CSE 127: Intro to Computer Security

WI24

Lecture 13 - Symmetric-key Cryptography

Slide Credit: Kirill Levchenko, Stefan Savage, Deian Stefan, Nadia Heninger, Earlence Fernandes, William Enck, Brad Reaves, Micah Sherr, Patrick McDaniel, Douglas Reeves, Peng Ning, Imani Munyaka

Announcements

HW/PA3 Due Friday (Changed)

HW/PA5 Available Friday

HW/PA4 - You will received an email letting you know you can start

PA4/PA5 - Due W10 1pm PST

Cryptography

- Is:
  - A tremendous tool
  - The basis for many security mechanisms
- Is not:
  - The solution to all security problems
  - Reliable unless implemented and used properly
  - Something you should try to invent yourself
  - Another word for blockchain

How Does It Work?

- Goal: learn how to use crypto 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

r

.

How Does It Work?

- Goal: learn how to use crypto 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
- Do not roll your own crypto*

  * Exceptions: You are Daniel J. Bernstein, Joan Daemen, Neal Koblitz, Dan Boneh, or similar, or you have finished your PhD in cryptography under an advisor of that caliber, and your work has been accepted at Crypto, Eurocrypt, Asiacrypt, FSE, or PKC and/or NIST is running another competition, and then wait several years for full standardization and community vetting.

Terms

- **cryptosystem**: method of disguising (encrypting) plaintext messages so that only select parties can decipher (decrypt) the ciphertext
- **cryptography**: the art/science of developing and using cryptosystems
- **cryptanalysis**: the art/science of breaking cryptosystems
- **cryptology**: the combined study of cryptography and cryptanalysis

Types

- Secret key == symmetric key
  - Encryption and decryption keys are the same
- Public key == asymmetric key
  - Encryption and decryption keys differ!
- We'll start with symmetric key

Cryptosystem

- A cryptosystem is a 6-tuple consisting of

(E, D, G, M, C, K)

- Where,
  - **E** is an *encryption* algorithm
  - **D** is a *decryption* algorithm
  - **G** is a *key generation* algorithm
  - **M** is the set of *plaintexts*
  - **C** is the set of *ciphertexts*
  - **K** is the set of *keys*

What is secure encryption?

- Should be
  - Impossible for attacker to recover the key
  - Impossible for attacker to recover plaintext from ciphertext
  - Impossible for attacker to recover any character of the plaintext from ciphertext
  - Developed such that ciphertext leaks no additional info about plaintext regardless of info attacker already has
-

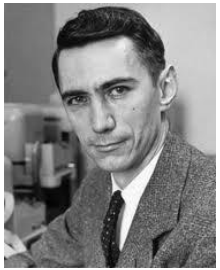
What is secure encryption? (webclicker)

a. Impossible for attacker to recover the key
b. Impossible for attacker to recover plaintext from ciphertext
c. Impossible for attacker to recover any character of the plaintext from ciphertext
d. Developed such that ciphertext leaks no additional info about plaintext regardless of info attacker already has
e. All of the above

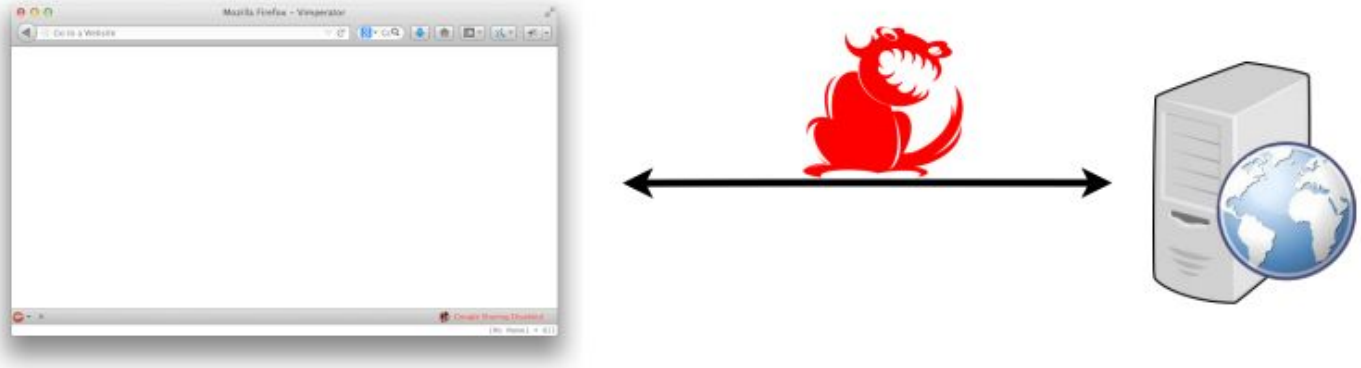
Kerckhoff's Principle

"The cipher method must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience"

"The enemy knows the system"

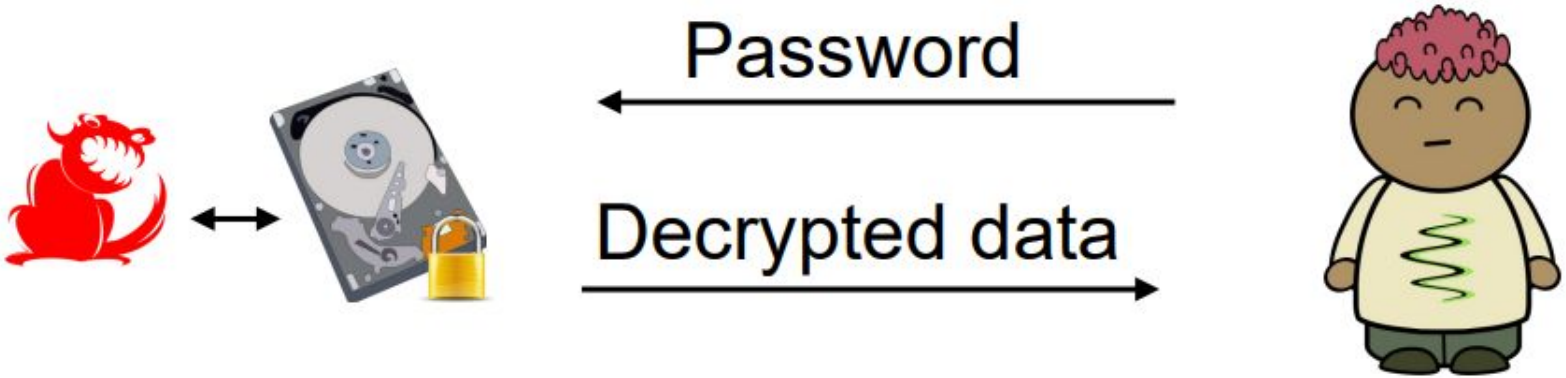


Real-world crypto: SSL/TLS



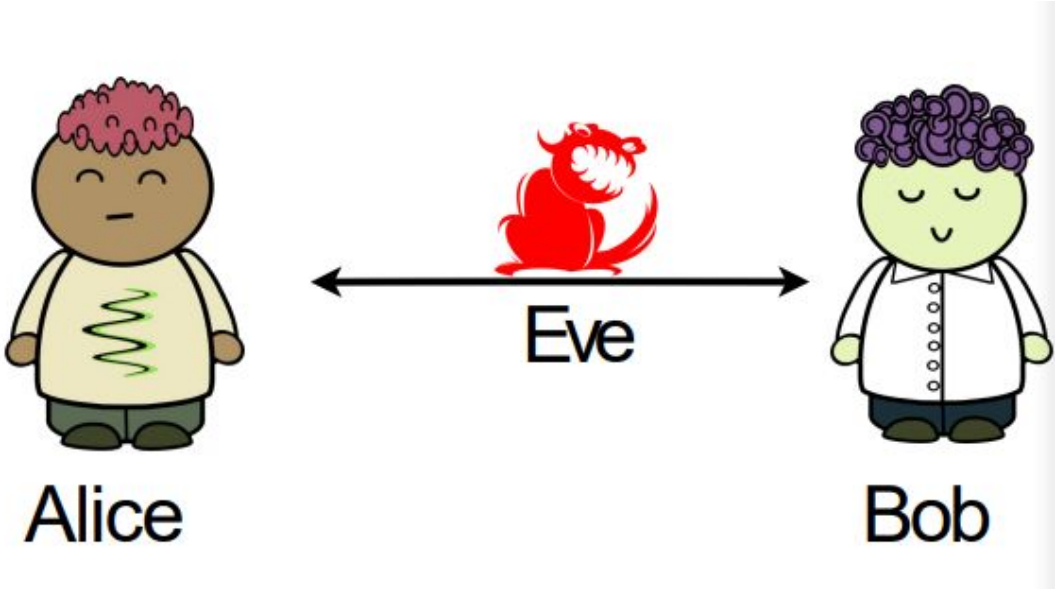
1. **Browser and web server run "handshake protocol":**
   - Establishes shared secret key using public-key cryptography (next lecture)
2. **Browser and web server use negotiated key to**

Real-world crypto: File encryption



- Files are symmetrically encrypted with a secret key
- The symmetric key is stored encrypted or in tamperproof hardware.
- The password is used to unlock the key so the data can be decrypted.

