain a root shell
- Readings on website will help!

## Project 1

- Start early! (That's why there's two deadlines.)
- Groups up to three people OK
  - Email Nick if you'd like us to pair you up
  - Goal is **not** to divide the vulnerable programs amongst yourselves
  - Goal is to work together on all vulnerable programs
    - You may be tested on how to attack these programs, and best way to deeply know the material is to do the attacks

## GDB will be helpful too!

- disassemble
- run
- continue
- break
  - break main
  - break *0x08048643
- step / stepi
- info register
- x
  - x/200x buf
  - x/200i buf
  - x/200a buf
  - x/200x $sp - 16

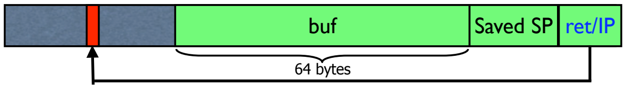
## Example

- Let's try attacking an example program

- Some of the following slides will not be online

## target0.c

```
int foo(char *arg, char *out) {
  strcpy(out, arg);
  return 0;
}
int main(int argc, char *argv[]) {
  char buf[64];  /* we want to overflow this buffer */
  if (argc != 2) { ... }
  foo(argv[1], buf);
  return 0;
}
```


buf
Saved SP
ret/IP
64 bytes