Cryptography

Prep-course, 9/30/25

Eike Kiltz

Slide deck based on Nadim’s Applied cryptography course: https://appliedcryptography.page/





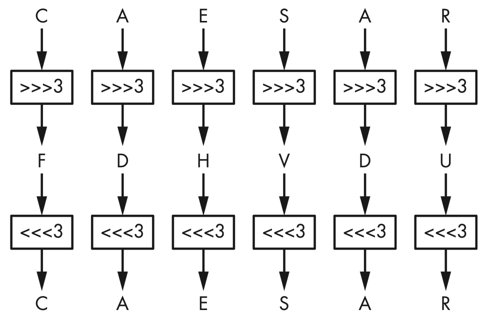

Cryptography in Bochum

- Asymmetric cryptography
- Cryptanalysis
- Cryptographic engineering
- Symmetric cryptography

Defining cryptography

What is Cryptography?

"The science of enabling secure and private computation, communication, verification, and delegation in the presence of untrusted parties, adversarial behavior, and mutually distrustful participants."



Source: Serious Cryptography, 2nd Edition

Pull out your phone!

Let's count the cryptographic operations happening right now:

- WiFi connection (WPA3)
- Cellular connection (5G AES)
- App notifications (TLS)
- Face/Touch ID (Secure Enclave)
- Background app refreshes

Real-time calculation

- Average smartphone: 100+ crypto operations/second
- In this 75-minute class: 450,000+ operations
- By end of semester: Billions of operations

You're already a crypto user!

Cryptography is everywhere

- Banking
- Buying stuff from the store
- Any digital payment system
- Messaging (WhatsApp, Signal, iMessage, Telegram)
- Voice calls
- Government and military systems
- SSH
- VPN access
- Visiting most websites (HTTPS)
- Disk encryption
- Cloud storage
- Video conferencing
- Unlocking your (newer) car
- Identity card systems
- Ticketing systems
- DRM solutions
- Private contact discovery
- Cryptocurrencies
- That Apple Photos feature that detects similar photos

Cryptographic building blocks

Components

- Cryptography manifests as a set of primitives, from which we build protocols intended to accomplish well-defined security goals.
- **Primitives**: AES, RSA, SHA-2, DH…
- **Protocols**: TLS, Signal, SSH, FileVault 2, BitLocker…

Examples

- **AES**: Symmetric encryption
  - $\text{Enc}(k, m) = c$, $\text{Dec}(k, c) = m$.
- **SHA-2**: Hash function
  - $\text{H}(m) = h$.
- **Diffie-Hellman**: Public key agreement
  - Allows two parties to agree on a secret key k.

Cryptographic building blocks

Security goals

- **Confidentiality**: Data exchanged between Client and Server is only known to those parties.
- **Authentication**: If Server receives data from Client, then Client sent it to Server.
- **Integrity**: If Server modifies data owned by Client, Client can find out.

Examples

- **Confidentiality**: When you send a private message on Signal, only you and the recipient can read the content.
- **Authentication**: When you receive an email from your boss, you can verify it actually came from them.
- **Integrity**: Your computer can verify that software update downloads haven't been tampered with during transmission.

Security goals: more examples

- **TLS (HTTPS)** ensures that data exchanged between the client and the server is confidential and that parties are authenticated.
  - Allows you to log into gmail.com without your ISP learning your password.
- **FileVault 2** ensures data confidentiality and integrity on your MacBook.
  - Prevents thieves from accessing your data if your MacBook is stolen.
- **Signal and WhatsApp** implement post-compromise security, an advanced security goal.
  - Allows a conversation to “heal” in the event of a temporary key compromise.
  - More on that later in the course.

The magic of cryptography

Cryptography lets us achieve what seems impossible

- Secure communication over insecure channels
- Prove information is true without revealing it
- Proof of computation without redoing it

Hard problems

- Cryptography is largely about equating the security of a system to the difficulty of solving a math problem that is thought to be computationally very expensive.
- With cryptography, we get security systems that we can literally mathematically prove as secure (under assumptions).
- Also, this allows for actual magic.
    - Alice and Bob meet for the first time in the same room as you.
    - You are listening to everything they are saying.
    - Can they exchange a secret without you learning it?

