CSE 127 Computer Security

Stefan Savage, Fall 2025, Lecture 13

Cryptography I: Primitives

based on slides by Kirill Levchenko, Stefan Savage and Alex Gantman

Today

Change of Focus

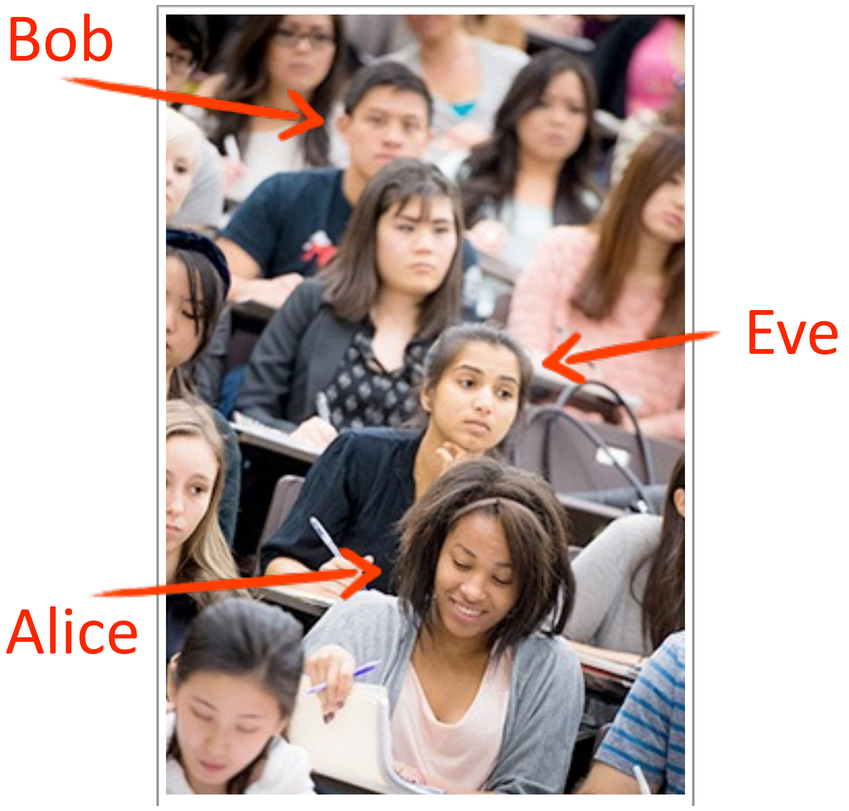
- Thus far we have largely focused on low-level security issues on a machine (i.e., how we try to protect ourselves from attacks on code or the OS)

- Today we're going to start looking at cryptography
  - Means for providing (principally) confidentiality and integrity/authenticity across trust boundaries

Cryptography: A Simple Example

Alice, Bob, Eve in a classroom
(back when we had those)

Alice and Bob can pass notes
… but only via Eve



A Simple Example

Alice wants to communicate to Bob whether or not she will work with him on CSE 127 Assignment 2

- 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

A Simple Example

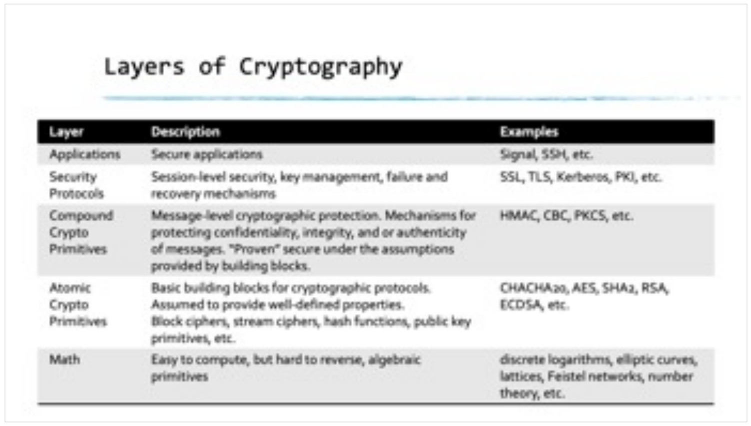
How can Alice communicate with Bob without Eve knowing her answer?

How can Bob know that the message is really from Alice without Eve being able to modify or spoof it?

