CSE 127: Computer Security

Cryptography

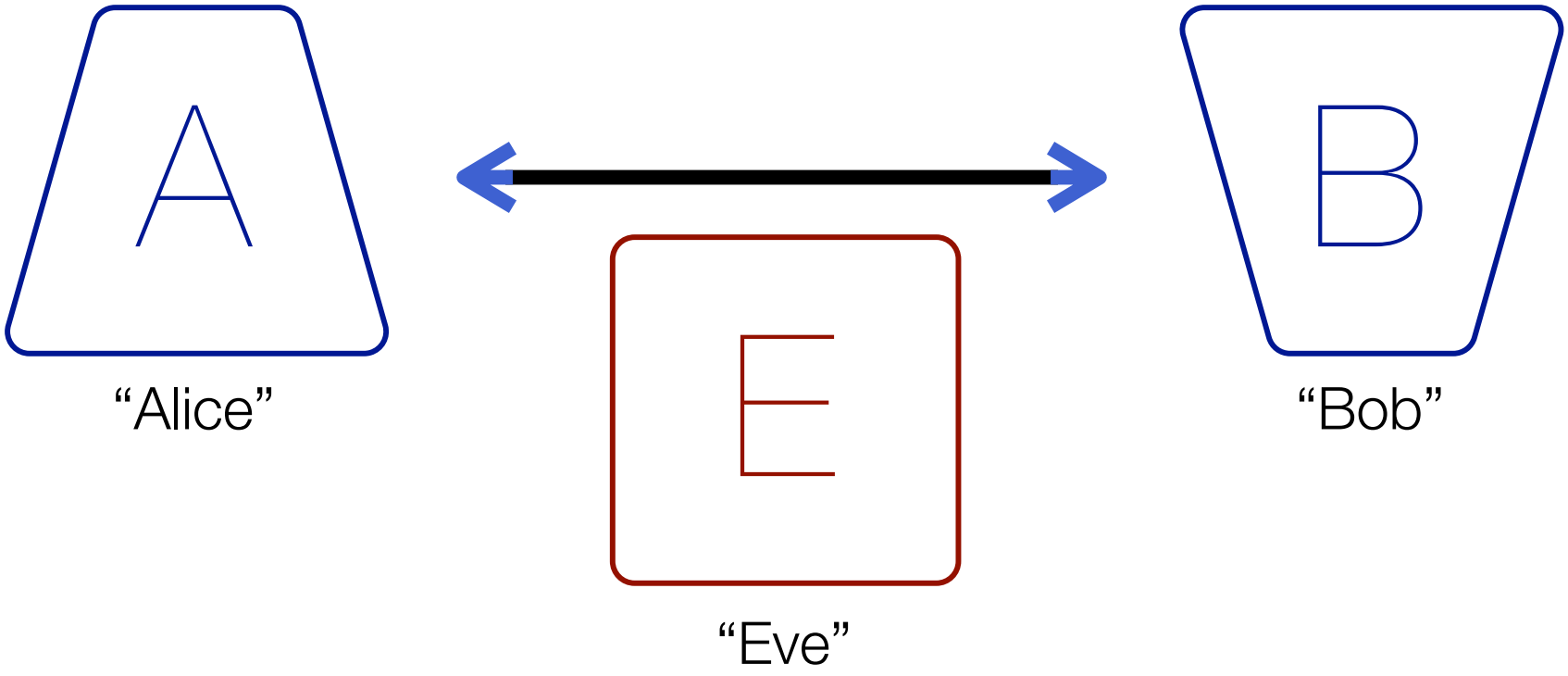
Kirill Levchenko

October 24, 2017

Motivation

- Two parties want to communicate securely
  - *Secrecy:* No one else can read messages
  - *Integrity:* messages cannot be modified
  - *Authenticity:* Parties cannot be impersonated
- **Example:** Military orders
  - Enemy can’t know what the orders say
  - Enemy can’t modify the orders
  - Enemy can’t send fake orders