The Modulo Operation

- $a \bmod n$ gives the remainder when dividing a by n
- Result is always in $\{0, 1, \ldots, n-1\}$
- **Even for negative numbers!**

$$21 \bmod 7 = 0$$

$$20 \bmod 7 = 6$$

$$-20 \bmod 7 = 1 \quad \text{(not -6!)}$$

- Think: "a is ($a \bmod n$) more than a multiple of n"

Time for actual magic

Setup

- Public parameters: $p = 13, g = 2$
- Alice picks secret: $a = 5$
- Bob picks secret: $b = 7$

Public Exchange

- Alice computes: $A = g^a \bmod p = 6$
- Bob computes: $B = g^b \bmod p = 11$
- Alice sends $A = 6$ to Bob
- Bob sends $B = 11$ to Alice

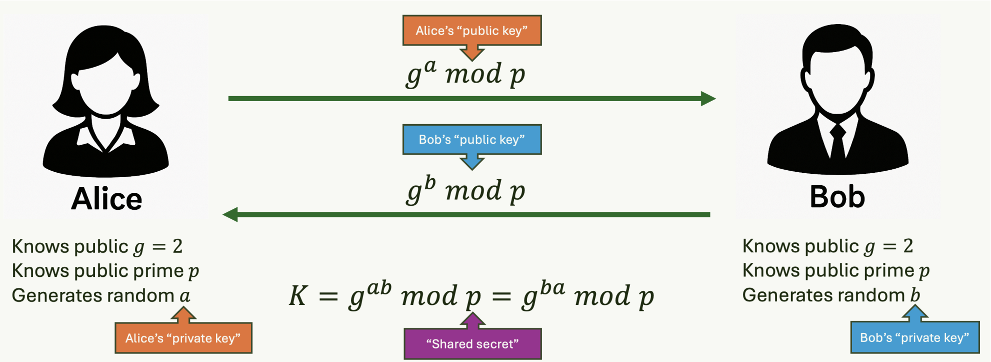
Shared Secret Computation

- Alice computes:
  $s = B^a \bmod p = 11^5 \bmod 13$
  - $= 161051 \bmod 13 = 9$
- Bob computes:
  $s = A^b \bmod p = 6^7 \bmod 13$
  - $= 279936 \bmod 13 = 9$
- **Shared secret: $s = 9$**

Eavesdropper sees only:

- $p = 13, g = 2, A = 6, B = 11$
- (In the real world, p, a and b are much larger numbers)

Time for actual magic



Why does this work?

- Listener only knows $g, p, g^a \bmod p, g^b \bmod p$
- Assuming a, b, p are large enough and p is prime...
- There is no known feasible computation to obtain K.

No known feasible computation

- The discrete logarithm problem:
  - Given a finite cyclic group G, a generator $g \in G$, and an element $h \in G$, find the integer x such that $g^x = h$
- In more concrete terms:
  - Let p be a large prime and let g be a generator of the multiplicative group $\mathbb{Z}_p^*$ (all nonzero integers modulo p).
  - Given:
    - $g \in \mathbb{Z}_p^*, h \in \mathbb{Z}_p^*$
    - Find $x \in \{0, 1, \ldots, p-2\}$ such that $g^x \equiv h \pmod{p}$
  - This problem is believed to be computationally hard when p is large and g is a primitive root modulo p.
    - "Believed to be" = we don't know of any way to do it that doesn't take forever, unless we have a strong, stable quantum computer (Shor's algorithm)

Signal's double ratchet: DH everywhere

- **Initial key exchange**: Uses X3DH (Extended Triple DH)
  - Combines **three** DH key exchanges for security.
  - Works even when recipient is offline (*"asynchronous"* protocol).[a]
- **Ongoing communication**: Uses Double Ratchet
  - New DH key exchange for every message!
  - Provides "forward secrecy" and "post-compromise security".
  - If your phone gets compromised today, yesterday's messages remain secure.
  - If your phone recovers from compromise, tomorrow's messages are secure again.

Signal uses DH key exchange dozens, hundreds of times per conversation.

[a] Everything on this slide will be covered in much more detail later in the course.