Cryptography

Cryptography provides mechanisms for enforcing confidentiality and integrity across time and space controlled by an adversary

Very broad subject



Layers of Cryptography

Layer	Description	Examples
Applications	Secure applications	Signal, SSH, etc.
Security Protocols	Session-level security, key management, failure and recovery mechanisms	SSL, TLS, Kerberos, PKI, etc.
Compound Crypto Primitives	Message-level cryptographic protection. Mechanisms for protecting confidentiality, integrity, and or authenticity of messages. "Proven" secure under the assumptions provided by building blocks.	HMAC, CBC, PKCS, etc.
Atomic Crypto Primitives	Basic building blocks for cryptographic protocols. Assumed to provide well-defined properties. Block ciphers, stream ciphers, hash functions, public key primitives, etc.	CHACHA20, AES, SHA2, RSA, ECDSA, etc.
Math	Easy to compute, but hard to reverse, algebraic primitives	discrete logarithms, elliptic curves, lattices, Feistel networks, number theory, etc.

We focus primarily on using it as a tool in designing secure systems.

– CSE107/207 goes deeper into design of compound primitives and security protocols

Motivation

Two parties want to communicate securely

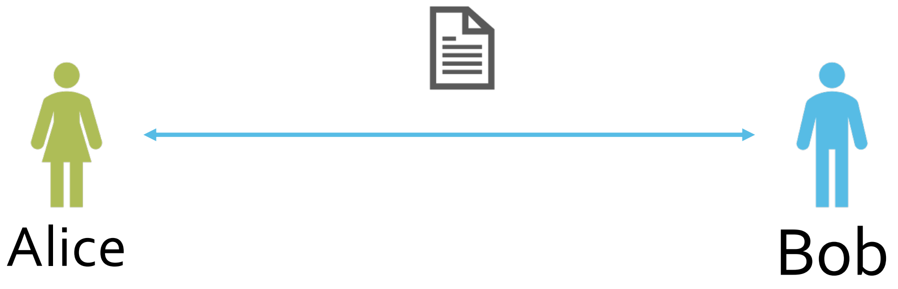
- Confidentiality: no one else can read messages
- Integrity: messages cannot be modified
- Authenticity: parties cannot be impersonated

Example: Military orders

- Enemy can't know what the orders say
- Enemy can't modify the orders
- Enemy can't send fake orders

Setting

Alice and Bob communicate by sending messages



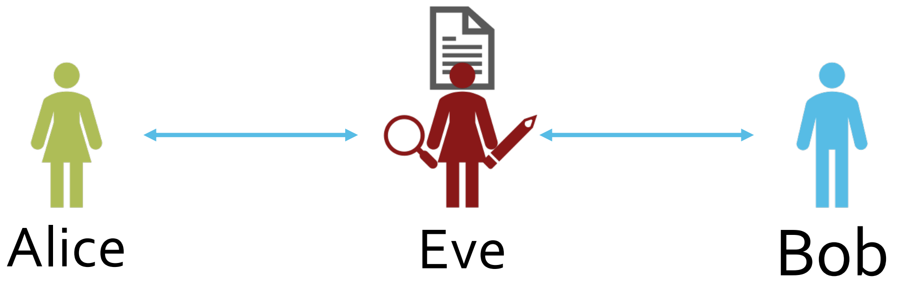
Setting

Alice and Bob communicate by sending messages

Eve can read, create, modify, or block messages

Attacker model determines Eve's control of the channel between Alice and Bob

- ***Passive*** attacker: can read only
- ***Active*** attacker: can read, create, and possibly modify, block
- ***Person-in-the-middle*** (PitM) attacker: can read, create, modify, block



Back to our Simple Example

Let's say Alice and Bob *pre-agree* on a secret code:

- "eagle": "yes"
- "snake": "no"

Alice sends message "eagle"

- Bob knows this means yes
- Eve learns nothing because she doesn't know the code

Back to our 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 confidentiality?

- Yes, but only for a single message!

Integrity? Authenticity?

The Basics

Know your threat model!

Know whether you need to protect ***confidentiality***, ***integrity***, or both.

Confidentiality and integrity are protected by different cryptographic mechanisms!

Having one does not imply the other!!!

The Basics

Know your threat model!

Know whether you need protection against a ***passive*** or ***active/PitM*** adversary.

Systems that are secure against the former may not be secure against the latter.

