

CSE 127 Computer Security

Stefan Savage, Spring 2018, Lecture 7

Cryptography I: Primitives

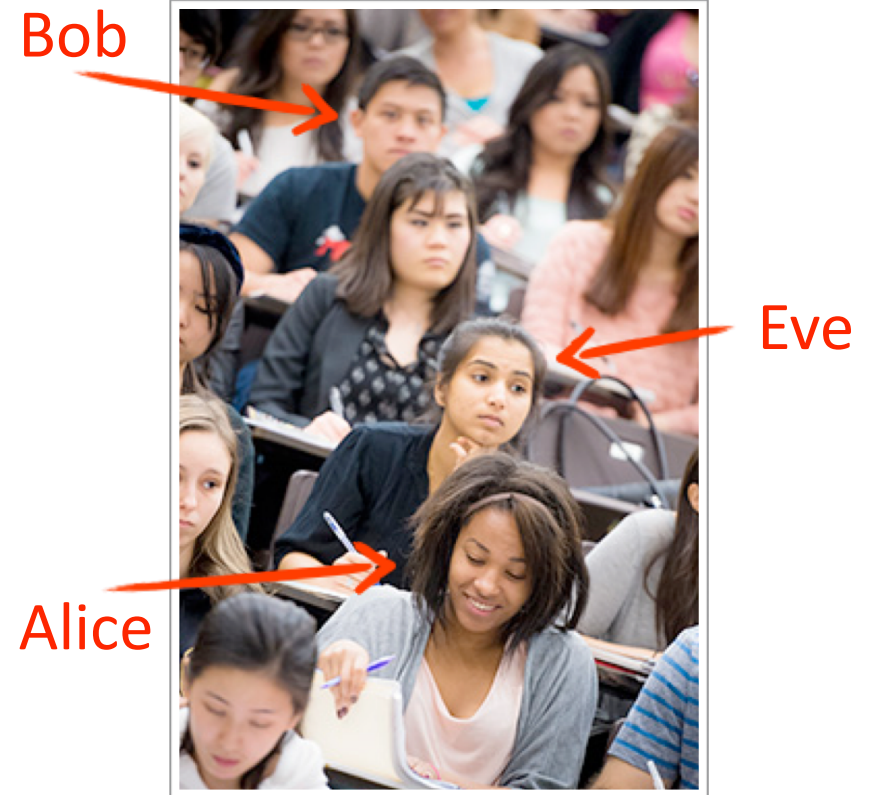
based on slides by Kirill Levchenko

Class Logistics

- Project 2 was due yesterday
- Project 3 is assigned. Due 4/30 @ 10pm
 - Exploiting side channel vulnerabilities
- Midterm is on 5/3
 - Everything covered up until that point

Cryptography: A Simple Example

- Alice, Bob, Eve in a classroom like this one
- Alice and Bob can pass notes ... via Eve



A 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

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.

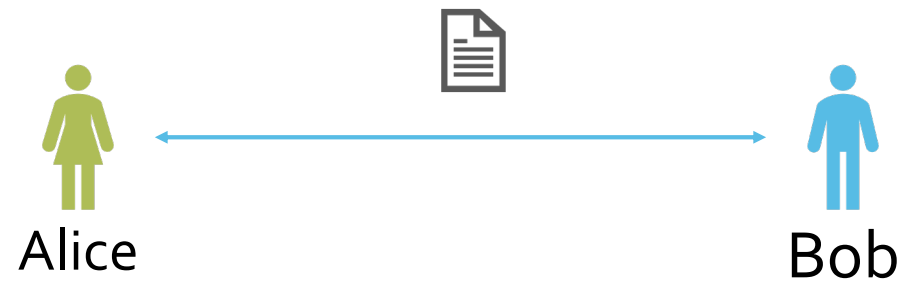
- We focus primarily on using it as a tool in designing secure systems.
 - CSE107/207 goes deeper into design of compound primitives and security protocols

Motivation

- Two parties want to communicate securely
 - 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
- Other examples?