Hard problems

Asymmetric Primitives

- Diffie-Hellman, RSA, ML-KEM, etc.
- "Asymmetric" because there is a "public key" and a "private key" for each party.
- Algebraic, assume the hardness of mathematical problems (as seen just now.)

Symmetric Primitives

- AES, SHA-2, ChaCha20, HMAC...
- "Symmetric" because there is one secret key.
- Not algebraic but unstructured, but on their understood resistance to n years of cryptanalysis.
- Can act as substitutes for assumptions in security proofs!
  - Example: hash function assumed to be a "random oracle"

Symmetric primitive example: hash functions

Hash Function Properties

- Takes input of **any size**
- Produces output of **fixed size**
- Is **deterministic** (same input → same output)
- Even a **tiny change** in input creates completely different output
- Is **efficient** to compute

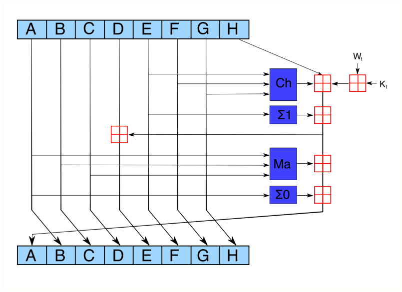
SHA256(hello) =
2cf24dba5fb0a30e26e83b2ac5
b9e29e1b161e5c1fa7425e7304
3362938b9824

SHA256(hullo) =
7835066a1457504217688c8f5d
06909c6591e0ca78c254ccf174
50d0d999cab0

Note: One character change → completely different hash!

Expected properties of a hash function

- **Collision resistance**: computationally infeasible to find two different inputs producing the same hash.
- **Preimage resistance**: given the output of a hash function, it is computationally infeasible to reconstruct the original input.
- **Second preimage resistance**: given an input and an output, it's computationally infeasible to find another different input producing the same output.



SHA-2 compression function. Source: Wikipedia

Hash functions: what are they good for?

- **Data integrity verification**: Hash a file. Later hash it again and compare hashes to check if the file has changed, suffered storage degradation, etc.
- **Proof of work**: Server asks client to hash something a lot of times before they can access some resource. Useful for anti-spam, Bitcoin mining, etc.
- **Zero knowledge proofs**: time for more actual magic

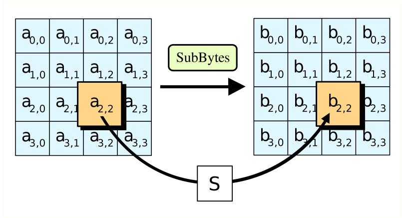
Time for more actual magic

- **Zero-knowledge proofs** allow you to prove that you know a secret without revealing any information about it.
- They built “zero-knowledge virtual machines” where you can execute an entire program that runs as a zero-knowledge proof.
- ZKP battleship game: server proves to the players that its output to their battleship guesses is correct, without revealing any additional information (e.g. ship location).

Battleship board game. Source: Hasbro

What about encryption?

- Symmetric primitive of choice for encryption: **AES**.
- Not that far off in terms of design process from hash functions, but:
  - AES is a PRP (pseudorandom permutation)
  - HMAC-SHA256 is a PRF (pseudorandom function)

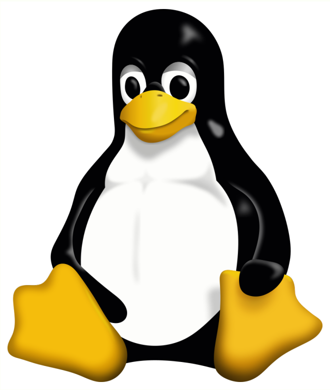


AES's SubBytes operation. Source: Wikipedia

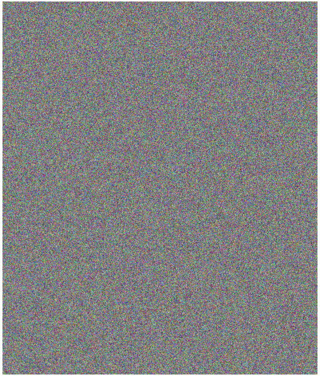
AES is a block cipher

- AES takes a 16-byte input, produces a 16-byte output.
- Key can be 16, 24 or 32 bytes.
- OK, so what if we want to encrypt more than 16 bytes?
- **Proposal**: split the plaintext into 16 byte chunks, encrypt each of them with the same key.