Motivation

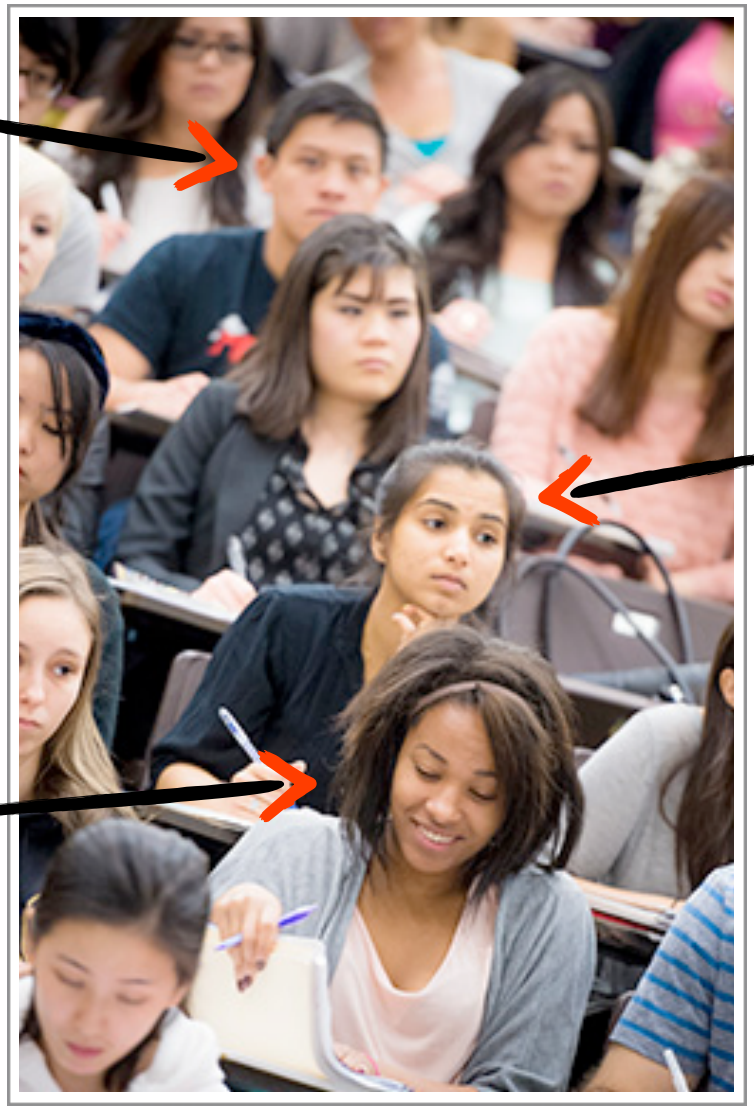


Setting

- Alice and Bob communicate by sending messages
- Eve can read, create, modify, or block messages
- Attacker model determines Eve's control of channel:
  - **Passive attack:** can read only
  - **Active attack:** can read and possibly create, modify, block
  - **Man-in-the-middle** (MitM) **attack:** read, create, modify, block

Simple Example

- Alice, Bob, Eve in lecture hall like this one
- Alice and Bob can pass notes … via Eve



Simple Example

- Alice wants to communicate to Bob whether or not she will work with him on CSE 127 Assignment 3
  - Think of this as one bit of information: “yes” or “no”
- Alice does not want Eve to know this … but Eve can observe every message between them
- How can Alice communicate with Bob without Eve knowing her answer?

Simple Example

- Let's say Alice and Bob create a secret code —
  - "Eagle" means **yes**, and "snake" means **no**
- Alice sends message "eagle" —
  - Bob knows this means **yes**
  - Eve learns nothing because she doesn't know code
- This is perfect secrecy
  - In practice we rarely get this

Simple Example

- What if Eve knows Alice and Bob have a secret code?
  - … that the secret code is a pair of code words?
  - … that the two code words are “eagle” and “snake”?
- Eve learns nothing even if she knows everything except which code word means **yes** and which means **no**
- Does this scheme provide **integrity? authenticity?**

Moar Encrypt

- We usually want to encrypt more than one bit of information
  - In general — binary string of arbitrary length
- Also want **integrity** and **authenticity**
  - Will get to this in a bit