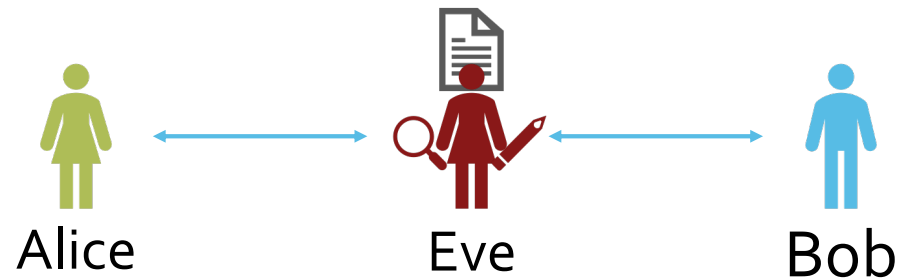
Setting

- 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
 - **Man-in-the-middle** (MitM) attacker: can read, create, modify, block



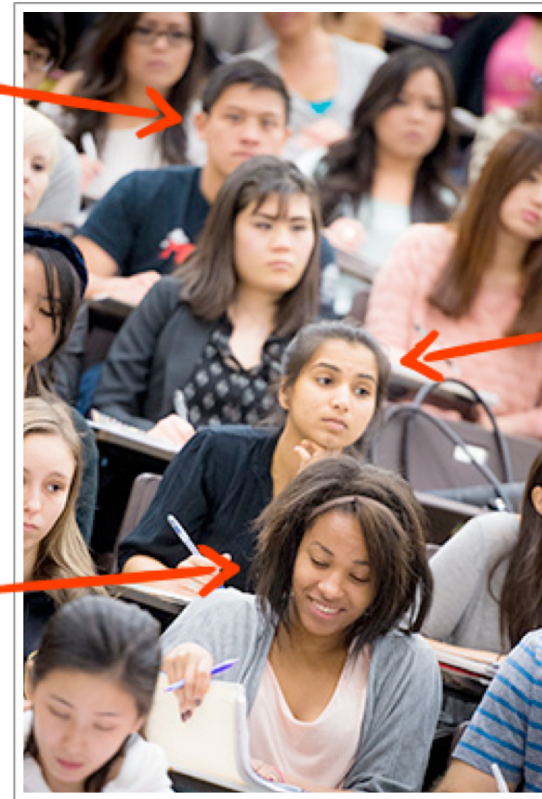
A Simple Example

- 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

Bob

Alice

Eve



A 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?
 - 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/MitM* 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
- 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?

$$c = m \oplus r$$

Plaintext (a binary string)

Random binary string of the same length as plaintext

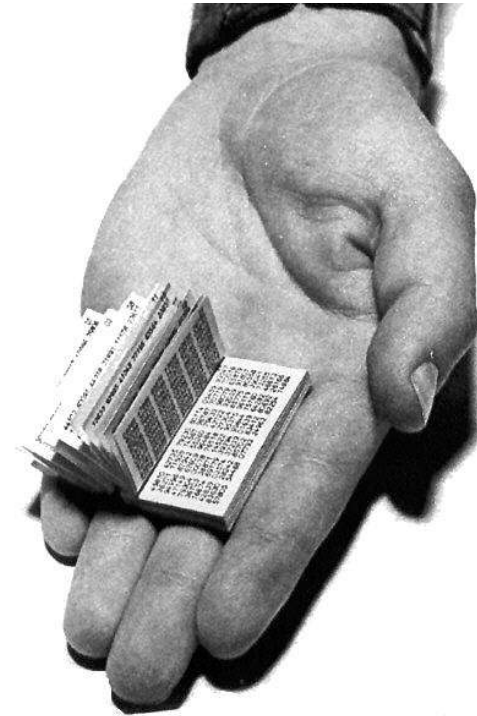
One-Time Pad

- For a given ciphertext, every plaintext is equally probable
 - Probability taken over random choice of pad r
- Eagle-Snake protocol as a one-time pad:
 - Plaintext: yes $\rightarrow 1$, no $\rightarrow 0$
 - Ciphertext: “eagle” $\rightarrow 1$, “snake” $\rightarrow 0$
 - If $r = 0$: “eagle” = yes, “snake” = no
 - If $r = 1$: “eagle” = no, “snake” = yes

$$c = m \oplus r$$

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.