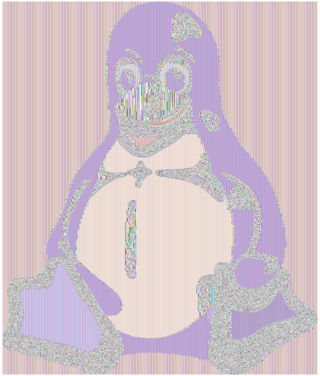
Block cipher examples



What we start with

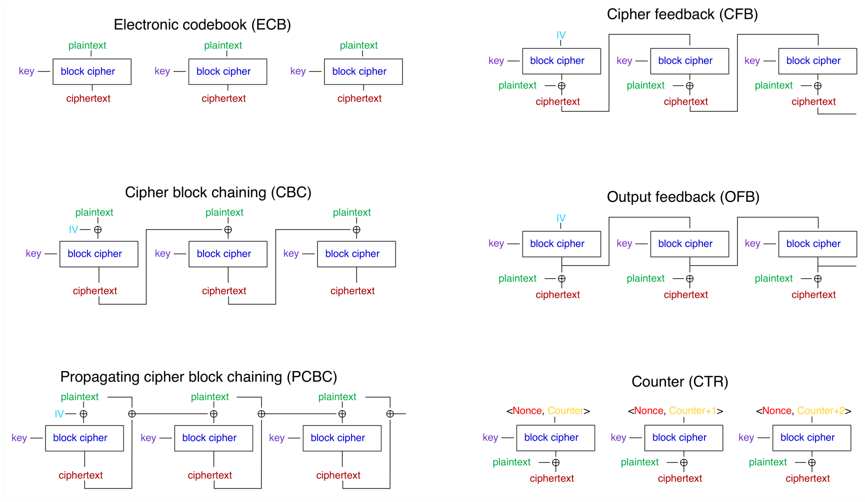


What we want



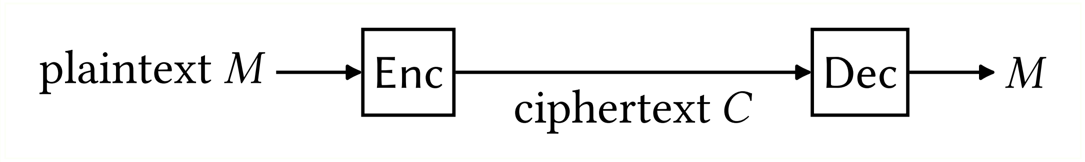
What we actually get

Block cipher modes of operation



Source: Wikipedia

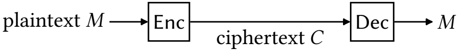
Thinking about secrecy



Source: The Joy of Cryptography

Thinking about secrecy

- Keep the whole design secret?
- **"Advantages"**:
  - Attacker doesn't know how our cipher (or system, more generally,) works.
- **Disadvantages**:
  - Figuring out how the thing works might mean a break.
  - Can't expose cipher to scrutiny.
  - Everyone needs to invent a cipher.

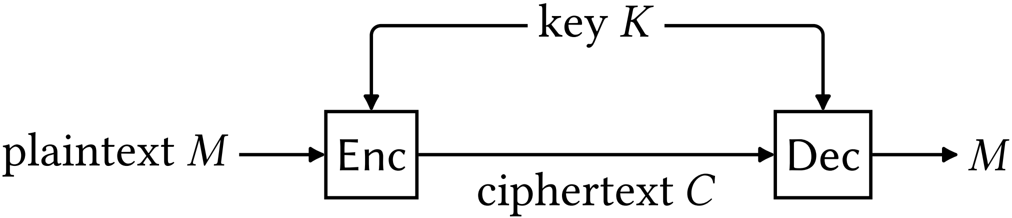


Source: The Joy of Cryptography

Kerckhoff's principle

- *"A cryptosystem should be secure even if everything about the system, except the key, is public knowledge."* — Auguste Kerckhoffs, 1883
- **Why it matters**:
  - No "security through obscurity"
  - The key is the only secret: the rest can be audited, tested, trusted
  - Encourages open standards and peer review
  - If your system's security depends on nobody knowing how it works, it's not secure.