Definitions

- **Plaintext:** unencrypted message to be communicated
  - From now on, assume this is a binary string
- **Ciphertext:** encrypted version of message
  - Also a binary string (may not be same length as plaintext)
- *Secrecy:* ciphertext reveals nothing about plaintext

$$C = E(M)$$

One-Time Pad

- We can achieve perfect secrecy if we XOR plaintext with a random stream of bits known only to Alice and Bob
- ***Why?***

$$C = M \oplus P$$

Plaintext (a binary string)

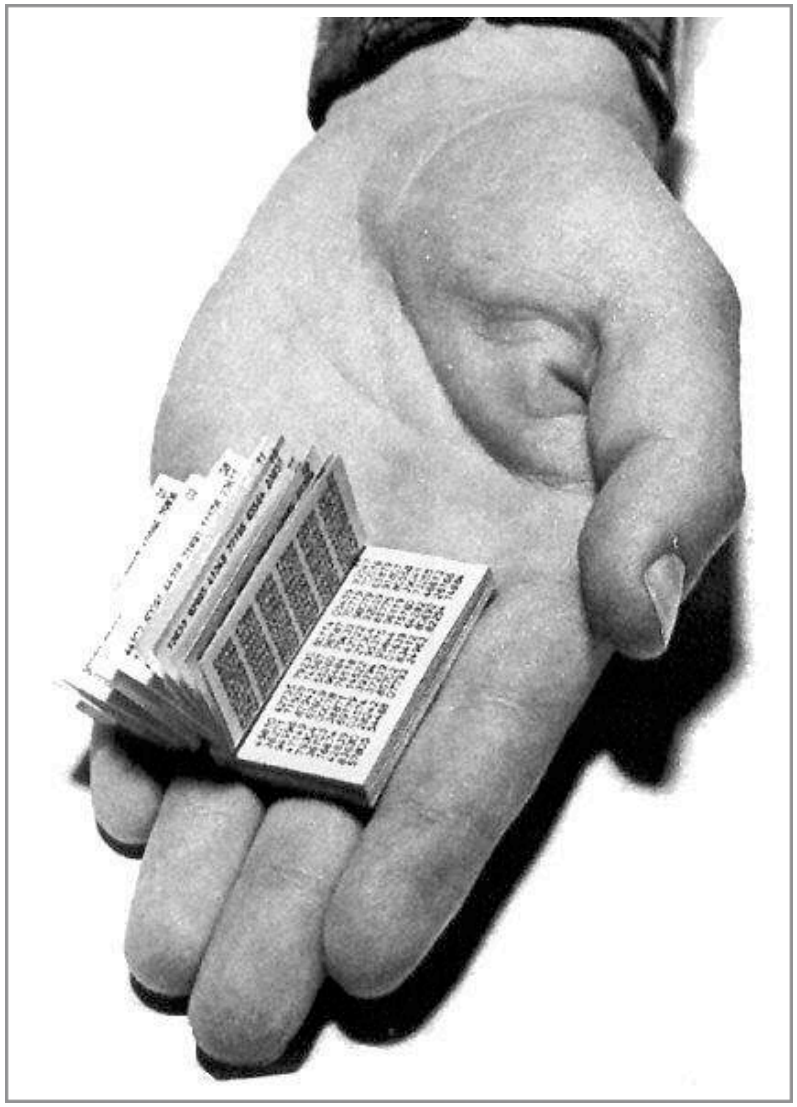
Random binary string of the same length as plaintext

$$C = M \oplus P$$

- For a given ciphertext, every plaintext is equally probable
  - Probability taken over random choice of pad P
- Eagle-Snake protocol as a one-time pad:
  - *Plaintext:* **yes** → 1, **no** → 0
  - *Ciphertext:* “eagle” → 1, “snake” → 0
  - If P = 0: “eagle” = **yes**, “snake” = **no**
  - If P = 1: “eagle” = **no**, “snake” = **yes**
- Perfect secrecy, no **integrity** or **authenticity**

One-Time Pads

- Used when perfect secrecy is necessary
- Requires a lot of pre-arranged secrets
- Each pad can only be used *once*
  - *Why?*



Computational Cryptography

- Want the size of the secret to be small
  - If pre-arranged secret smaller than message, not all plaintexts equally probable — ciphertext reveals info about plaintext
- Modern cryptography based on idea that learning anything about plaintext from ciphertext is *computationally* difficult without secret

Symmetric Cryptography

$$C = \mathrm{E}_k(M)$$

Encryption function

shared secret key

$$M = \mathrm{D}_k(C)$$

Decryption function

shared secret key

Ciphers

- **Stream cipher:** generate a pseudorandom pad as long as plaintext
  - **Pseudorandom**: hard to tell apart from random
    - Hard: computationally hard to distinguish from random
  - Can't reuse pad *(remember one-time pad!)*
- **Block cipher:** Encrypt/decrypt fixed-size messages
  - Need a way to encrypt longer or shorter messages

How Does It Work?