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 has to 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
 - NIST developing SHA-3 for when SHA-2 is broken
- Asymmetric cryptographic primitives [atomic]:
 - based on computational complexity of certain problems
 - Breaking one means breakthrough in solving hard problem
 - Have 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.
 - Need separate message authentication.

Cryptographic Primitives

- ***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!
 - Need separate message encryption.

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

Good News and Bad News

- “Encryption works. Properly implemented strong crypto systems are one of the few things that you can rely on.”
 - Edward Snowden
- “Crypto will not be broken. It will be bypassed.”
 - Adi Shamir

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 given output.
- ***Collision Resistance***
 - It is impractical to find two inputs that hash to the same output.
- How do they compare to properties of encryption?

$$h = H(m)$$

Hash Functions

- SHA-2: Secure Hash Algorithm 2
 - Designed by NSA
 - Output: 224, 256, 384, or 512 bits
 - Recommended for use today
 - Do not use older, obsolete SHA-1 or MD5!!!

Hash Functions

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

Message Authentication Codes (MACs)

- Goal: Validate message integrity 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 $m||a$.
- 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.

$$a = \text{MAC}_k(m)$$

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 $H(k||m)$?
 - Don't make up your own MAC constructions!!!
- Cipher-based MACs covered 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 pad (remember one-time pad!)
- Block cipher: Encrypt/decrypt fixed-size blocks of bits
 - Need a way to encrypt longer or shorter messages

Cryptographic Randomness

- Cryptography relies on good random numbers.

- What is random?

- Uniformly distributed?
- Unpredictable?

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

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

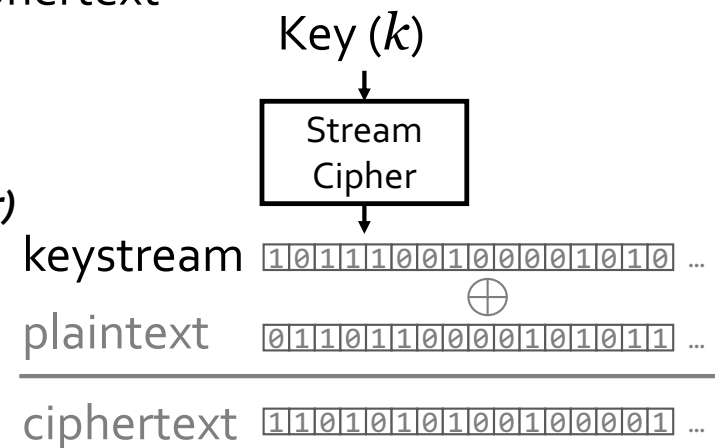
- Given the first n bits of a random sequence, can't* predict $(n+1)^{\text{th}}$ bit with probability better than $\frac{1}{2}$.
- If internal state has been revealed (or guessed correctly), can't* reconstruct the stream of random numbers prior to the revelation.
 - * within practical computational bounds.

Cryptographic Randomness

- Cryptography relies on good random numbers.
 - Very common
 - Pay attention to randomness requirements when using cryptographic APIs.
- Need to know which library/system APIs return random numbers suitable for cryptographic use.
 - Do not seed PRNGs (exclusively) with low-entropy inputs, like date/time or process id.