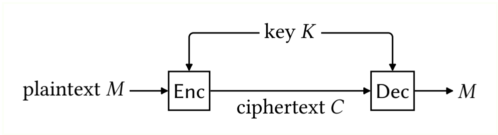
Thinking about secrecy



Concentrate all the need for secrecy in the key!

Thinking about secrecy

- Cipher can be scrutinized, used by anyone.
- Design can be shown to hold so long as the key is secret.
- This is how virtually all cryptography is designed today.



Source: The Joy of Cryptography

One-time pad

First look at a symmetric cipher

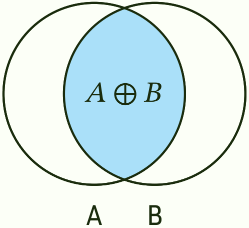
$$\underline{\textsc{Enc}(K, M):}$$
$$C := K \oplus M$$
$$\text{return } C$$

$$\underline{\textsc{Dec}(K, C):}$$
$$M := K \oplus C$$
$$\text{return } M$$

XOR (Exclusive OR) operation

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Table: Truth table for XOR operation



- XOR returns 1 when inputs differ
- XOR returns 0 when inputs are the same
- Key property: $x \oplus x = 0$ and $x \oplus 0 = x$
- Self-inverse: $(M \oplus K) \oplus K = M$

One-time pad

First look at a symmetric cipher

$$
\begin{array}{rll}
 & \texttt{11101111101111100011} & M \\
\oplus & \texttt{00011001110000111101} & K \\
\hline
= & \texttt{11110110011111011110} & C = \mathsf{Enc}(K, M)
\end{array}
$$

(We're encoding the message and key as bits)

One-time pad

First look at a symmetric cipher

$$
\begin{array}{lll}
 & \texttt{11110110011111011110} & C \\
\oplus & \texttt{00011001110000111101} & K \\
\hline
= & \texttt{11101111101111100011} & M = \mathrm{Dec}(K, C)
\end{array}
$$

(We're encoding the message and key as bits)

One-time pad

Correctness proof

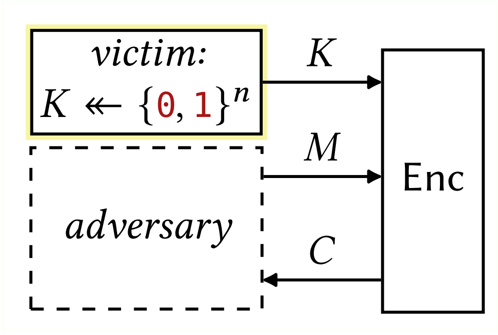
- $\forall (n > 0,\ K \in \{0,1\}^n,\ M \in \{0,1\}^n),\ \mathsf{Dec}(K, \mathsf{Enc}(K, M)) = M$
- For all positive n, any key of n bits and message of n bits will decrypt back to the same plaintext if encrypted into a ciphertext.
- **Proof**:

$$\begin{aligned}
\mathsf{Dec}(K, \mathsf{Enc}(K, M)) &= \mathsf{Dec}(K, K \oplus M) \\
&= K \oplus (K \oplus M) \\
&= (K \oplus K) \oplus M \\
&= 0^n \oplus M \\
&= M \qquad \square
\end{aligned}$$

One-time pad

How do we prove security?

- When we prove security, we prove what is or isn't possible by the attacker calling Attack(M).

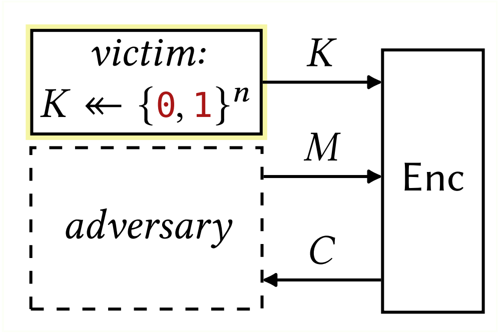


Source: The Joy of Cryptography

One-time pad

How do we prove security?