Encryption

We usually want to encrypt more than one bit of information

- In general — binary string of arbitrary length
- Cipher: *mechanical algorithm for transforming plaintext to/from ciphertext*

Plaintext (m): unencrypted message to be communicated

- From now on, assume this is a binary string

Ciphertext (c): encrypted version of message

- Also a binary string (may not be same length as plaintext)

$$c = E(m)$$

$$m = D(c)$$

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?

$$c = \underbrace{m}_{\text{Plaintext (a binary string)}} \oplus \underbrace{r}_{\text{Random binary string of the same length as plaintext}}$$

One-Time Pad

For a given ciphertext, every plaintext is **equally probable**

- Probability taken over random choice of pad r *(i.e., the key)*

One bit yes/no protocol as a one-time pad:

- Plaintext: yes $\longrightarrow$ `1`, no $\longrightarrow$ `0`
- Ciphertext (if r = `0`): `0 → 0, 1 → 1`
- Ciphertext (if r = `1`): `0 → 1, 1 → 0`

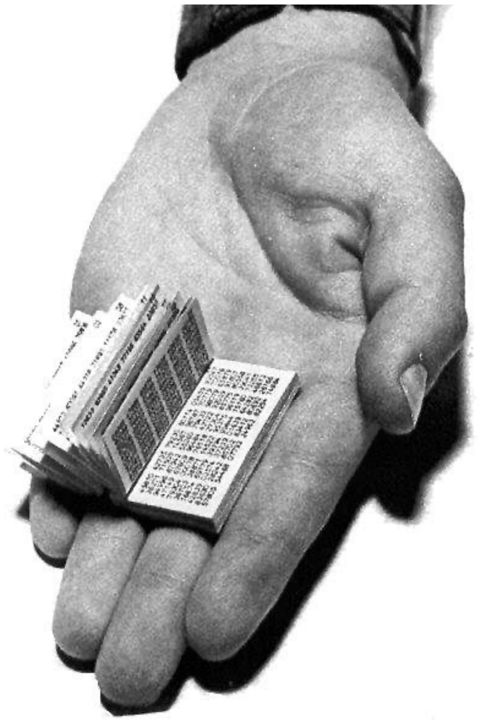
$$c = m \bigoplus r$$

One-Time Pads

Perfect secrecy.

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

No integrity or authenticity



http://www.ranum.com/security/computer_security/papers/otp-faq/otp.jpg

Computational Cryptography

Sharing large secrets is impractical

Modern cryptographic systems depend on small(-er) secrets

But, if the pre-arranged secret is smaller than the message, then not all plaintexts are equally probable

- Ciphertext reveals some information about plaintext

Practical cryptography must sacrifice perfect secrecy

- It’s no longer **impossible** to learn anything about the plaintext from the ciphertext
  … just computationally impractical for the adversary without the secret
  … we hope

Computational Cryptography

Kerckhoffs's Principle: A cryptosystem should be secure even if everything about the system, except the [secret] key, is public knowledge.

(related) Shannon's Maxim: "the enemy knows the system",

- i.e., "one ought to design systems under the assumption that the enemy will immediately gain full familiarity with them"

Assume all details of the algorithm are public

- **Only the key is secret**
- No *reliance* on "security through obscurity"

Cryptographic Primitives

Symmetric Cryptography

- Alice and Bob share a ***secret key*** that they use to secure their communications
- Secret keys are random bit-strings

- aka Secret-Key or Shared-Key Cryptography

Cryptographic Primitives

Asymmetric Cryptography

- Each subject has two keys: Private and Public
- ***Public keys*** can be used by anyone for “unprivileged” operations
  - Encrypt message for intended receiver. Verify signature
- ***Private keys*** are secret and used for “privileged” operations
  - Decrypt message. Sign message
- Public and private key parts are related in algorithm-dependent way
  - Can’t just pick a random bit-string as your key as with symmetric keys
  - Need a **key-generation function**.

- aka Public-Key Cryptography

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 107/207, Number Theory, etc.

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 [atomic]:
1 part arcane magic and folk superstition +
2 parts bitter experience of past failures

- When a primitive gets broken — move on to another one
- E.g. AES replaces DES, SHA-3 standardized to replace SHA-2, etc

Asymmetric cryptographic primitives [atomic]:
based on computational complexity of certain problems

- Breaking one means a breakthrough in solving a hard mathematical problem
- Have (mostly) weathered the test of time better

Cryptographic Primitives

Encryption: provides confidentiality, without integrity protection.

- Formally: adversary can't* distinguish which of the two plaintexts were encrypted without knowing the [secret] key.

  * within practical computational bounds.

- Does not provide integrity protection!