- **Goal:** learn how to use cryptographic primitives correctly
  - We will treat them as a black box that mostly does what it says
- To learn what's inside black box take CSE 107
- Avoid making your own crypto at all costs
  - This often fails, even when very smart people do it

How Does It Work?

- **Symmetric cryptographic primitives:**
  1 part arcane magic and folk superstition +
  2 parts bitter experience of past failures
  - When a primitive gets broken — move on to another one
  - NIST developing SHA-3 for when SHA-2 is broken
- **Asymmetric cryptographic primitives:**
  based on computational complexity of certain problems
  - Breaking one means breakthrough in solving hard problem
  - Have weathered the test of time better

“Encryption works. Properly implemented strong crypto systems are one of the few things that you can rely on.”

— *Edward Snowden*

Brute Force

- All encryption breakable by brute force given enough knowledge about plaintext
- Try to decrypt ciphertext with every possible key until a valid plaintext is found
- Attack complexity proportional to size of key space
  - Keys are just binary strings, size of key space expressed in bits
  - 64-bit key requires 2^{64} decryption attempts

Block Ciphers

- Block ciphers operate on fixed-size blocks
  - Common sizes: 64 and 128 bits
- A block cipher is a permutation of fixed-size inputs
  - Each input mapped to exactly one output
- Eagle-Snake protocol as a block cipher
  - Block size is 1 bit

Block Ciphers

- DES: Data Encryption Standard
  - Very old and widely-used cipher
  - **Key size:** 56 bits — vulnerable to brute force attack
  - **Block size:** 64 bits
- 3DES: 3 applications of DES (encrypt-decrypt-encrypt)
  - Can re-use existing DES hardware or libraries
  - 3 times slower than DES
  - **Key size:** 168 bits
  - **Block size:** 64 bits

Block Ciphers

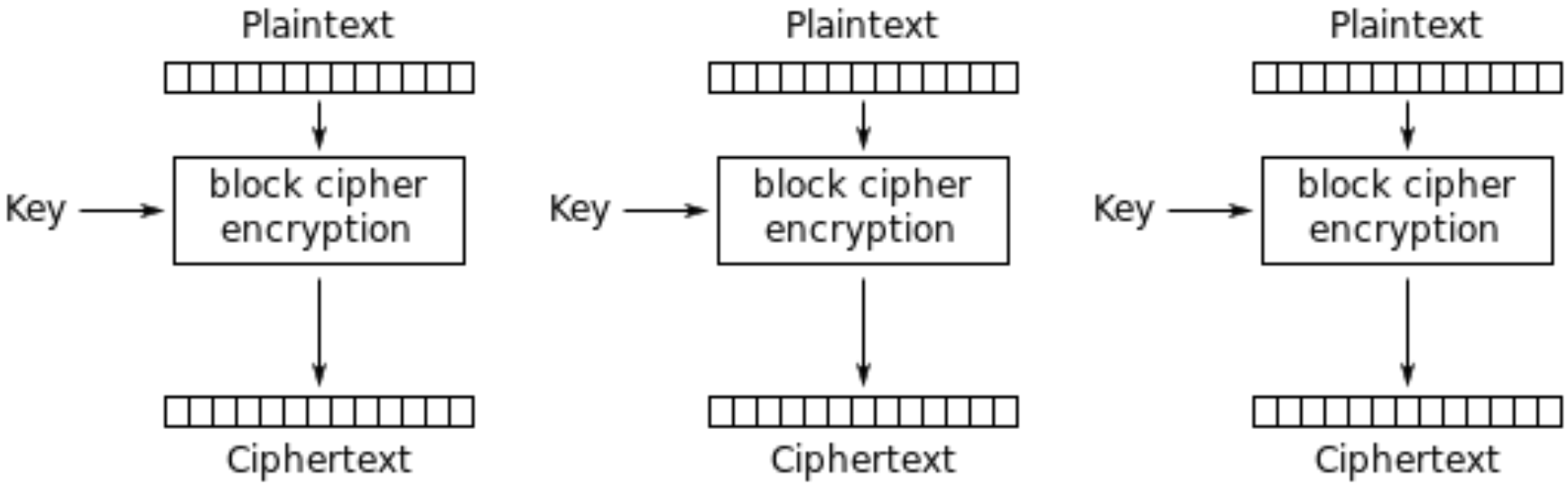
- AES: Advanced Encryption Standard
  - Replacement for DES based on Rijndael cipher
  - Faster than DES in software
  - **Key size:** 128, 192, 256 bits
  - **Block size:** 128 bits
- Twofish: AES competition finalist
  - Based on earlier Blowfish by Bruce Schneier
  - **Key size:** 128, 192, 256 bits
  - **Block size:** 128 bits