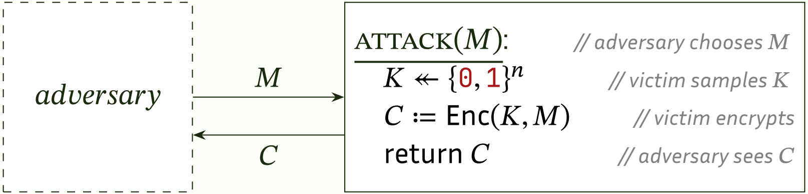
- "Victim" chooses their key.
  - Fresh key for each message (each key used only once)
  - This means output will differ even if same plaintext is input twice by adversary
- Adversary chooses the message and receives the ciphertext.
- We say that **the adversary has access to an encryption oracle**.



Source: The Joy of Cryptography

One-time pad

How do we prove security?



One-time pad

How do we prove security?

- **Generally**: a cipher is secure if the adversary can't distinguish the output of calls to $ATTACK$ from random junk.
- **Formally**: For all positive integers n and all choices of plaintext $M \in \{0, 1\}^n$, the output of the following subroutine is uniformly distributed:

$$\underline{\textsc{Attack}(M):}$$
$$K \twoheadleftarrow \{0, 1\}^n$$
$$C := K \oplus M$$
$$\text{return } C$$

One-time pad

How do we prove security?

- If the key is random, the output will be uniformly distributed!
- Suppose $M = 01$:
  - $K = 00$ is chosen with probability $1/4$: $C = K \oplus M = 00 \oplus 01 = 01$.
  - $K = 01$ is chosen with probability $1/4$: $C = K \oplus M = 01 \oplus 01 = 00$.
  - $K = 10$ is chosen with probability $1/4$: $C = K \oplus M = 10 \oplus 01 = 11$.
  - $K = 11$ is chosen with probability $1/4$: $C = K \oplus M = 11 \oplus 01 = 10$.

$\textsc{Attack}(M)$:

$K \twoheadleftarrow \{0,1\}^n$

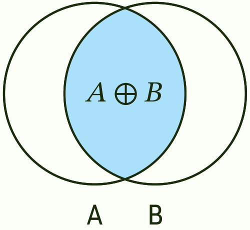
$C := K \oplus M$

return C

XOR (Exclusive OR) operation

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Table: Truth table for XOR operation



- XOR returns 1 when inputs differ
- XOR returns 0 when inputs are the same
- Key property: $x \oplus x = 0$ and $x \oplus 0 = x$
- Self-inverse: $(M \oplus K) \oplus K = M$

One-time pad

What's so special about XOR?

- Let's replace $\oplus$ with $\wedge$. What would happen?
- Output no longer uniform!

A	B	A ∧ B
0	0	0
0	1	0
1	0	0
1	1	1

Table: Truth table for AND operation

$\underline{\text{ATTACK}(M)}$:

$K \twoheadleftarrow \{0,1\}^n$

$C := K \wedge M$

return C

One-time pad

How do we prove security?

- What if this is true only for $M = \texttt{01}$?
- Fine, let's pick any $M, C \in \{\texttt{0}, \texttt{1}\}^n$.
- What is $\Pr[\texttt{Attack(M) = C}]$?
- Answer: Exactly when $C = \textsf{Enc}(K, M) = K \oplus M$.
- ...which occurs for exactly one K.
- Since K is chosen uniformly from $\{\texttt{0}, \texttt{1}\}^n$, the probability of choosing that K is $\frac{1}{2^n}$. □

$\textsc{Attack}(M)$:

$K \twoheadleftarrow \{\texttt{0}, \texttt{1}\}^n$

$C := K \oplus M$

return C

One-time pad

From the adversary's perspective...

$\underline{\textsc{Attack}(M)}$:

$K \twoheadleftarrow \{0, 1\}^n$

$C := K \oplus M$

return C

$\approxeq$

(indistinguishable from)

$\underline{\textsc{Junk}(M)}$:

$C \twoheadleftarrow \{0, 1\}^n$

return C

“Real or random?”

One-time pad

What about $\pmod{n}$?

- Let's replace $\oplus$ with $\pmod{n}$. What would happen?
- Still good!
- Can you prove correctness and security?

$\underline{\textsc{Attack}(M):}$

$K \twoheadleftarrow \mathbb{Z}_n$

$C := (K + M) \pmod{n}$

return C