  Changes to ciphertext may lead to predictable changes in decrypted plaintext.

  Needs separate message authentication.

Message Authentication Code (symmetric) and ***Digital Signature*** (asymmetric): provides integrity, without confidentiality.

- Formally: adversary can't* generate a valid MAC or signature for a new message without knowing the [secret] key.

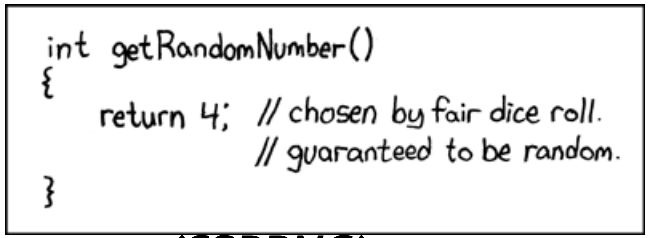
  * within practical computational bounds.

- Does not provide confidentiality!

  Needs separate message encryption.

Aside: Cryptographic Randomness

Almost all cryptography relies on good random numbers.



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

What is random?

- Uniformly distributed?
- Unpredictable?

Cryptographically Secure Pseudo-Random Number Generator (CSPRNG) requirements:

- Given the first *n* bits of a random sequence, can't* predict (*n*+1)th bit with probability better than ½.
- Even if internal state has been revealed (or guessed correctly), can't* reconstruct the stream of random numbers prior to the revelation.

  * within practical computational bounds.

Pay attention to randomness requirements when using cryptographic APIs

- Need to use the system APIs for cryptographic random numbers (rand() won't cut it)

https://xkcd.com/221/

Aside: Brute Force

All modern cryptography is breakable by brute force given enough ***knowledge about plaintext***

Try to decrypt ciphertext with every possible key until expected 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 (worst case)

Hash Functions

A ***cryptographic hash function*** maps arbitrary length input into a fixed-size string and has the following properties:

Pre-image Resistance

– Given a specific hash function output, it is impractical to find an input (pre-image) that generates the same given output.

Collision Resistance

– It is impractical to find any two inputs that hash to the same output.

$$h = H(m)$$

Hash Functions

Example: SHA-2: Secure Hash Algorithm 2

- Designed by NSA
- Output: 224, 256, 384, or 512 bits
- Previous hash functions (SHA-1, MD5) now considered weak (e.g. computationally feasible to generate collisions)

SHA-3: Secure Hash Algorithm 3

- Result of NIST SHA-3 contest (original candidate name: Keccak)
- Output: arbitrary size
- Recommended for new applications and replacement if SHA-2 broken

Message Authentication Codes (MACs)

Goal: Validate message integrity and authenticity based on a shared secret.

- How can Bob know that the message is really from Alice and has not been modified or spoofed by Eve?

MAC: Message Authentication Code

- Function of message **and secret key.**
- Impractical to forge without knowing the key.
    - i.e. to come up with a valid MAC for a new message.

$$a = MAC_k(m)$$

Message Authentication Codes (MACs)

Alice sends $m||a$ (|| *means "concatenated with")*

Bob uses his copy of the secret key k to independently compute a' on m and compare to the one received.

Note, **no confidentiality guarantees**.

$$a = MAC_k(m)$$

Message Authentication Codes (MACs)

MACs can be constructed out of hash functions or ciphers

HMAC: MAC based on hash function

- HMAC-SHA2: HMAC construction using SHA-2
- What's wrong with just using a hash of the message $H(m)$ as a MAC?
- What about $H(k||m)$? (The operator "||" here stands for concatenation)

  Many hash functions susceptible to length extension attack;
  attacker can create valid H(x||y) from H(x) without knowing x, just its length

  Don't make up your own MAC constructions!!!

Cipher-based MACs covered briefly later

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

Symmetric Encryption

$$c = E_k(m)$$

Encryption function

shared secret key

$$m = D_k(c)$$

Decryption function

shared secret key

Symmetric Ciphers

Stream cipher: generate a pseudorandom string of bits as long as the plaintext and XOR w/plaintext

- Pseudorandom: hard to tell apart from random
  - Hard: computationally hard to distinguish from random
- Can’t reuse string of bits (remember one-time pad!)