Block Cipher Modes

- Block ciphers encrypt/decrypt a single fixed-size block
- Several modes of operation for longer messages
- Shorter messages padded to full block size
  - Must be possible to distinguish pad from plaintext

Block Cipher Modes

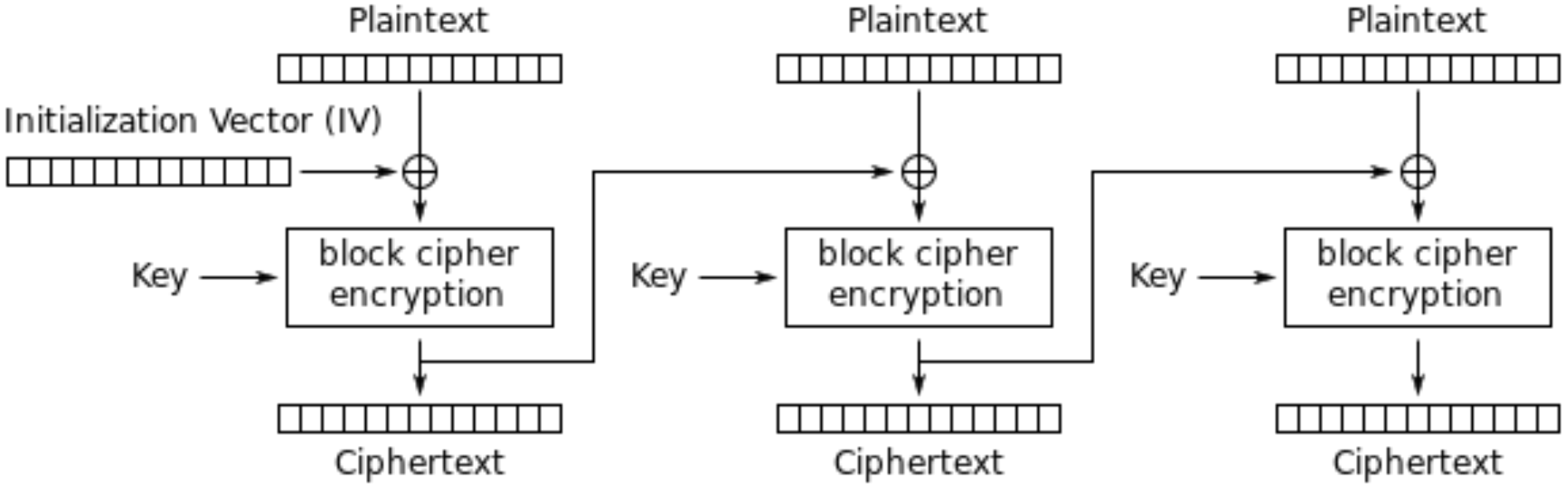
- **ECB:** Encrypt each block separately
- Avoid — ***why?***