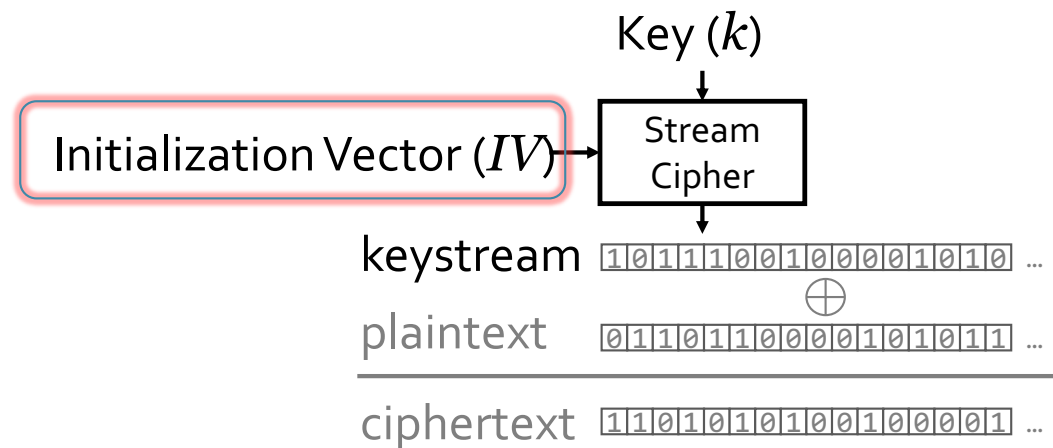
Stream Ciphers

- 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
- Example: ChaCha20
 - 256-bit keys
 - 96-bit **nonce** (*also called Initialization Vector*)



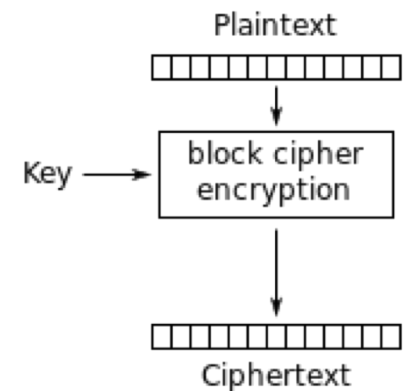
Stream Ciphers

- Insecure if key used more than once
 - Need mechanism to generate one-time keys from master
 - Or a random initialization value on each use
- As with one-time pad, attacker can change plaintext in controlled manner (malleability)



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
- Example: AES: Advanced Encryption Standard
 - Replacement for DES based on Rijndael cipher
 - Key size: 128, 192, 256 bits
 - Block size: 128 bits

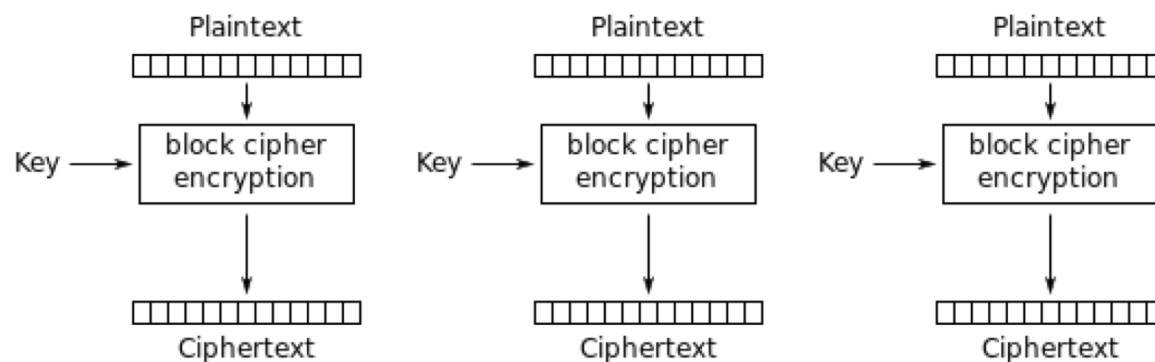


Block Ciphers

- 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*.