Block cipher: Encrypt/decrypt fixed-size blocks of bits

- Need a way to encrypt longer or shorter messages

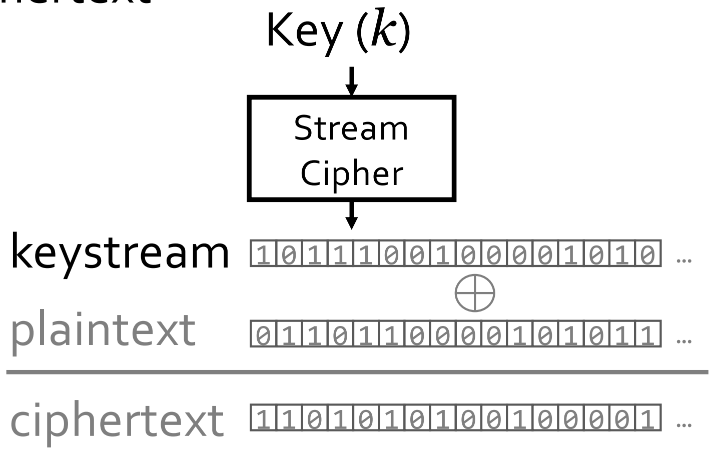
Stream Ciphers

Produces a pseudorandom ***keystream***

- Each key results in a unique, pseudorandom keystream

To encrypt, keystream is XORed with plaintext

To decrypt, keystream is XORed with ciphertext



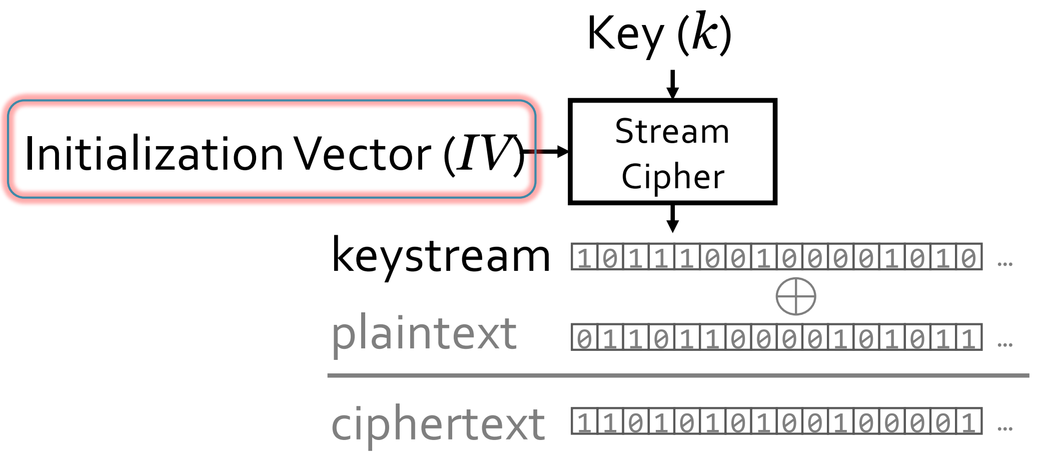
Stream Ciphers

Insecure if key used more than once

- Need mechanism to either
    - generate different one-time keys from a master key, or
    - a random **initialization vector** on each use (sometimes also called a nonce)

Example: ChaCha20

- 256-bit keys
- 96-bit ***Initialization Vector***



Block Ciphers

Block ciphers operate on fixed-size blocks

- Common sizes: 64 and 128 bits

A block cipher is typically a combination of **permutation**

- Each input mapped to exactly one output

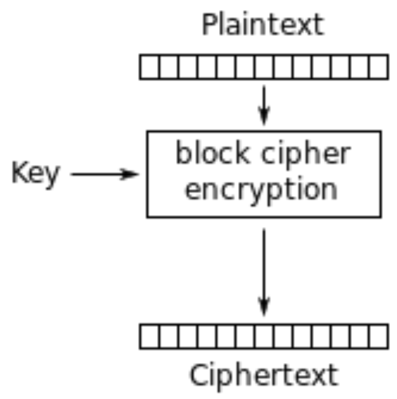
... and **substitution**

- Some codewords mapped to other codewords

Typically in *multiple rounds*

Example: AES: Advanced Encryption Standard

- Replacement for DES based on Rijndael cipher
- Key size: 128, 192, 256 bits
- Block size: 128 bits



https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Block Ciphers

Block ciphers encrypt/decrypt fixed-size blocks.

How to encrypt a message shorter than a block?

- Pad plaintext to full block size
- Must be able to *unambiguously distinguish padding from plaintext*
- **Don’t make up your own padding scheme!**

How to encrypt a message longer than a block?