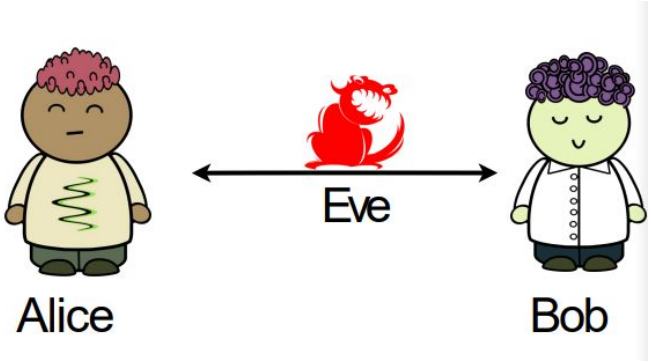
This class: secure communication



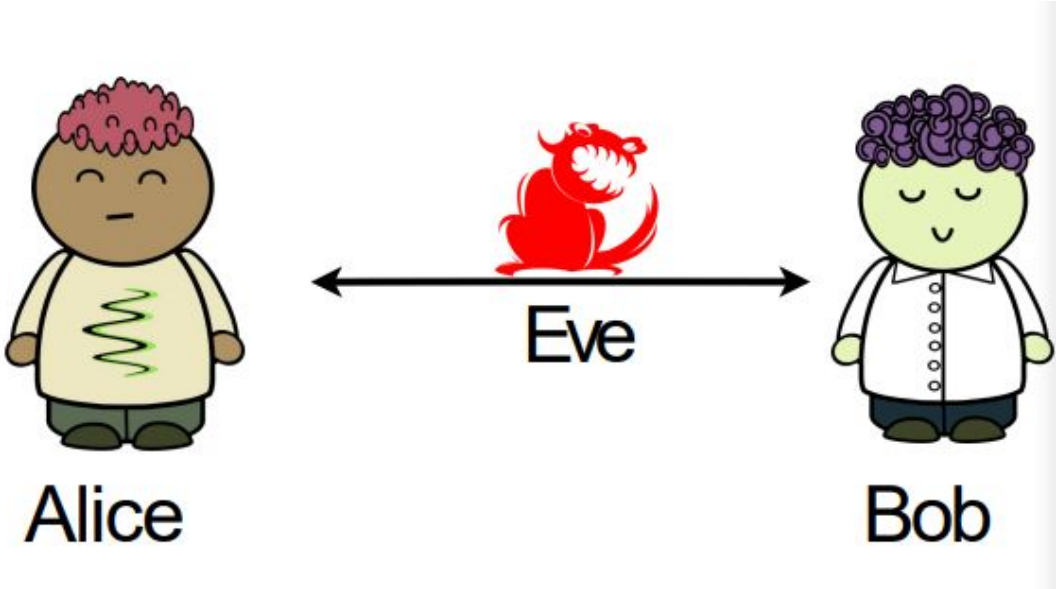
- Authenticity: Parties cannot be impersonated
- Secrecy: No one else can read messages
- Integrity: Messages cannot be modified

This class: secure communication

- **Confidentiality**
  - Keep data and communication secret
  - Encryption / decryption
- **Integrity**
  - Protect reliability of data against tampering
  - “Was this the original message that was sent?”
- **Authenticity**
  - Provide evidence that data/messages are from their purported originators
  - “Did Alice really send this message?”



Attacker models



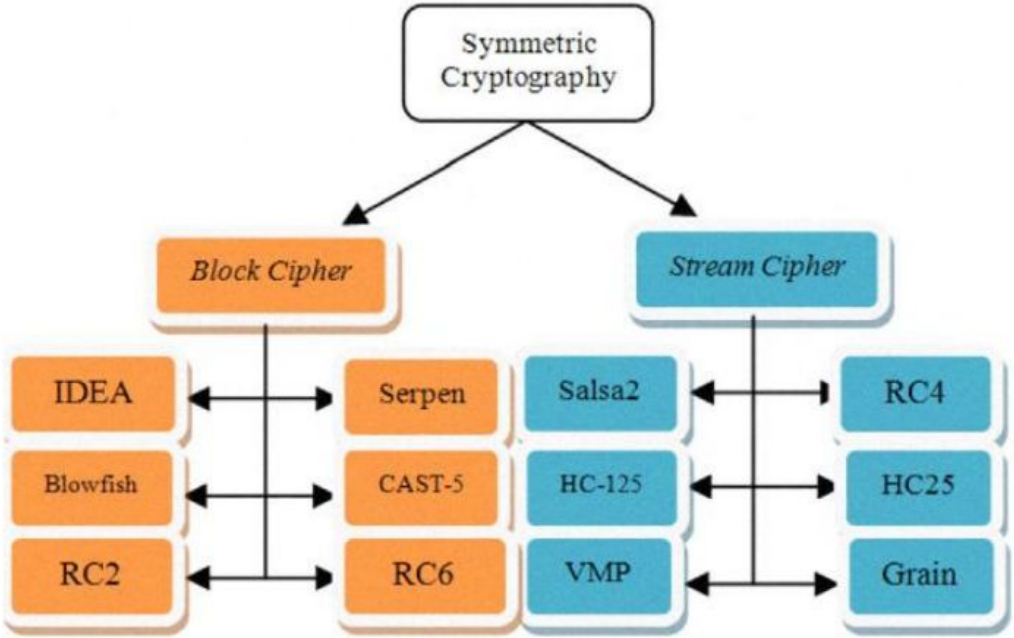
Passive attacker: Eve only snoops on channel

Active attacker: Eve can snoop, inject, block, tamper, etc.

Outline

- Symmetric-key crypto
  - Encryption
  - Hash functions
  - Message authentication codes
- Next time: asymmetric (public-key) crypto
  - Key exchange
  - Digital signatures

Symmetrical Cryptography

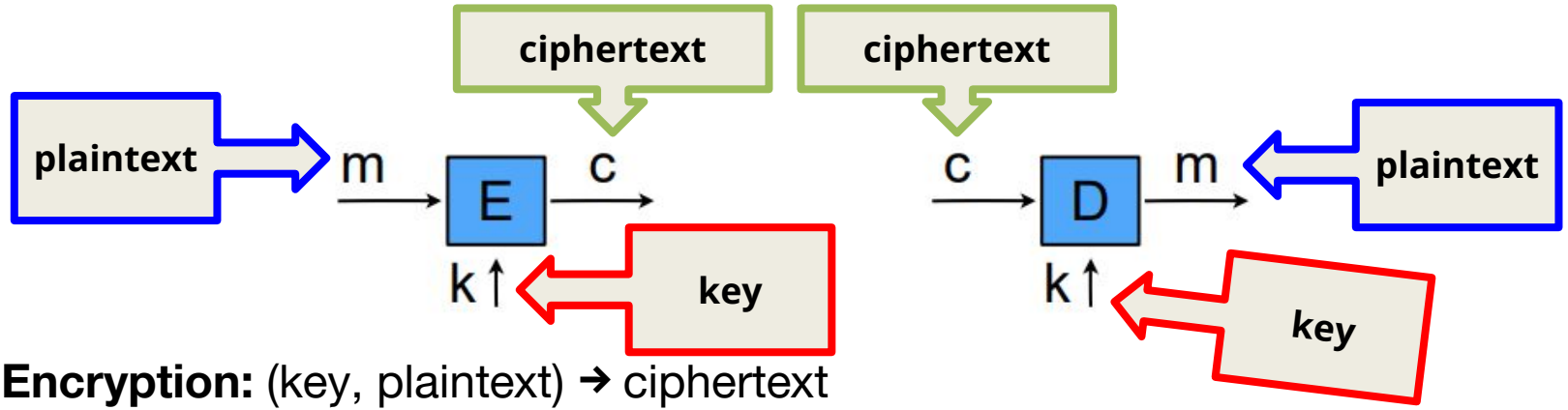


Sharif and Mansoor (2010)

Symmetric-key encryption

- **Encryption:** (key, plaintext) ➔ ciphertext
  - ➤ $E_k(m) = c$
- **Decryption:** (key, ciphertext) ➔ plaintext
  - ➤ $D_k(c) = m$
- **Functional property:** Where $D_k(E_k(m)) = m$

Symmetric-key encryption



- **Encryption:** (key, plaintext) ➔ ciphertext
  - $E_k(m) = c$
- **Decryption:** (key, ciphertext) ➔ plaintext
  - $D_k(c) = m$
- **Functional property:** Where $D_k(E_k(m)) = m$

Symmetric-key encryption

- **One-time key:** used to encrypt one message
  - E.g., encrypted email, new key generate per email

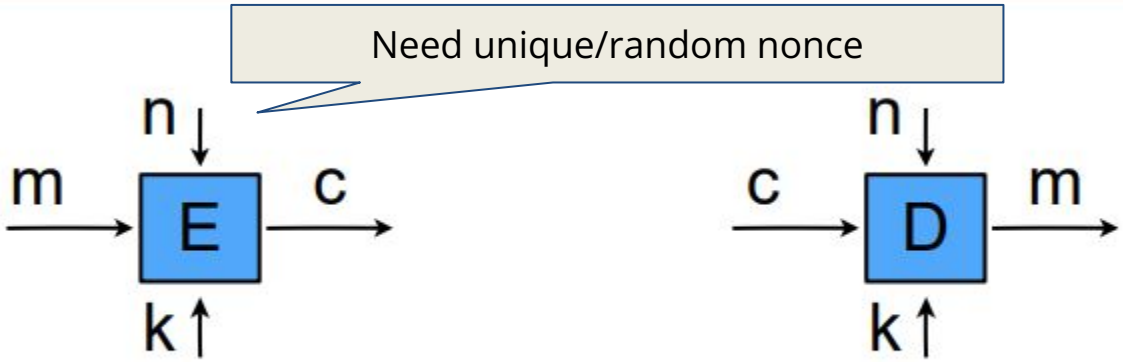
Symmetric-key encryption

- **One-time key:** used to encrypt one message
  - ➢ E.g., encrypted email, new key generate per email
- **Multi-use key:** used to encrypt multiple messages
  - ➢ E.g., same key used to encrypt many packets

Symmetric-key encryption

- **One-time key:** used to encrypt one message
  - ➢ E.g., encrypted email, new key generate per email
- **Multi-use key:** used to encrypt multiple messages
  - ➢ E.g., same key used to encrypt many packets

Symmetric-key encryption



- **One-time key:** used to encrypt one message
  - ➢ E.g., encrypted email, new key generate per email
- **Multi-use key:** used to encrypt multiple messages
  - ➢ E.g., same key used to encrypt many packets

Nonce - random number

Security definition: Passive eavesdropper

- Simplest security definition

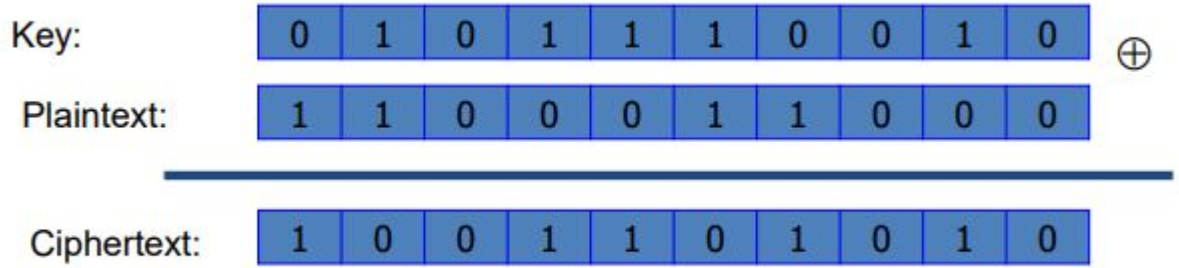
  ➢ How do you know an encryption scheme is secure against a passive eavesdropper?

  ➢ Want: “Ciphertext reveals nothing about plaintext”

  ➢ Informal formal definition: Given $E_k(m_1)$ and $E_k(m_2)$, attacker can’t distinguish which ciphertext encrypts which plaintext without key

Example: One Time Pad

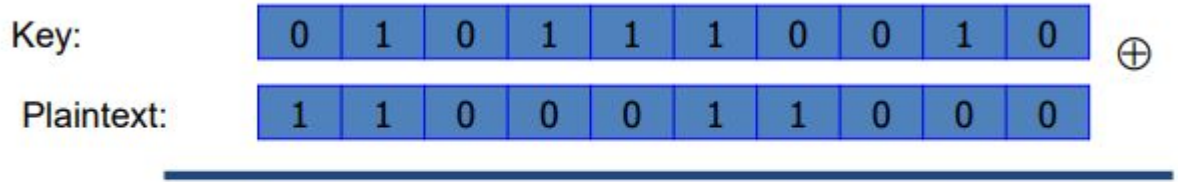
Vernam (1917)



- **Encryption:**
- **Decryption:**

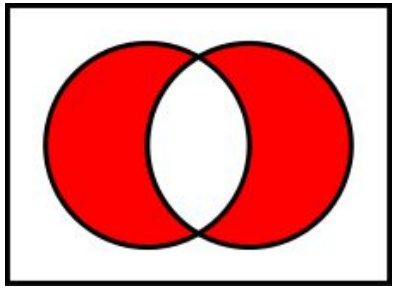
Example: One Time Pad

Vernam (1917)



Reminder XOR: true if and only if one is true and the other is false

- 1 XOR 1 = 0
- 1 XOR 0 = 1
- 0 XOR 1 = 1
- 0 XOR 0 = 0

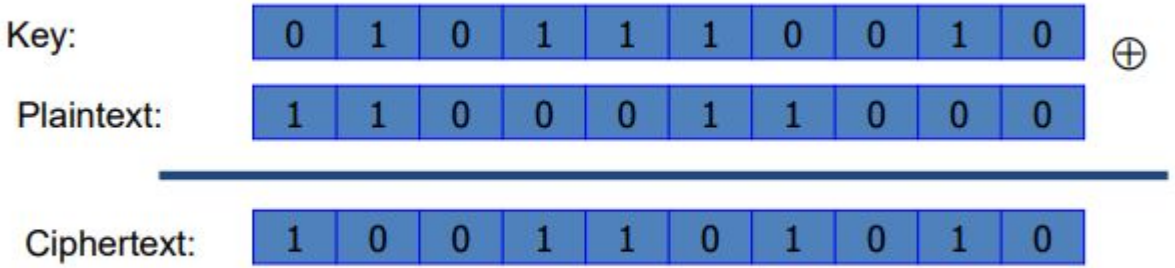


wikipedia.com

- E
- D

Example: One Time Pad

Vernam (1917)



- **Encryption:** $c = E_k(m) = m \oplus k$
- **Decryption:** $D_k(c) = c \oplus k = (m \oplus k) \oplus k = m$

OTP security

- Shannon (1949)
  - ➤ Information-theoretic security: without key, ciphertext reveals no “information” about plaintext
- Problems with OTP
  - Can only use key once
  - Key is as long as the message
  - No integrity protection

Confidentiality

- *Unconditional* or *probabilistic security*: cryptosystem offers provable guarantees, irrespective of computational abilities of an attacker
  - Given ciphertext, the probabilities that bit i of the plaintext is 0 is p and the probability
    that it is 1 is (1-p)
  - E.g., one-time pad
  - often requires key sizes that are equal to size of plaintext

- *Conditional* or *computational security*: cryptosystem is secure assuming a computationally bounded adversary, or under certain hardness assumptions (e.g., P<>NP)
  - E.g., DES, 3DES, AES, RSA, DSA, ECC, DH, MD5, SHA
  - Key sizes are much smaller (~128 bits)

- Almost all deployed modern cryptosystems are conditionally secure

Computational cryptography

- Want to encrypt with shorter keys
  - ➤ Problem: information-theoretic security is impossible if key space is smaller than message space.
- Solution: Use a more practical security notion
  - ➤ It should be infeasible for a computationally bounded attacker to violate security
  - ➤ In practice: attacks should take at least e.g., 2^{128} time

— Practice

Convert the following into Ciphertexts using the Vernam's OTP.

Question 1

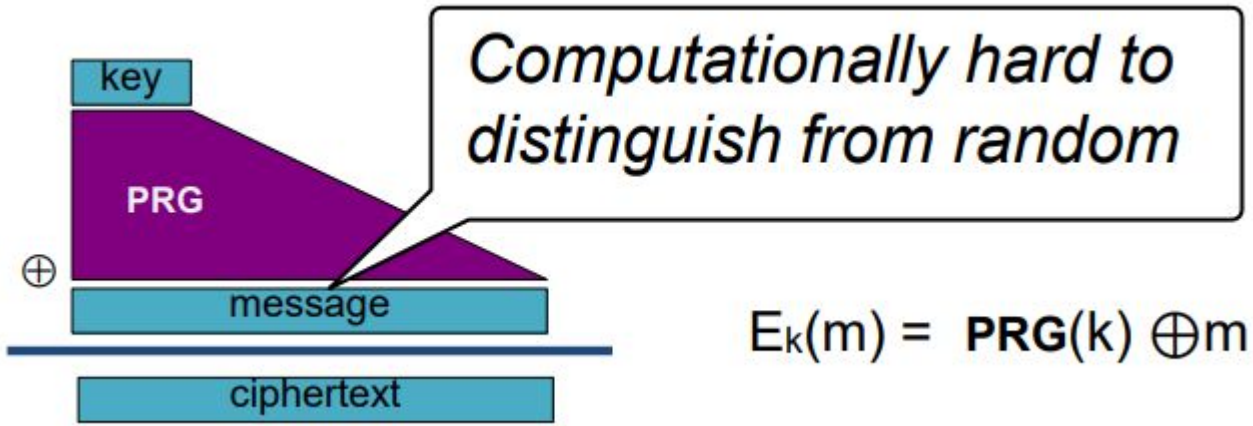
Key:	0	1	1	0	0	0	0	0	⊕
Plaintext:	1	0	0	1	0	0	0	0	
Ciphertext:									

Question 2

Key:	1	1	0	1	1	0	0	0	⊕
Plaintext:	1	1	0	1	0	0	0	0	
Ciphertext:									

Stream ciphers

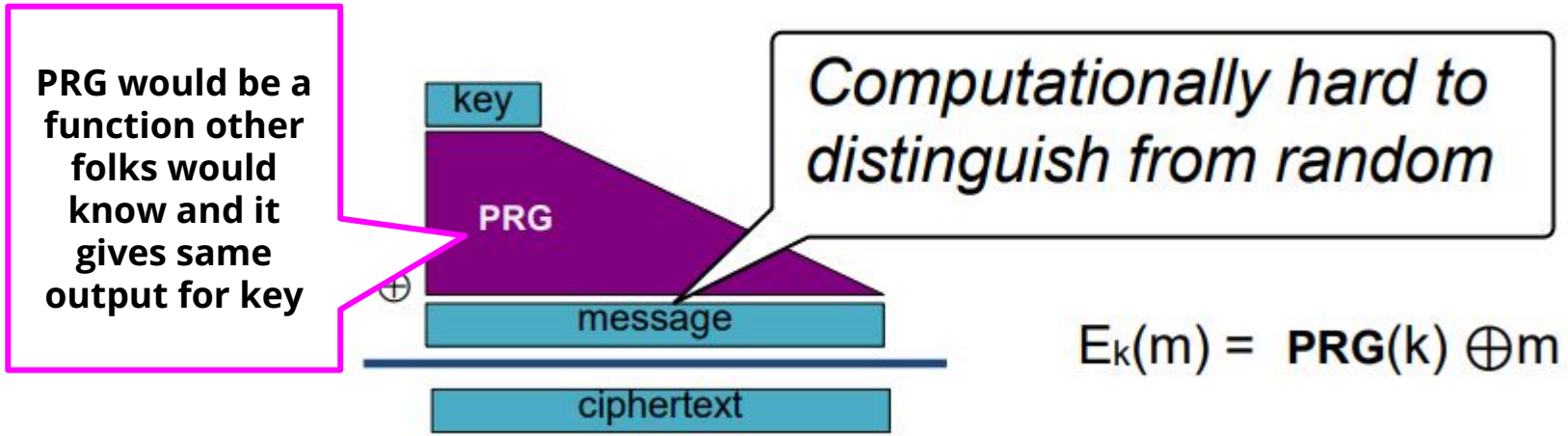
- Problem: OTP key is as long as message
- Solution: Pseudo random generator
- Stream ciphers uses bit-by-bit to generate ciphertext.



Examples: ChaCha, Salsa, etc.

Stream ciphers

- Problem: OTP key is as long as message
- Solution: Pseudo random generator
- Stream ciphers uses bit-by-bit to generate ciphertext.



Examples: ChaCha, Salsa, etc.

Dangers in using stream ciphers

- Can we use a key more than once?

  ➤ E.g., $c_1 \leftarrow m_1 \oplus PRG(k)$
  $c_2 \leftarrow m_2 \oplus PRG(k)$

  ➤Yes? No?

  ➤ Eavesdropper does: $c_1 \oplus c_2 \rightarrow m_1 \oplus m_2$

  ➤ Enough redundant information in English that:
  $m_1 \oplus m_2 \rightarrow m_1 , m_2$

Chosen plaintext attacks

- Attacker can learn encryptions for arbitrary plaintexts
- Historical example:
  - During WWII the US Navy sent messages about Midway Island and watched Japanese ciphertexts to learn codename (“AF”)
- More recent (but still a bit old) example:
  - WEP WiFi encryption has poor randomization and can result in the same stream cipher used multiple times

Block Ciphers vs. Stream Ciphers

- *Stream Ciphers*
  - Combine (e.g., XOR) plaintext with pseudorandom stream of bits
  - Pseudorandom stream generated based on key
  - XOR with same bit stream to recover plaintext
  - E.g., RC4, FISH

- *Block Ciphers*
  - Fixed block size
  - Encrypt block-sized portions of plaintext
  - Combine encrypted blocks (more on this later)
  - E.g., DES, 3DES, AES

Block ciphers: crypto work horses

- Block cipher: permutation of fixed-size input block
  - Each input is mapped to one output (depends on key)
- Common examples:
  - E.g., 3DES: |m| = |c| = 64 bits, |k| = 168 bits
  - E.g., AES: |m| = |c| = 128 bits, |k| = 128, 192, 256

Block ciphers: crypto work horses

- Block cipher: permutation of fixed-size input block
  - Each input is mapped to one output (depends on key)
- Common examples:
  - E.g., 3DES: |m| = |c| = 64 bits, |k| = 168 bits
  - E.g., AES: |m| = |c| = 128 bits, |k| = 128, 192, 256

Correct block cipher choice: AES

Challenges with block ciphers

- Block ciphers operate on single fixed-size block

Challenges with block ciphers

- Block ciphers operate on single fixed-size block
- How do we encrypt longer messages?

➢ Several modes of operation for longer messages

Challenges with block ciphers

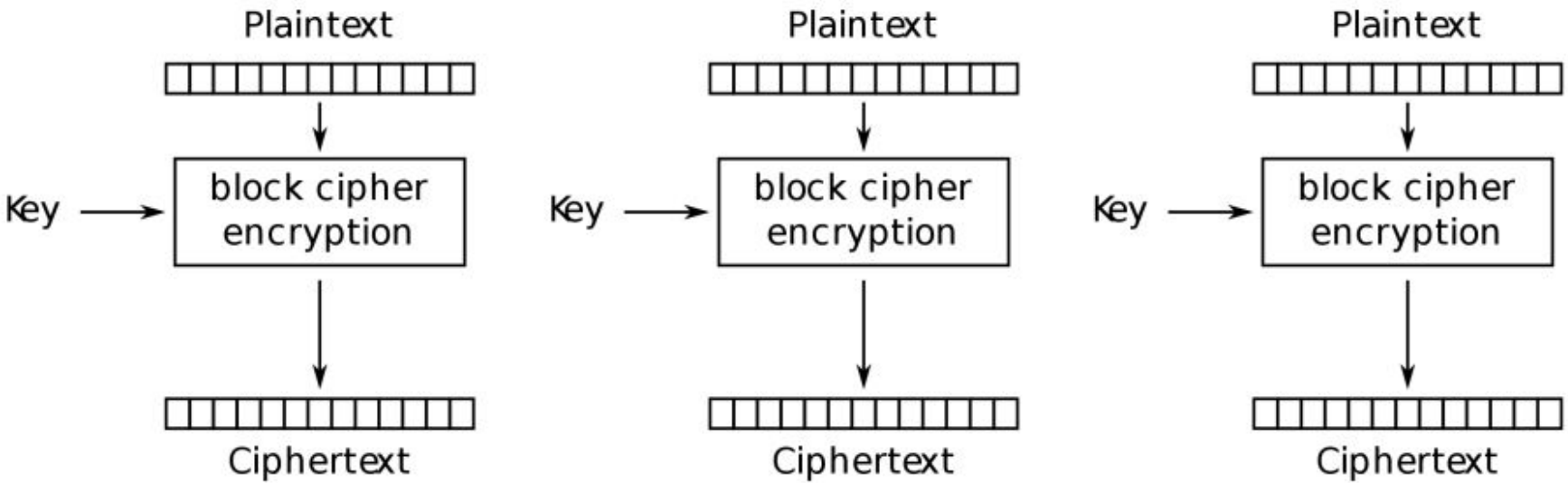
- Block ciphers operate on single fixed-size block
- How do we encrypt longer messages?

➢ Several modes of operation for longer messages

- How do we deal with messages that are not block-aligned?

➢ Must pad messages in a distinguishable way

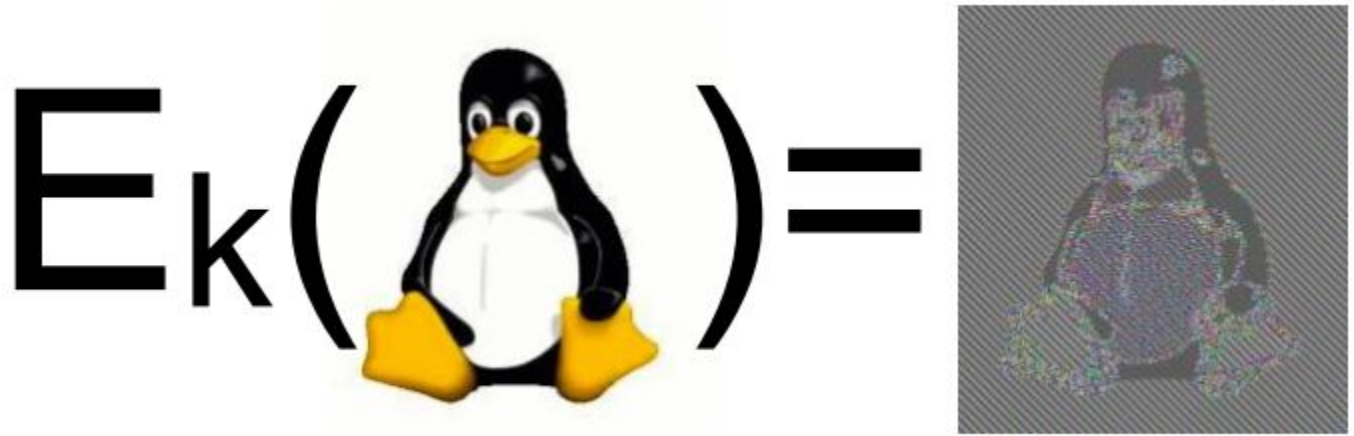
Insecure block cipher usage: ECB mode



Electronic Codebook (ECB) mode encryption

Source: wikipedia

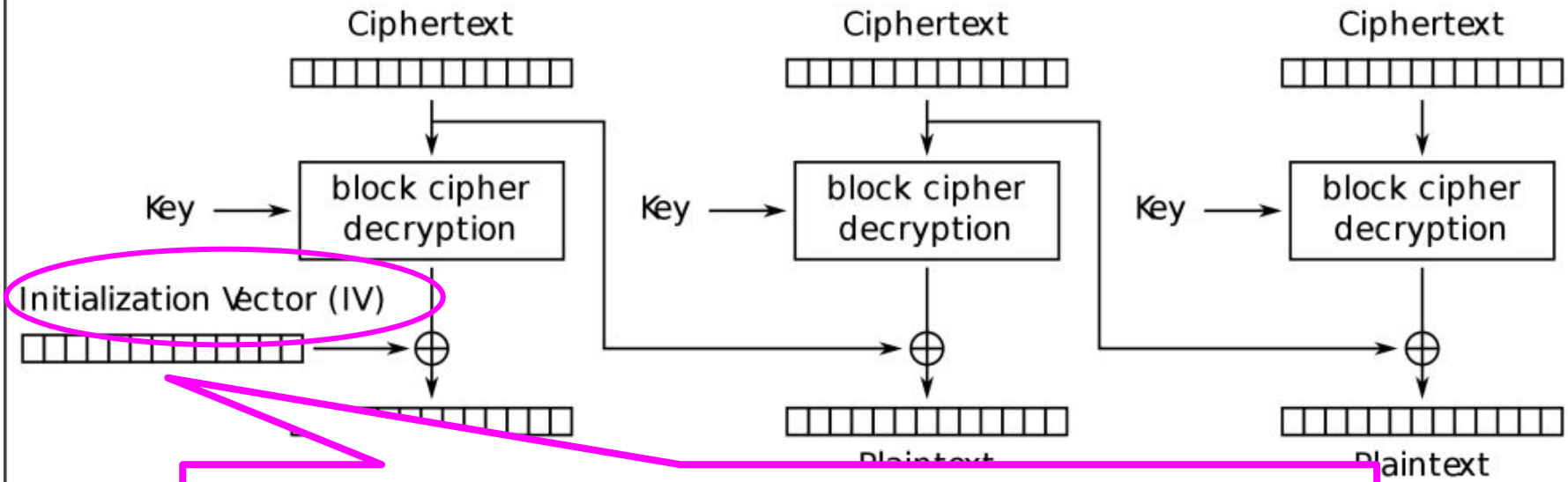
Why is ECB so bad?



Leaks info about message

Source: wikipedia

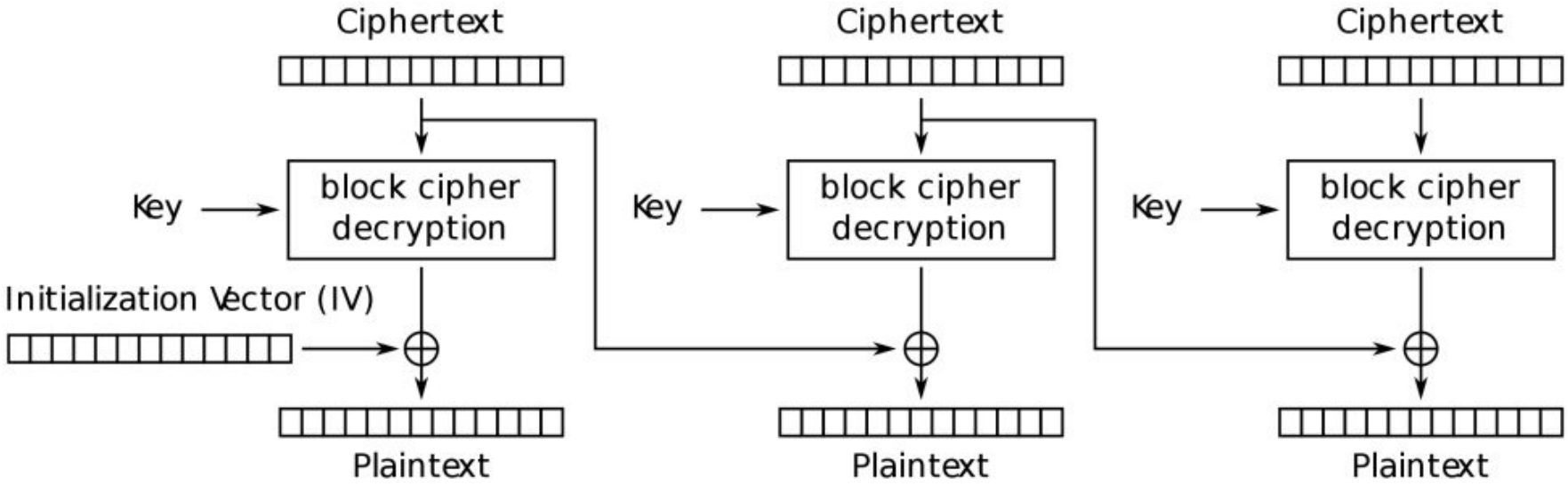
Moderately secure usage:CBC mode with random IV



Adds in a random initialization vector (IV) to randomize ciphertext appearance. IV isn't secret! It can be sent in the clear

Source: wikipedia

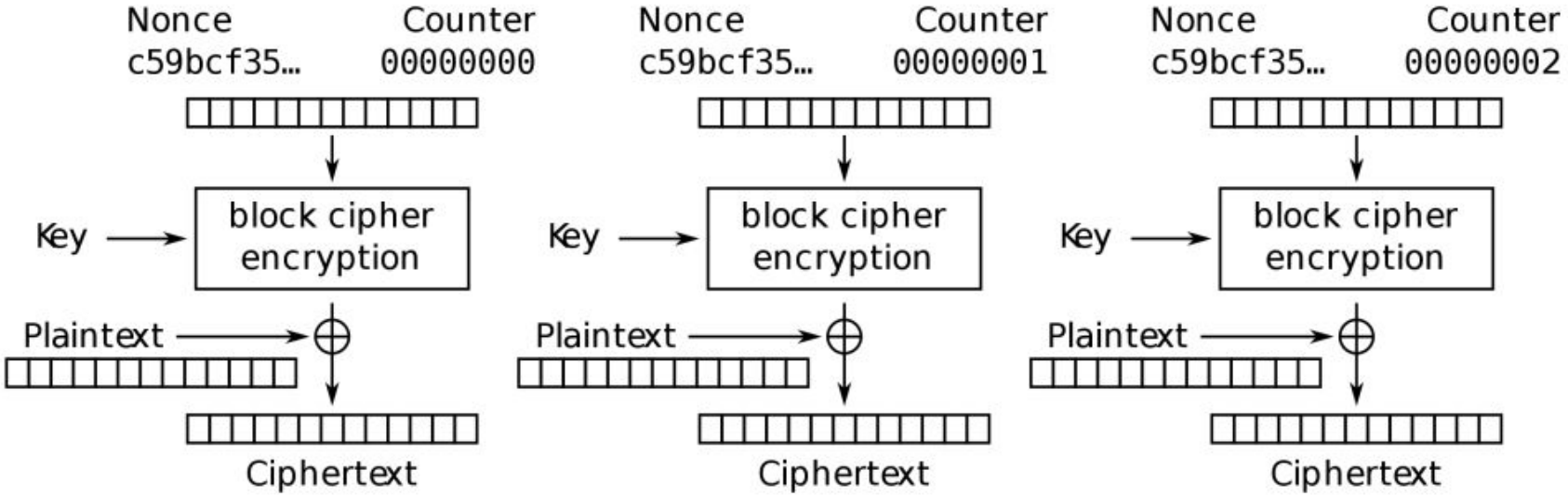
Moderately secure usage:CBC mode with random IV



Cipher Block Chaining (CBC) mode decryption

Subtle attacks that abuse padding possible!

Better block cipher usage: CTR mode with random IV



Counter (CTR) mode encryption

Essentially use block cipher as stream cipher!

Source: wikipedia

Better block cipher usage: CTR mode with random IV

- A block cipher is a *pseudorandom function*
  - Given an input but no key, it's hard to predict any part of the output
- Use this to generate a keystream by encrypting a counter the size of the block
  - Effectively turns a block cipher into a stream cipher
  - To prevent keystream reuse, we can start the counter at a *random* value

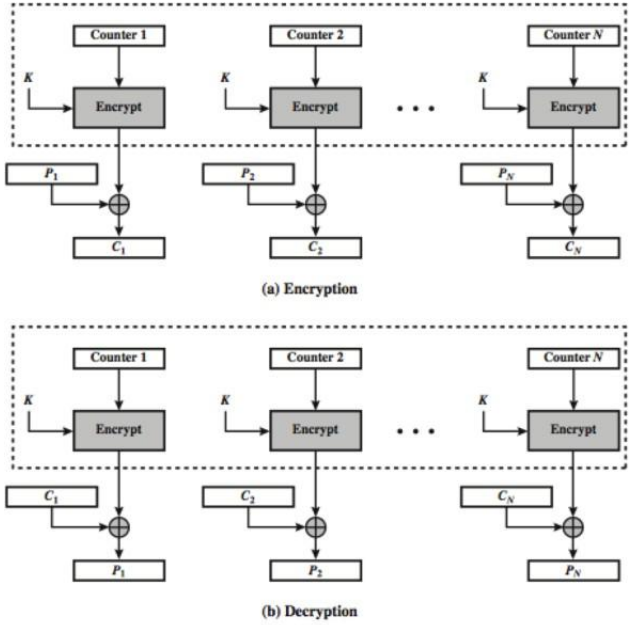


Figure 2.11 Counter (CTR) Mode

Source: wikipedia

What mode should you choose?

If your crypto library is making you choose a block cipher mode of operation, use a different library.

(Right answer: block cipher mode of operation can be built into an AEAD mode (end of lecture).)

Issues for Modes of Operation

- *Information leakage*: Does it reveal info about the plaintext blocks?
- *Ciphertext manipulation*: Can an attacker modify ciphertext block(s) in a way that will produce a predictable/desired change in the decrypted plaintext block(s)?
  - Note: assume the structure of the plaintext is known, e.g., first block is employee #1 salary, second block is employee #2 salary, etc.
- *Parallel/Sequential*: Can blocks of plaintext (ciphertext) be encrypted (decrypted) in parallel?
- *Error Propagation*: If there is an error in a plaintext (ciphertext) block, will there be an encryption (decryption) error in more than one ciphertext (plaintext) block?

Answer The Following (in pairs)

	ECB	CBC	CTR
Information Leakage	?	?	?
Ciphertext Manipulation	?	?	?
Parallel	?	?	?
Error Propagation	?	?	?
Random Access	?	?	?

Answer The Following (in pairs)

	ECB	CBC	CTR
Information Leakage	Yes	No	No
Ciphertext Manipulation	Yes	Yes	Yes
Parallel	Yes	no (encryption) yes (decryption)	Yes
Error Propagation	No	yes (encryption) a little (decryption)	No
Random Access	Yes	No	Yes

Choose the Right Cipher and Mode

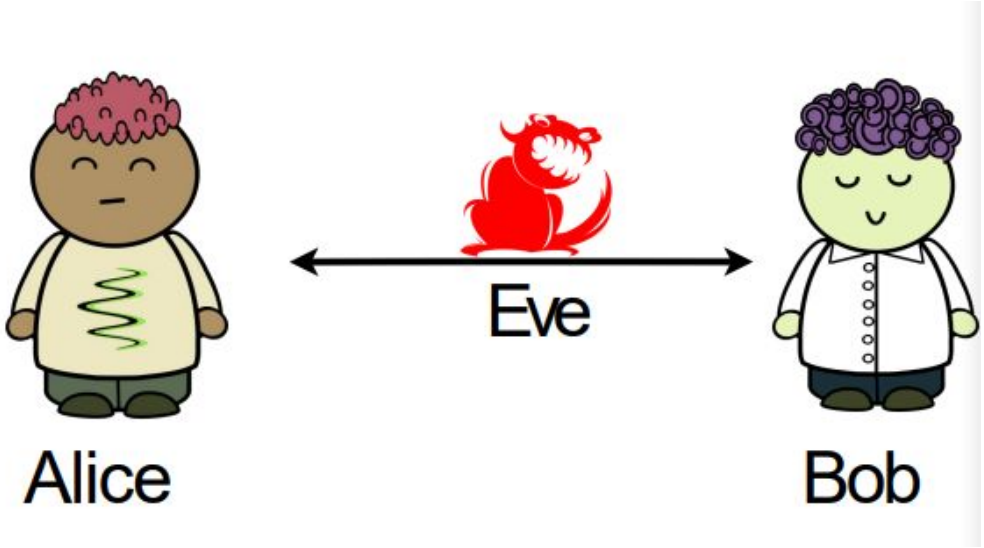
- Which ciphers and modes should be used for:
  - A messaging system where bit errors are common?
  - A network authentication scheme using short messages and ephemeral (short-lived, “throw-away”) keys?
  - A large file on disk with sequential access?
  - A large file on disk with random access?
  - Network messages that must not be altered in transmission?

What security do we get?

- 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
  - ➤ 128-bit key requires 2^{128} decryption attempts

Chosen ciphertext attacks

- What if Eve can alter the ciphertexts sent between Alice and Bob?



- Symmetric encryption alone is not enough to ensure security.
  - ➢ Need to protect integrity of ciphertexts (and thus underlying encrypted messages)

Outline

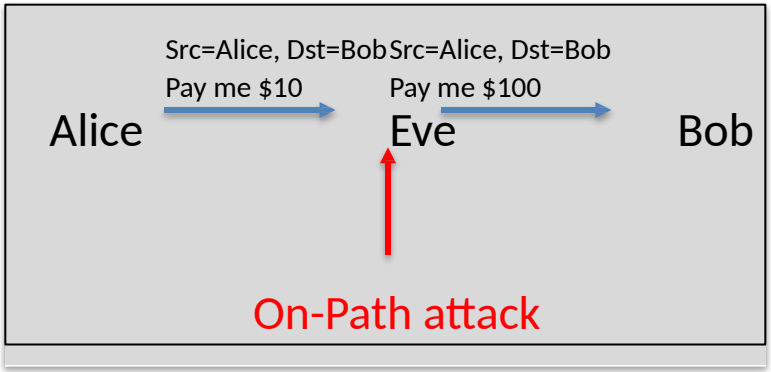
- Symmetric-key crypto
  - Encryption
  - Hash functions
  - Message authentication codes
- Asymmetric (public-key) crypto
  - Key exchange
  - Digital signatures

Outline

- Symmetric-key crypto
  - Encryption
  - Hash functions
  - Message authentication codes
- Asymmetric (public-key) crypto
  - Key exchange
  - Digital signatures

What Encryption Does and Does Not

- Does
  - Confidentiality
- Doesn't do
  - Data integrity
  - Source authentication
- **Need**: ensure that data is not altered and is from an authenticated source



Authentication vs. Integrity

- Data/Message *integrity* is about ensuring that data cannot change in an unauthorized way
  - Data has not changed on disk
  - Data has not changed in transit
- Data *authentication* / *authenticity* is about ensuring that a message originated from a particular source
  - A packet came from Alice
  - A program from the Internet was released by Microsoft
- Many of the tools used for one can be used for the other

Hash Functions

- A *hash function* is a function with a
  - *Arbitrary* and *variable* length input (called a “pre-image”)
  - *Fixed length* output (called a “hash” or “message digest”)
- Used all over computer science
  - Examples?
  - Hash Tables
  - Parity, Checksums

Properties of Hash Functions

- **Compression**: reduces arbitrary length string to fixed length hash
- **Ease of computation**: given message M, h(M) is easy to compute

Hash Function Properties

- Finding a preimage is hard
  - ➤ Given h, find m such that H(m)=h
- Finding a second preimage is hard
  - ➤ Given m_1, find m_2 such that $H(m_1)=H(m_2)$
- Finding a collision is hard
  - ➤ Find m_1 and m_2 such that $H(m_1)=H(m_2)$

Cryptographic Hash Functions

- A cryptographic hash function is a hash function with certain properties

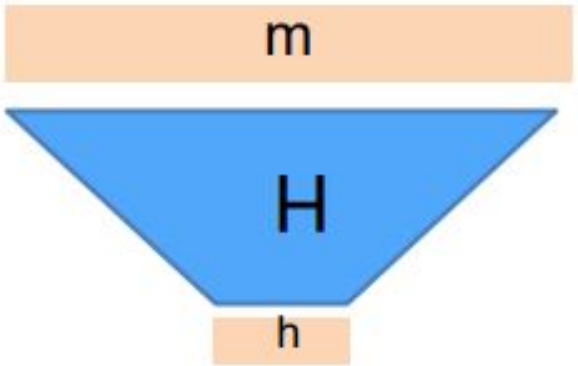
– preimage resistance: given digest y, computationally infeasible to find preimage x' such that h(x')=y (also called “one-way property”)

– 2nd-preimage resistance: given preimage x, computationally infeasible to find preimage x' such that h(x)=h(x')
(also called “weak collision resistance” and “target collision resistance")

– collision resistance: computationally infeasible to find preimages i,j such that h(i)=h(j)
(also called “strong collision resistance”)

Hash Functions

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



h=H(m)

- |m| is arbitrarily large
- |h| is fixed, usually 128-512 bits

Hash function security

- A 128-bit hash function has 64 bits of security
  - ➤ Birthday bound: find collision in time 2^{64}

Real-world crypto: Hash functions

- Versioning systems (e.g., git)
  - ➤ Better than _1, _final, _really_final
- Sub-resource integrity
  - ➤ Integrity of files you include from CDN
- File download integrity
  - ➤ Make sure the thing you download is the thing you thought you were downloading
- Blockchain

blob: 41732ca416bc88034636778b4a76fa0ea03c4ebc (plain)

```
# Maintainer: Deian Stefan

pkgname=xwrits
pkgver=2.26
pkgrel=1
pkgdesc="reminds you to take wrist breaks "
arch=('any')
url="http://www.lcdf.org/xwrits/"
license=('GPLv2')
depends=()
makedepends=()
conflicts=()
source=("http://www.lcdf.org/xwrits/$pkgname-$pkgver.tar.gz")
sha256sums=('aaca4809b4cd62a627335ca14a231d4ab556fc872458bdb6fdbf6e76b103fed8')
sha512sums=('c8beeca957e41468d85819a7d6d4475c83a99735ff17d13d724658a421d1d3b9a15191ee8ab903104ab19b869a4832103dbe7d3ec2a9bf89ae95a7899e92f927')

build() {
  cd "$pkgname-$pkgver"
  ./configure --prefix=/usr
  make
}

check() {
  cd "$pkgname-$pkgver"
  make -k check
}

package() {
  cd "$pkgname-$pkgver"
  make DESTDIR="$pkgdir/" install
}
```

Popular broken hash functions

- MD5: Message Digest
  - Designed by Ron Rivest
  - Output: 128 bits
- SHA-1: Secure Hash Algorithm 1
  - Designed by NSA
  - Output: 160 bits

Hash functions

- SHA-2: Secure Hash Algorithm 2
  - Designed by NSA
  - Output: 224, 256, 384, or 512 bits
- SHA-3: Secure Hash Algorithm 3
  - Result of NIST SHA-3 contest
  - Output: arbitrary size
  - Replacement once SHA-2 broken

Consequences of These Properties

- For a good cryptographic hash, if you change one bit of the input, the output should change drastically and unpredictably
  - MD5(“ABCD”) = 0xed5d34c74e59d16bd6d5b3683db655c3
  - MD5(“ABCE”) = 0x95741cb5c4ee614792f6f5a44f2e107a

- So if you need to know if a file has changed, hash it!

Birthday Attack

- A birthday attack is a name used to refer to a class of brute-force attacks
  - Birthday paradox: the probability that two or more people in a group of 23 share the same birthday is greater than 50%

- General formulation (How to estimate)
  - On repeated random inputs $n=\{n_1, n_2, \ldots, n_k\}$
    - $Pr(n_i = n_j) > 0.5 \Rightarrow 1.2k^{1/2}$, for some $1 <= i,j <= k, 1 <= j < k, I != j$
    - E.g., $1.2(365^{1/2}) \sim= 23$
  - *Implication*: Collisions can be found in approximately square root of the hash output size

Finding Collisions

- Choosing two messages that have the same hash h(x) = h(x') is more practical than you might think
- Example attack: secretary is asked to write a "bad" letter, but wants to replace with a "good" letter
  - Boss signs the letter after reading.

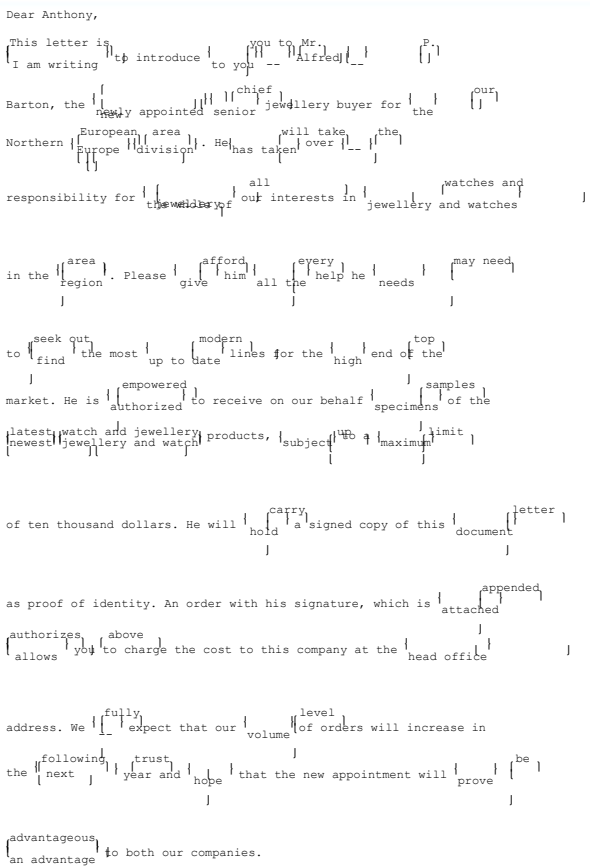
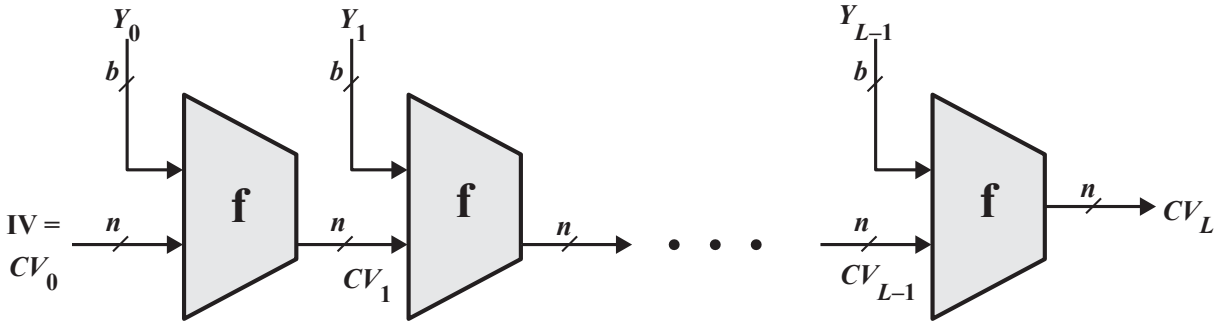


Figure 11.7 A Letter in 2^{37} Variations

(from Stallings, Crypto and Net

General Structure of Hash

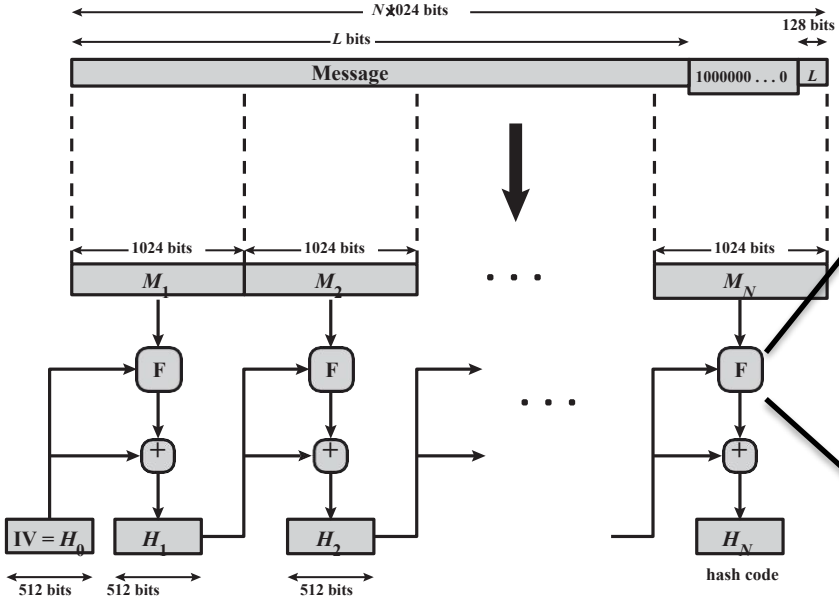


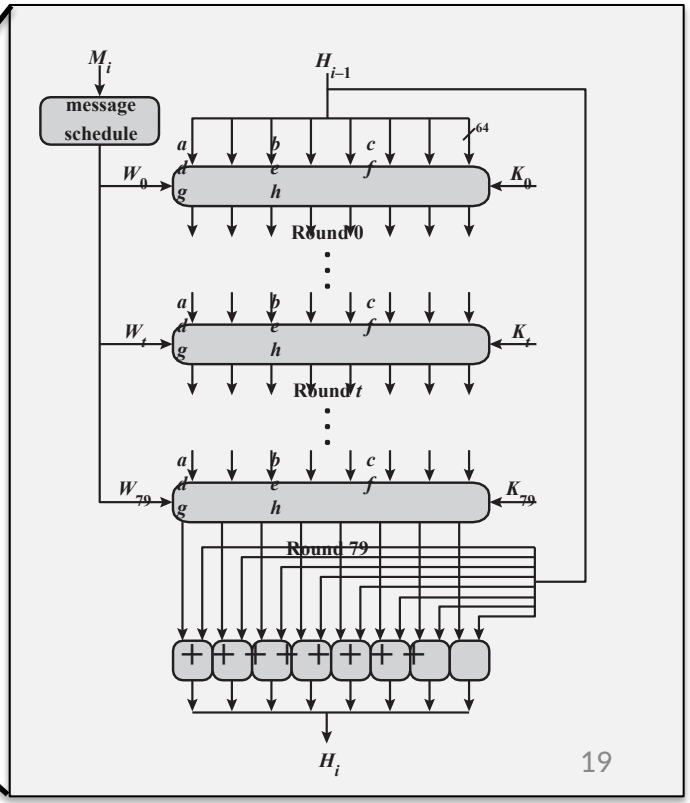
IV = Initial value
CV_i = chaining variable
Y_i = ith input block
f = compression algorithm

L = number of input blocks
n = length of hash code
b = length of input block

(from Stallings, Crypto and Net Security)

Example: SHA-512





(from Stallings, Crypto and Net Security)

Outline

- Symmetric-key crypto
  - Encryption
  - Hash functions
  - Message authentication codes
- Asymmetric (public-key) crypto
  - Key exchange
  - Digital signatures

MACs

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

$$a=MAC_k(m)$$

Message Authentication Codes (MACs)

- MACs provide message *integrity* and *authenticity*
- $MAC_K(M)$ – use symmetric encryption to produce short sequence of bits that depends on both the message (M) and the key (K)
- MACs should be resistant to *existential forgery*: Eve should not be able to produce a valid MAC for a message M' without knowing K

- To provide confidentiality, authenticity, and integrity of a message, Alice sends – $[E_K(M), MAC_K(E_K(M))]$ where $E_K(X)$ is encryption of X using key K
- Proves that M was encrypted (confidentiality and integrity) by someone who knew K (authenticity)

* In practice, you want to use different keys for E and MAC

A Better MAC

- Objectives
  - Use available hash functions without modification
  - Easily replace embedded hash function as more secure ones are found
  - Preserve original performance of hash function
  - Easy to use

HMAC construction

- HMAC: MAC based on hash function

$$MAC_k(m) = H(k \oplus opad \,\|\, H(k \oplus ipad \,\|\, m))$$

  - HMAC-SHA256: HMAC construction using SHA-256

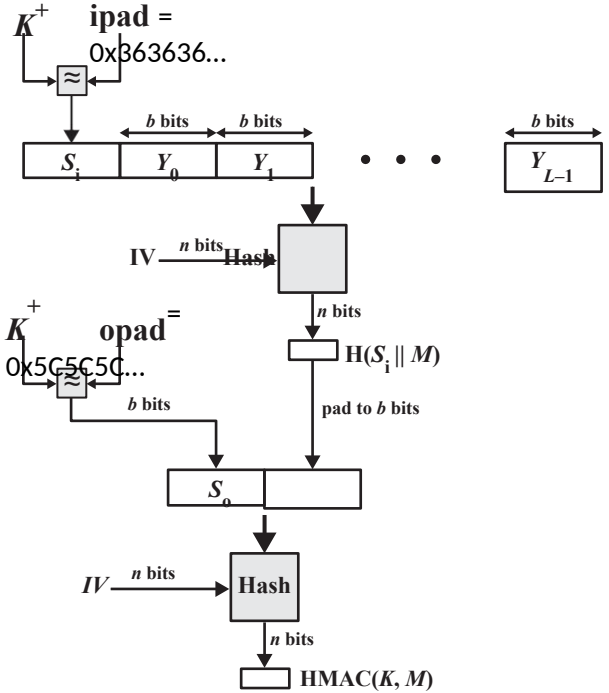
HMAC

- HMAC(k,M) =

  H(k⊕opad || H(k⊕ipad || M))

  – Attacker cannot extend MAC as before

  – Prove it to yourself



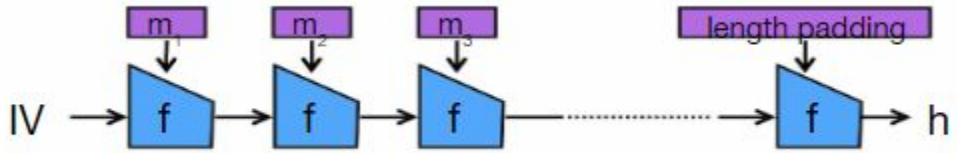
(from Stallings, Crypto and Net Securit_y_)

Creating a MAC from a Hash

- Consider the following simple hash-based MAC
  - $MAC_k(M) = h(k|M)$
- Suppose Eve wanted to append M' to M?
  - Goal: compute h(k|M|M') without knowing k
- Solution: Use h(k|M) as IV for next f iteration in h()
- Known as a Message Extension Attack

Length extension attack

- Merkle-Damgård construction: hash function from collision-resistant compression function f



- Attacker that can observe $MAC_k(m)$ can forge $MAC_k(m||padding||r)$ for an r of their choice

Other MAC constructions

- In 2009, Flickr required API calls to use authentication token that looked like:

  MD5(secret || arg1=val1&arg2=val2&...)

- Is $MAC_k(m) = H(k \,||\, m)$ a secure MAC?
  - ➤ No! If H is MD5, SHA1 or SHA2

Other MAC constructions

• In 2009, Flickr required API calls to use authentication token that looked like:

MD5(secret || arg1=val1&arg2=val2&...)

• Is $MAC_k(m) = H(k || m)$ a secure MAC?

- ➤ No! If H is MD5, SHA1 or SHA2
- ➤ Use HMAC!

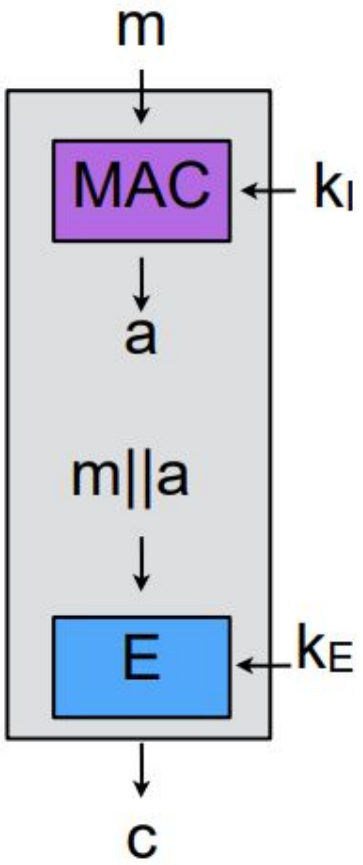
Combining Crypto has Implications

- Traditionally three ways of combining encryption and MAC as discrete operations:
  - Encrypt-then-MAC (EtM) [used in IPsec] {best}
  - Encrypt-and-MAC (E&M) [used in SSH] {bad}
  - MAC-then-Encrypt (MtE) [used in SSL/TLS] {padding attacks}
- All three are used in real protocols, but E&M and MtE require protocol modifications to be strongly unforgeable
- For more information, see:
  - https://en.wikipedia.org/wiki/Authenticated_encryption
  - https://eprint.iacr.org/2014/206 (EuroCrypt'14), starts the discussion that all three have problems in their generic composition

Combining MAC with encryption

MAC then Encrypt (SSL)

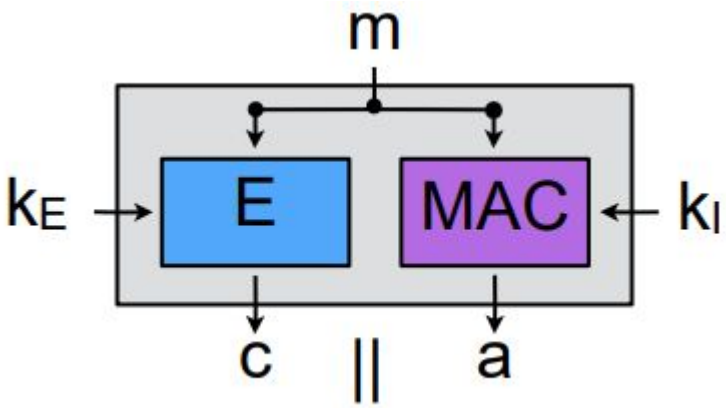
- Integrity for plaintext not ciphertext
- Issue: need to decrypt before you can verify integrity
- Hard to get right!



Combining MAC with encryption

Encrypt and MAC (SSH)

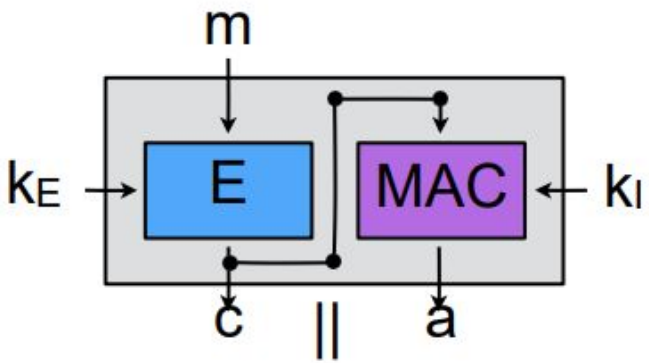
- Integrity for plaintext not ciphertext
- Issue: need to decrypt before you can verify integrity
- Hard to get right!



Combining MAC with encryption

Encrypt then MAC (IPSec)

- Integrity for plaintext and ciphertext
- Almost always right!



Authenticated Encryption (AE)

- Several modes of operation provide both encryption and authentication at the same time!
- Popular modes include
  - CCM (Counter with CBC-MAC)
  - GCM (Galois/Counter Mode)

AEAD construction

- Authenticated Encryption with Associated Data
  - AES-GCM, AES-GCM-SIV
- Always use an authenticated encryption mode
  - Combines mode of operation with integrity protection/ MAC in the right way

Galois/Counter Mode (GCM)

- Message is encrypted in a variant of CTR mode
- MAC comes from multiplication of ciphertext and key (H) in GF(2^{128}), in a chain similar to CBC

(https://en.wikipedia.org/wiki/Galois/Counter_Mode)

Your Security Zen

A study on malicious plugins in WordPress Marketplaces

https://securityaffairs.co/wordpress/135032/reports/wordpress-malicious-plugins.html

August 30, 2022 By Pierluigi Paganini

> *"YODA uncovered 47,337 malicious plugins on 24,931 unique websites. Among these, $41.5K had been spent on 3,685 malicious plugins sold on legitimate plugin marketplaces. Pirated plugins cheated developers out of $228K in revenues. Post-deployment attacks infected $834K worth of previously benign plugins with malware." reads the research paper. "Lastly, YODA informs our remediation efforts, as over 94% of these malicious plugins are still active today."*

More info:

Good libraries have good defaults

Libsodium documentation

GitHub repository Download Quickstart Libhydrogen

Search...

- Introduction
- Installation
- Quickstart and FAQ
- Projects using libsodium
- Commercial support
- Bindings for other languages
- Usage
- Helpers
- Padding
- Secure memory
- Generating random data
- Secret-key cryptography
  - Authenticated encryption
  - Encrypted streams and file encryption
  - Encrypting a set of related messages

Authenticated encryption

Example

```
#define MESSAGE ((const unsigned char *) "test")
#define MESSAGE_LEN 4
#define CIPHERTEXT_LEN (crypto_secretbox_MACBYTES + MESSAGE_LEN)

unsigned char key[crypto_secretbox_KEYBYTES];
unsigned char nonce[crypto_secretbox_NONCEBYTES];
unsigned char ciphertext[CIPHERTEXT_LEN];

crypto_secretbox_keygen(key);
randombytes_buf(nonce, sizeof nonce);
crypto_secretbox_easy(ciphertext, MESSAGE, MESSAGE_LEN, nonce, key);

unsigned char decrypted[MESSAGE_LEN];
if (crypto_secretbox_open_easy(decrypted, ciphertext, CIPHERTEXT_LEN, nonce, key) != 0)
    /* message forged! */
}
```