Electronic Code Book (ECB) Mode

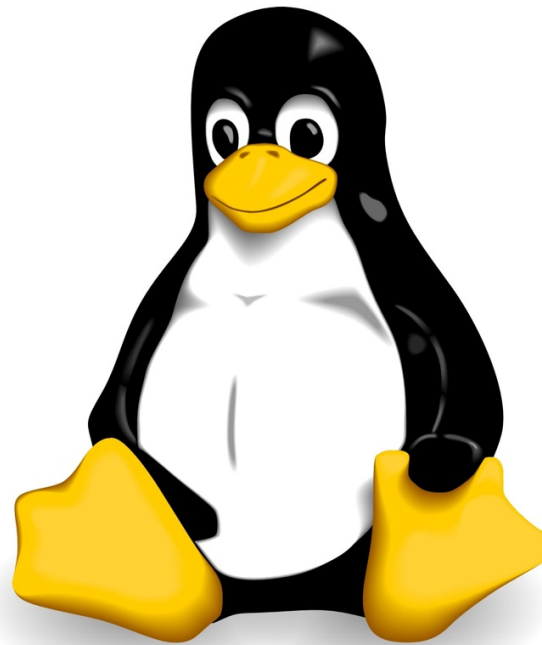
- Naïve mode of operation: encrypt each block separately
 - As if looking it up in a code book. Known as Electronic Code Book (ECB) mode.
- **DO NOT USE!!!**
 - why?



Electronic Codebook (ECB) mode encryption

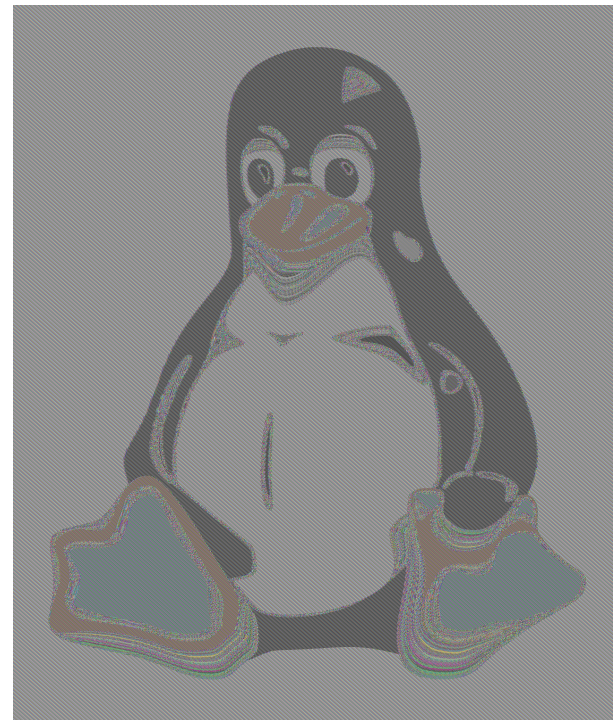
Electronic Code Book (ECB) Mode

- What if we encrypt this bitmap picture in ECB mode?



Electronic Code Book (ECB) Mode

- What if we encrypt this bitmap picture in ECB mode?



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

Cipher Properties

- Encryption and decryption are inverse operations:

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

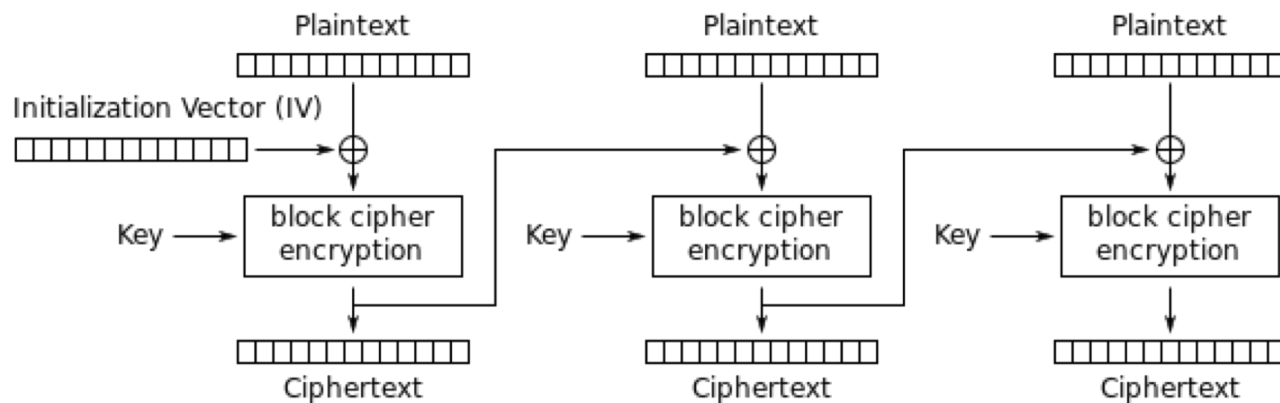
- Informally: ciphertext reveals nothing about plaintext
 - More formally: can't distinguish which of two plaintexts were encrypted without key
 - Is ECB secure under this definition?
 - $E_k(m_o || m_o)$ is trivially distinguishable from $E_k(m_o || m_1)$
- Non-property: integrity
 - May be possible to change decrypted plaintext in known way
 - Need separate message authentication