- “Chain” individual blocks
- Methods of chaining are known as ***modes of operation***.
  There are a bunch of choices, I’ll describe three of the best known

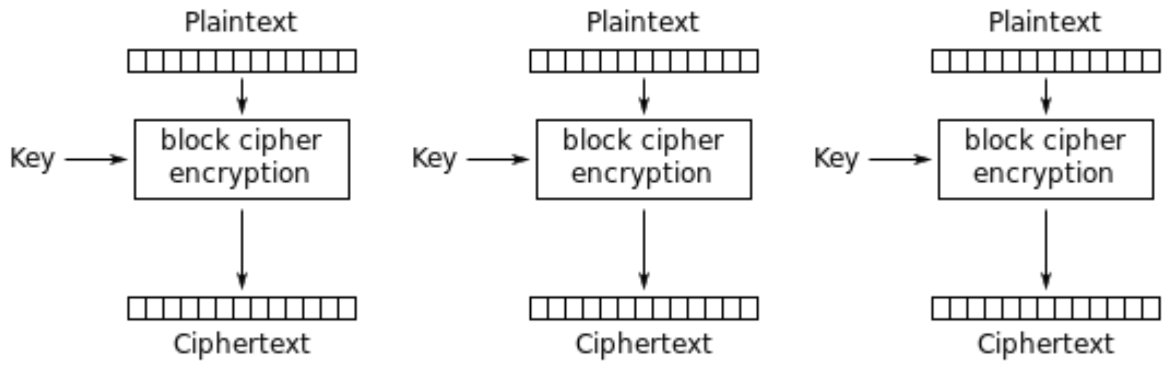
Electronic Code Book (ECB) Mode

Naïve mode of operation: encrypt each block separately

- As if looking it up in a code book. Known as Electronic Code Book (ECB) mode
  Very fast (easy to parallelize)

DO NOT USE without very good reason!!!

- why?

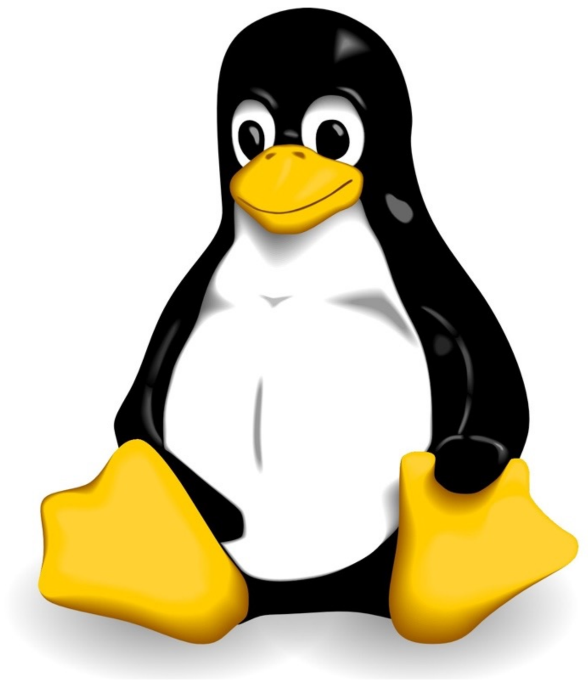


Electronic Codebook (ECB) mode encryption

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Electronic Code Book (ECB) Mode

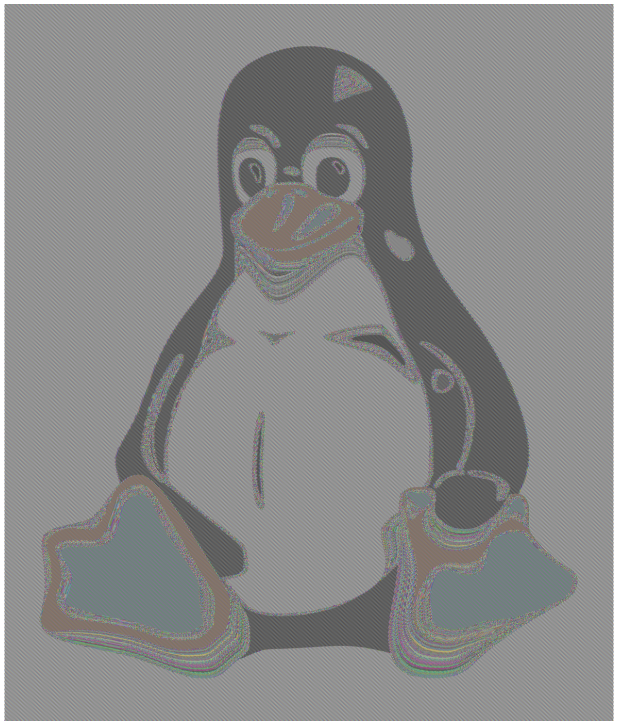
What if we encrypt this bitmap picture in ECB mode?



https://www.macitynet.it/linux-quello-ancora-non-funziona-ce-sistemare/

Electronic Code Book (ECB) Mode

What if we encrypt this bitmap picture in ECB mode?