Electronic Codebook (ECB) mode encryption

Block Cipher Modes

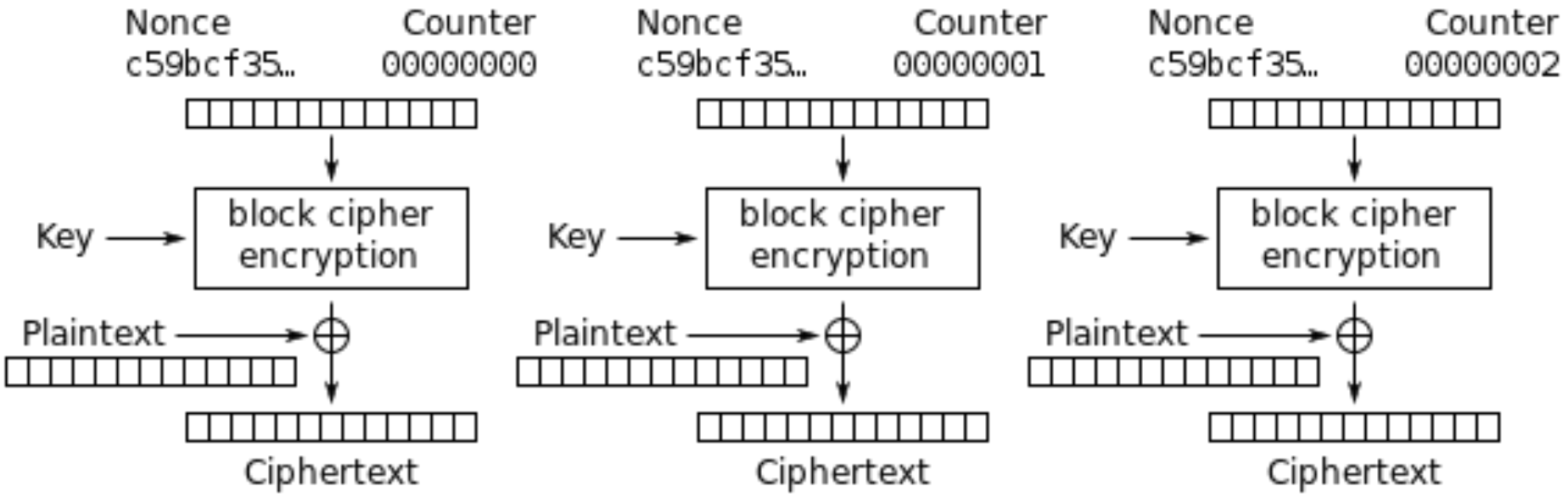
- **CBC:** XOR ciphertext block into next plaintext
- Use random IV
  - Subtle attack possible if attacker knows IV, controls plaintext



Cipher Block Chaining (CBC) mode encryption

Block Cipher Modes

- **CTR:** Counter mode
- Block cipher becomes stream cipher



Counter (CTR) mode encryption

Hash Functions

- A (cryptographic) hash function maps arbitrary length input into a fixed-size string

$$H = \mathrm{H}(M)$$

- Hash functions provide collision resistance — It is hard to find two inputs that have the same hash value

Hash Functions

- **MD5:** Message Digest
  - Designed by Ron Rivest
  - Very popular hash function
  - **Output:** 128 bits
  - Broken — **do not use**

Hash Functions

- **SHA-1:** Secure Hash Algorithm 1
  - Designed by NSA
  - **Output:** 160 bits
  - Considered to have weaknesses
- **SHA-2:** Secure Hash Algorithm 2
  - Designed by NSA
  - **Output:** 224, 256, 384, or 512 bits
  - Recommended for use today

Hash Functions

- **SHA-3:** Secure Hash Algorithm 3
  - Result of NIST SHA-3 contest
  - **Output:** arbitrary size
  - Replacement if SHA-2 broken

MACs

- *Goal:* Validate message integrity based on shared secret
- **MAC:** Message Authentication Code
- Keyed hash function using shared secret
- Hard compute hash without knowing key

$$A = \mathrm{MAC}_k(M)$$

MACs

- **HMAC:** MAC based on hash function
  - **HMAC-MD5:** HMAC construction using MD5
  - **HMAC-SHA1:** HMAC construction using SHA-1
- **CMAC:** MAC based on block cipher