Block Cipher Modes of Operation

- How else can we “chain” multiple blocks together?
- Padding
- Initialization Vector (iv)
- Tradeoffs
 - Parallelizable
 - Random access
 - Need decryption function?

Cipher Block Chaining (CBC) Mode

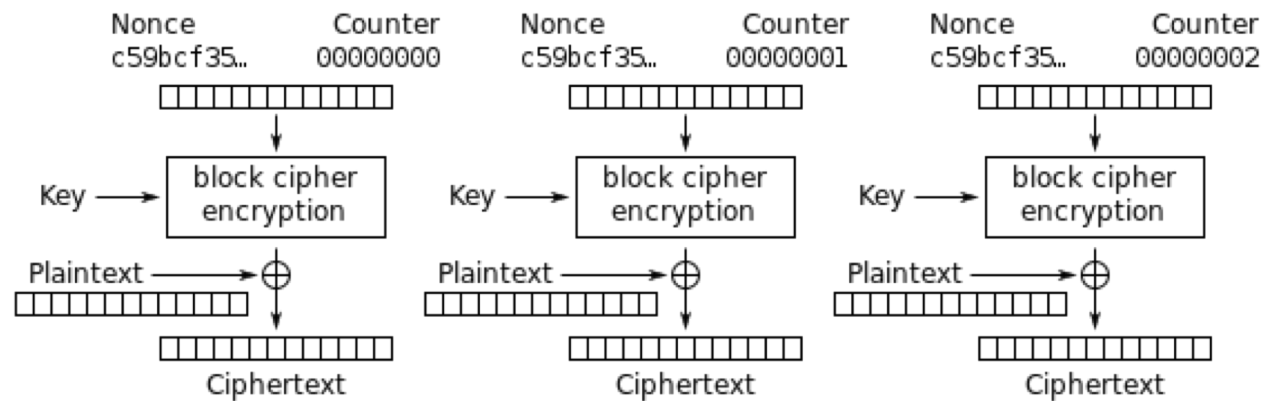
- XOR ciphertext block into next plaintext
- Use random IV
 - Subtle attack possible if attacker knows IV, controls plaintext



Cipher Block Chaining (CBC) mode encryption

Counter (CTR) Mode

- Encrypt successive counter values and XOR result with plaintext
- Block cipher becomes stream cipher



Counter (CTR) mode encryption

Cipher-based MACs

- CMAC
- Poly1305

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 to select a symmetric-key cipher suite (in the next year), 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 establish pairwise keys

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.

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

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!



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.

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

Table 2: Comparable strengths

Security Strength	Symmetric key algorithms	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC (e.g., ECDSA)
≤ 80	2TDEA ²¹	$L = 1024$ $N = 160$	$k = 1024$	$f = 160-223$
112	3TDEA	$L = 2048$ $N = 224$	$k = 2048$	$f = 224-255$
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256-383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384-511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$

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
-
- Decryption: decrypt ephemeral key, decrypt message

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

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

Summary: Symmetric Primitives

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

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

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

Summary: Asymmetric Primitives

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

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

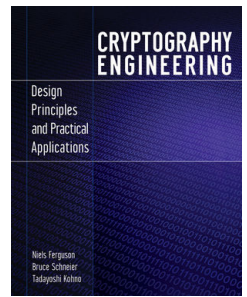
$$M = D_k(C) \quad V_K(M, S)$$

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 effected (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>

Next time

- Key distribution and side channels