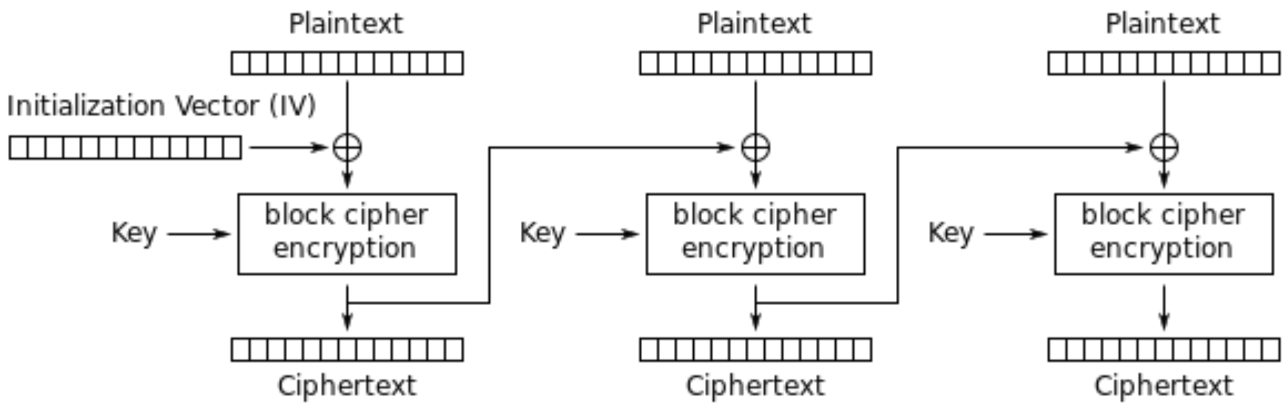
https://blog.filippo.io/the-ecb-penguin/

Cipher Block Chaining (CBC) Mode

XOR ciphertext block into next plaintext

Use random IV

- Subtle attack possible if attacker knows IV, controls plaintext



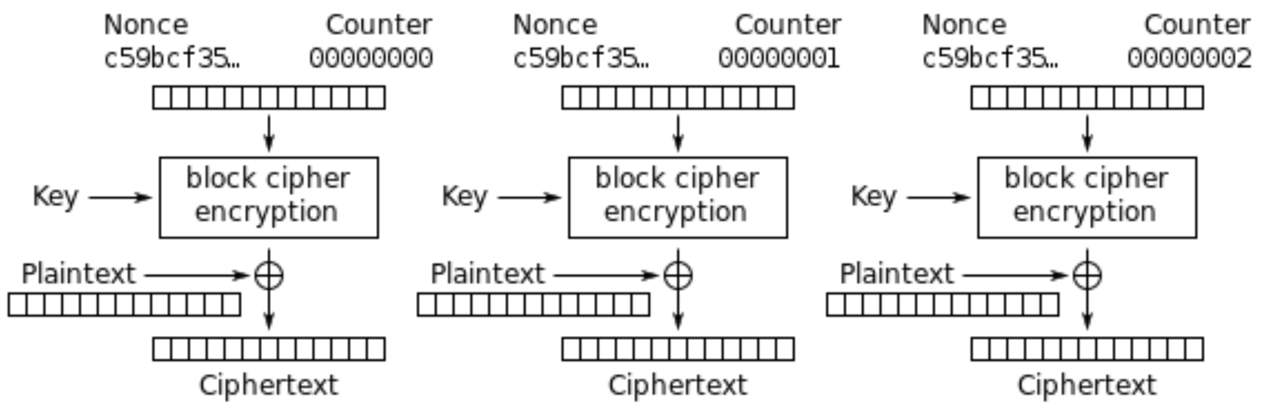
Cipher Block Chaining (CBC) mode encryption