Symmetric Primitives

$$A = \mathrm{MAC}_k(M)$$

$$C = \mathrm{E}_k(M)$$

$$M = \mathrm{D}_k(C)$$

Hash Function Properties

- Finding a pre-image is hard
  - pre-image is an input that hashes to a particular value
- Finding a collision is hard
  - collision are two inputs that has to same value
- **Non-property:** *secrecy*
  - Hash value may leak useful information about input
    - Although current hash functions not know not leak useful information
  - Compare to properties of ciphers

Cipher Properties

- Encryption and decryption are inverse operations:

$$M = \mathrm{D}_k(\mathrm{E}_k(M))$$

- Informally: ciphertext reveals nothing about plaintext
  - More formally: can't distinguish which of two plaintexts were encrypted without key
- **Non-property:** *integrity*
  - May be possible to change decrypted plaintext in known way
  - Need separate message authentication

MAC Properties

- Input of arbitrary length mapped to fixed-size string

$$A = \mathrm{MAC}_k(M)$$

- **Collision resistance:** Hard to find $M \neq M'$ such that
  - Pre-image resistance: hard to find M for given hash value S
  - Collision-resistance implies pre-image resistance
- **Non-property:** *secrecy*
  - Hash value may leak useful information about input

Asymmetric Cryptography

- Also called public key cryptography
- **Two separate keys:** public key and private (secret) key
- Public key known to everyone
- Private key used to decrypt and sign

Asymmetric Primitives

- Encryption and decryption
- Signing and verification

Asymmetric Keys

- Each user has a public and private key
- Keys related to each other in algorithm-dependent way
  - Can't just pick a random string as your key as with symmetric
  - Need a key-generation function
- Notation: K denotes public key, k denotes private key, r denotes random bits

$$(K, k) \leftarrow \text{Keygen}(r)$$

Encryption and Decryption

- Encryption uses **public key**

$$C = E_K(M)$$

- Decryption uses **private key**

$$M = D_k(C)$$

- Computationally hard to decrypt without private key
- Messages are fixed size

Asymmetric Usage

- Public directory contains everyone's public key
- To encrypt to a person, get their public key from directory
- No need for shared secrets!

Signing and Verification

- Signing uses **private key**

$$S = S_k(M)$$

- Verification uses **public key**

$$V_K(M, S)$$

- Computationally hard to sign without private key
- Messages of fixed size

Asymmetric Encryption

- **ElGamal encryption** (1985)
  - Based on Diffie-Helman key exchange (1976)
  - *Computational basis:* hardness of discrete logarithms
- **RSA encryption** (1978)
  - Invented by Rivest, Shamir, and Adleman
  - *Computational basis:* hardness of factoring

Asymmetric Signatures

- **DSA:** Digital Signature Algorithm (1991)
  - Closely related to ElGamal signature scheme (1984)
  - *Computational basis:* hardness of discrete logarithms
- **RSA signatures**
  - Invented by Rivest, Shamir, and Adleman
  - *Computational basis:* hardness of factoring

Key Sizes

Symmetric	ECC	DH/DSA/RSA
80	163	1024
112	233	2048
128	283	3072
192	409	7680
256	571	15360

Table 1: Comparable Key Sizes (in bits)

From RFC 4492

Practical Considerations

- Asymmetric cryptography operations generally much more expensive than symmetric operations
- Asymmetric primitives operate on fixed-size messages
- Usually combined with symmetric for performance
  - Use asymmetric to bootstrap ephemeral secret

Typical Encryption Usage

- Generate a ephemeral (one time) symmetric secret key
- Encrypt message using ephemeral secret key
- Encrypt ephemeral key using asymmetric encryption
- Send encrypted message, encrypted key

$$(E_K(k'), E_{k'}(M))$$

- Decryption: decrypt ephemeral key, decrypt message

Typical Signature Usage

- **Signing:** Compute cryptographic hash of message and sign it using asymmetric signature scheme
- **Verification:** Compute cryptographic hash of message and verify it using asymmetric signature scheme

Symmetric Primitives

$$A = \mathrm{MAC}_k(M)$$

$$C = \mathrm{Enc}_k(M)$$

$$M = \mathrm{Dec}_k(C)$$

Asymmetric Primitives

$(K, k) \leftarrow \text{Keygen}(r)$ $(K, k) \leftarrow \text{Keygen}(r)$

$C = E_K(M)$ $S = S_k(M)$

$M = D_k(C)$ $V_K(M, S)$