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Counter (CTR) Mode

Encrypt successive counter values and XOR result with plaintext

Block cipher becomes stream cipher



Counter (CTR) mode encryption

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Cipher Properties

Encryption and decryption are inverse operations:

$$m = D_k(E_k(m))$$

Informally: ciphertext reveals nothing about plaintext

- More formally: can't distinguish which of two plaintexts were encrypted without key
- Is ECB secure under this definition?
  $E_k(m_0||m_0)$ is trivially distinguishable from $E_k(m_0||m_1)$

Non-property: integrity

- May be possible to change decrypted plaintext in known way
- Needs separate message authentication

Key hygiene:

- Do not use same key with different modes
  (or for separate encryption and authentication operations)

Authenticated Encryption

Many real-world systems were broken because encryption and authentication were combined in insecure ways.

Authenticated encryption simultaneously provides confidentiality, integrity, and authenticity.

- Designed to work with single key.

Rule of thumb: If you have a job in which you have to select a symmetric-key cipher suite (in the next year),
maybe use one of the following:

- ***AES-GCM*** (Galois/Counter Mode)
- ***ChaCha20+Poly1305AES***

Limitations of Symmetric Cryptography

We can now protect confidentiality and integrity of messages without sharing very large secrets.

But…

- We still need to securely establish **pairwise secret keys** between all parties
- Challenging in practice

Asymmetric Cryptography

aka ***Public Key Cryptography***

Two separate keys: ***public key*** and ***private key*** (secret)

Public key ***known to everyone.***

- Given Alice's public key
  - Anyone can send an encrypted message to Alice.
  - Anyone can verify that a message was signed by Alice.

Private key is kept secret.

- Only Alice can decrypt messages encrypted with her public key.
- Only Alice can sign messages so that they can be verified with her public key.

Asymmetric Primitives

Confidentiality: encryption and decryption.

Integrity and Authenticity: signing and verification.

Asymmetric Cryptography

Each subject 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: public key
- k: private key
- r: random bits.

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

Asymmetric 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!

https://commons.wikimedia.org/wiki/File:Address-book.svg

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 are fixed size.

Classic Asymmetric Ciphers

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 (ish)

Classic 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 (ish)

Practical Considerations

Asymmetric cryptography operations generally **much** more expensive than symmetric operations

- Both in compute time
- And key size

Asymmetric primitives operate on fixed-size messages

Usually combined with symmetric crypto for performance

- Use asymmetric to bootstrap ephemeral secret

Example use of combined symmetric/asymmetric encryption

Generate an ephemeral (one time) symmetric secret key (i.e., random)

Encrypt message using this ephemeral secret key

Encrypt ephemeral key using asymmetric encryption

Send encrypted message and encrypted ephemeral key

Decryption: decrypt ephemeral key, use it to decrypt message

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

Example use of combined symmetric/asymmetric signing

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

Summary: Symmetric Primitives

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

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

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

Summary: Asymmetric Primitives

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

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

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

Review

Confidentiality and integrity are protected by different cryptographic mechanisms!

- Having one does not imply the other!!!

Kerckhoffs's Principle: A cryptosystem should be secure even if everything about the system, except the [secret] key, is public knowledge.

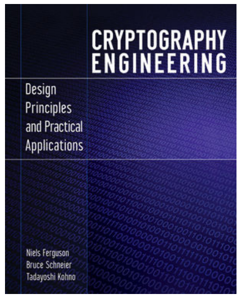
Use existing methods and tools. If possible, use existing applications...

- Do not descend into lower layers unless you are an expert and brought back up
- Same goes for implementation. Do not modify or re-implement cryptographic libraries. Minor changes can lead to catastrophic failure.
  - Even if functionality is not affected (i.e. produces "correct" results)

Additional Resources

Cryptography Engineering by Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno

– https://www.goodreads.com/book/show/7602360-cryptography-engineering



NIST Cryptographic Standards and Guidelines

– https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines

Cryptographic Right Answers

– http://latacora.singles/2018/04/03/cryptographic-right-answers.html

A Few Thoughts on Cryptographic Engineering by Matthew Green

– https://blog.cryptographyengineering.com/

Additional Resources

Introduction to Modern Cryptography by Mihir Bellare

- AKA CSE 107/207
- https://cseweb.ucsd.edu/~mihir/cse107/classnotes.html
- https://cseweb.ucsd.edu/~mihir/cse207/classnotes.html
- https://cseweb.ucsd.edu/~mihir/papers/gb.html
- http://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf