What is this course about?

Aims
This course provides an overview of basic modern cryptographic techniques and covers essential concepts that users of cryptographic standards need to understand to achieve their intended security goals.

Objectives
By the end of the course you should

▶ be familiar with commonly used standardized cryptographic building blocks;
▶ be able to match application requirements with concrete security definitions and identify their absence in naive schemes;
▶ understand various adversarial capabilities and basic attack algorithms and how they affect key sizes;
▶ understand and compare the finite groups most commonly used with discrete-logarithm schemes;
▶ understand the basic number theory underlying the most common public-key schemes, and some efficient implementation techniques.

Related textbooks

Main reference:
▶ Jonathan Katz, Yehuda Lindell: Introduction to Modern Cryptography
  2nd ed., Chapman & Hall/CRC, 2014

Further reading:
▶ Christof Paar, Jan Pelzl: Understanding Cryptography
  Springer, 2010
  http://www.springerlink.com/content/978-3-642-04100-6/
  http://www.crypto-textbook.com/
▶ Douglas Stinson: Cryptography – Theory and Practice
  3rd ed., CRC Press, 2005
▶ Menezes, van Oorschot, Vanstone: Handbook of Applied Cryptography
  CRC Press, 1996
  http://www.cacr.math.uwaterloo.ca/hac/

The course notes and some of the exercises also contain URLs with more detailed information.
Common information security targets

Most information-security concerns fall into three broad categories:

- **Confidentiality** ensuring that information is accessible only to those authorised to have access
- **Integrity** safeguarding the accuracy and completeness of information and processing methods
- **Availability** ensuring that authorised users have access to information and associated assets when required

Basic threat scenarios:

- **Eavesdropper**: Alice ——— Bob
  (passive)
  Eve

- **Middle-person attack**: Alice ——— Mallory ——— Bob
  (active)

- **Storage security**: Alice ——— disk ——— Mallory

Message integrity schemes

Other cryptographic algorithm triples instead aim at authenticating the integrity and origin of a message:

**Message authentication code (MAC)**

- \( K \leftarrow \text{Gen} \) private-key generation
- \( T := \text{Mac}_K(M) \) message tag generation
- \( M' \neq M \implies \text{Mac}_K(M') \neq T \) MAC verification:
  recalculate and compare tag

**Digital signature**

- \( (PK, SK) \leftarrow \text{Gen} \) public/secret key-pair generation
- \( S \leftarrow \text{Sign}_{SK}(M) \) signature generation using secret key
- \( \text{Vrfy}_{PK}(M, S) = 1 \) signature verification using public key

Encryption schemes

Encryption schemes are algorithm triples (Gen, Enc, Dec) aimed at facilitating message confidentiality:

**Private-key (symmetric) encryption scheme**

- \( K \leftarrow \text{Gen} \) private-key generation
- \( C \leftarrow \text{Enc}_K(M) \) encryption of plain-text message \( M \)
- \( \text{Dec}_K(C) = M \) decryption of cipher-text message \( C \)

**Public-key (asymmetric) encryption scheme**

- \( (PK, SK) \leftarrow \text{Gen} \) public/secret key-pair generation
- \( C \leftarrow \text{Enc}_{PK}(M) \) encryption using public key
- \( \text{Dec}_{SK}(C) = M \) decryption using secret key

**Probabilistic algorithms**: Gen and (often also) Enc access a random-bit generator that can toss coins (uniformly distributed, independent).
Notation: \( \leftarrow \) assigns the output of a probabilistic algorithm, \( := \) that of a deterministic algorithm.

When is an encryption scheme “secure”?

If no adversary can . . .

- . . . find out the secret/private key?
- . . . find the plaintext message \( M \)?
- . . . determine any character/bit of \( M \)?
- . . . determine any information about \( M \) from \( C \)?
- . . . compute any function of the plaintext \( M \) from ciphertext \( C \)?
  \( \Rightarrow \) “semantic security”

And for integrity schemes, should we demand that no adversary can . . .

- . . . find out the secret/private key?
- . . . create a new message \( M' \) and matching tag/signature?
- . . . create a new \( M' \) that verifies with a given tag/signature?
- . . . modify or recombine a message+tag so they still verify?
- . . . create two messages with the same signature?
What capabilities may the adversary have?

- access to some ciphertext \( C \)
- access to some plaintext/ciphertext pairs \((M, C)\) with \( C \leftarrow \text{Enc}_K(M) \)
- ability to trick the user of \( \text{Enc}_K \) into encrypting some plaintext of the adversary’s choice and return the result? (“oracle access” to \( \text{Enc} \))
- ability to trick the user of \( \text{Dec}_K \) into decrypting some ciphertext of the adversary’s choice and return the result? (“oracle access” to \( \text{Dec} \))?
- ability to modify or replace \( C \) en route? (not limited to eavesdropping)
- how many applications of \( \text{Enc}_K \) or \( \text{Dec}_K \) can be observed?
- unlimited / polynomial / realistic (\( \ll 2^{80} \) steps) computation time?
- knowledge of all algorithms used

**Wanted:** Clear definitions of what security of an encryption scheme means, to guide both designers and users of schemes, and allow proofs.

### Kerckhoffs’ principles (1883)

**Requirements for a good traditional military encryption system:**

1. The system must be substantially, if not mathematically, undecipherable;
2. The system must not require secrecy and can be stolen by the enemy without causing trouble;
3. It must be easy to communicate and remember the keys without requiring written notes, it must also be easy to change or modify the keys with different participants;
4. The system ought to be compatible with telegraph communication;
5. The system must be portable, and its use must not require more than one person;
6. Finally, regarding the circumstances in which such system is applied, it must be easy to use and must neither require stress of mind nor the knowledge of a long series of rules.


### Kerckhoffs’ principle today

**Requirement for a modern encryption system:**

1. It was evaluated assuming that the enemy knows the system.
2. Its security relies entirely on the key being secret.

**Note:**

- The design and implementation of a secure communication system is a major investment and is not easily and quickly repeated.
- Relying on the enemy not knowing the encryption system is generally frowned upon as “security by obscurity”.
- The most trusted cryptographic algorithms have been published, standardized, and withstood years of cryptanalysis.
- A cryptographic key should be just a random choice that can be easily replaced, by rerunning a key-generation algorithm.
- Keys can and will be lost: cryptographic systems should provide support for easy rekeying, redistribution of keys, and quick revocation of compromised keys.

### A note about message length

We explicitly do not worry in the following about the adversary being able to infer something about the length \( m \) of the plaintext message \( M \) by looking at the length \( n \) of the ciphertext \( C \).

Therefore, we will consider here in security definitions for encryption schemes only messages of *fixed* length \( m \).

Variable-length messages could be extended to a fixed length, by padding, but this can be expensive. It will depend on the specific application whether the benefits of fixed-length padding outweigh the added transmission cost.

Nevertheless, in practice, ciphertext length must always be considered as a potential information leak. Examples:

- Encrypted-file lengths often permit unambiguous reconstruction of what pages a HTTPS user accessed on a public web site.
  G. Danezis: Traffic analysis of the HTTP protocol over TLS. http://www0.cs.ucl.ac.uk/staff/G.Danezis/papers/TLSanon.pdf
  Also: CVE-2012-4929/CRIME
- Data compression can be abused to extract information from an encrypted message if an adversary can control part of that message.
Demo: leaking plaintext through compressed data length

```
$ cat compression-leak
#!/bin/bash
PLAINTEXT=cafe
←
KEY="N-326bQf7j9emVr.69H1fX"
ENC="openssl enc -aes-128-ctr -pass pass:$KEY"
for t in {a,b,c,d,e,f}{a,b,c,d,e,f}{a,b,c,d,e,f}{a,b,c,d,e,f} ; do
echo -n "$t "
echo $t $PLAINTEXT | gzip -c | $ENC | wc -c
done | sort -nk2
$ ./compression-leak
```

Historic examples of simple ciphers

**Shift Cipher:** Treat letters \( \{A, \ldots, Z\} \) like integers \( \{0, \ldots, 25\} = \mathbb{Z}_{26} \). Choose key \( K \in \mathbb{Z}_{26} \), encrypt each letter individually by addition modulo 26, decrypt by subtraction modulo 26.

Example with \( K = 25 \equiv 1 \pmod{26} \): IBM \( \rightarrow \) HAL. 
\( K = -3 \) known as *Caesar Cipher*, \( K = 13 \) as rot13.

The tiny key-space size 26 makes brute force key search trivial.

**Transposition Cipher:** \( K \) is permutation of letter positions.

Key space is \( n! \), where \( n \) is the permutation block length.

**Substitution Cipher (monoalphabetic):** Key is permutation \( K: \mathbb{Z}_{26} \leftrightarrow \mathbb{Z}_{26} \). Encrypt plaintext \( M = m_1 m_2 \ldots m_n \) with \( c_i = K(m_i) \) to get ciphertext \( C = c_1 c_2 \ldots c_n \), decrypt with \( m_i = K^{-1}(c_i) \).

Key space size \( 26! \) \( > 4 \times 10^{26} \) makes brute force search infeasible.

Statistical properties of plain text

The most common letters in English:
The most common bigrams in English:
TH, HE, IN, ER, AN, RE, ED, ON, ES, ST, EN, AT, TO, . . .
The most common trigrams in English:
THE, ING, AND, HER, ERE, ENT, THA, NTH, WAS, ETH, . . .

English text is highly redundant: very roughly 1 bit/letter entropy.

Monalphabetic substitution ciphers allow simple ciphertext-only attacks based on
digram or trigram statistics (for messages of at least few hundred characters).
Vigenère cipher

Inputs:
- Key word $K = k_1 k_2 \ldots k_l$
- Plain text $M = m_1 m_2 \ldots m_n$

Encrypt into ciphertext:
$$c_i = (m_i + k_{(i-1) \mod l+1}) \mod 26$$

Example: $K = \text{SECRET}$

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>C</th>
<th>R</th>
<th>E</th>
<th>T</th>
<th>S</th>
<th>E</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>K</td>
<td>A</td>
<td>T</td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

The modular addition can be replaced with XOR:
$$c_i = m_i \oplus k_{(i-1) \mod l+1} \quad m_i, k_i, c_i \in \{0, 1\}$$

Vigenère is an example of a *polyalphabetic* cipher.

Perfect secrecy

**Computational security**

The most efficient known algorithm for breaking a cipher would require far more computational steps than all hardware available to any adversary can perform.

**Unconditional security**

Adversaries have not enough information to decide (from the ciphertext) whether one plaintext is more likely to be correct than another, even with unlimited computational power at their disposal.

Perfect secrecy II

Consider a private-key encryption scheme
$$\text{Enc} : K \times M \to C, \quad \text{Dec} : K \times C \to M$$
with $\text{Dec}_K(\text{Enc}_K(M)) = M$ for all $K \in K, M \in M$, where $M, C, K$ are the sets of possible plaintexts, ciphertexts and keys, respectively.

Let also $M \in M, C \in C$ and $K \in K$ be values of plaintext, ciphertext and key. Let $P(M)$ and $P(K)$ denote an adversary's respective a-priori knowledge of the probability that plaintext $M$ or key $K$ are used.

The adversary can then calculate the probability of any ciphertext $C$ as
$$P(C) = \sum_{K \in K} P(K) \cdot P(\text{Dec}_K(C)),$$
and can also determine the conditional probability
$$P(C|M) = \sum_{\{K \in K | M = \text{Dec}_K(C)\}} P(K).$$
Perfect secrecy III

Having eavesdropped some ciphertext $C$, an adversary can then use Bayes’ theorem to calculate for any plaintext $M \in \mathcal{M}$

$$P(M|C) = \frac{P(M) \cdot P(C|M)}{P(C)} = \frac{P(M) \cdot \sum_{K \in \mathcal{K}} P(K = \text{Dec}_K(C))}{\sum_K P(K) \cdot P(\text{Dec}_K(C))}. $$

Perfect secrecy

An encryption scheme over a message space $\mathcal{M}$ is perfectly secret if for every probability distribution over $\mathcal{M}$, every message $M \in \mathcal{M}$, and every ciphertext $C \in \mathcal{C}$ with $P(C) > 0$ we have

$$P(M|C) = P(M).$$

In other words: looking at the ciphertext $C$ leads to no new information beyond what was already known about $M$ in advance ⇒ eavesdropping $C$ has no benefit, even with unlimited computational power.


Vernam cipher / one-time pad I

Shannon’s theorem:

Let $(\text{Gen}, \text{Enc}, \text{Dec})$ be an encryption scheme over a message space $\mathcal{M}$ with $|\mathcal{M}| = |\mathcal{K}| = |\mathcal{C}|$. It is perfectly secret if and only if

1. Gen chooses every $K$ with equal probability $1/|\mathcal{K}|$;
2. for every $M \in \mathcal{M}$ and every $C \in \mathcal{C}$, there exists a unique key $K \in \mathcal{K}$ such that $C = \text{Enc}_K(M)$.

The standard example of a perfectly-secure symmetric encryption scheme:

**One-time pad**

$\mathcal{K} = \mathcal{C} = \mathcal{M} = \{0,1\}^n$

- $\text{Gen} : K \in_R \{0,1\}^m$ ($m$ uniform, independent coin tosses)
- $\text{Enc}_K(M) = K \oplus M$ ($\oplus =$ bit-wise XOR)
- $\text{Dec}_K(C) = K \oplus C$

Example:

$0xbd4b083f6aae \oplus "\text{Vernam}" = 0xbd4b083f6aae \oplus 0x5665726e616d = 0xeb2e7a510bc3$

Vernam cipher / one-time pad II

The one-time pad is a variant of the Vigenère Cipher with $l = n$: the key is as long as the plaintext. No key bit is ever used to encrypt more than one plaintext bit.

**Note:** If $x$ is a random bit with any probability distribution and $y$ is one with uniform probability distribution ($P(y = 0) = P(y = 1) = \frac{1}{2}$), then the exclusive-or result $x \oplus y$ will have uniform probability distribution. This also works for addition modulo $m$ (or for any finite group).

For each possible plaintext $M$, there exists a key $K = M \oplus C$ that turns a given ciphertext $C$ into $M = \text{Dec}_K(C)$. If all $K$ are equally likely, then also all $M$ will be equally likely for a given $C$, which fulfills Shannon’s definition of perfect secrecy.

What happens if you use a one-time pad twice?

One-time pads have been used intensively during significant parts of the 20th century for diplomatic communications security, e.g. on the telex line between Moscow and Washington. Keys were generated by hardware random bit stream generators and distributed via trusted couriers.

In the 1940s, the Soviet Union encrypted part of its diplomatic communication using recycled one-time pads, leading to the success of the US decryption project VENONA.

http://www.nsa.gov/public_info/declass/venona/
Making the one-time pad more efficient

The one-time pad is very simple, but also very inconvenient: one key bit for each message bit!

Many standard libraries contain pseudo-random number generators (PRNGs). They are used in simulations, games, probabilistic algorithms, testing, etc.

They expand a “seed value” \( R_0 \) into a sequence of numbers \( R_1, R_2, \ldots \) that look very random:

\[
R_i = f(R_{i-1}, i)
\]

The results pass numerous statistical tests for randomness (e.g. Marsaglia’s “Diehard” tests).

Can we not use \( R_0 \) as a short key, split our message \( M \) into chunks \( M_1, M_2, \ldots \) and XOR with (some function \( g \) of) \( R_i \) to encrypt \( M_i \)?

\[
C_i = M_i \oplus g(R_i, i)
\]

But what are secure choices for \( f \) and \( g \)?

What security property do we expect from such a generator, and what security can we expect from the resulting encryption scheme?

Private-key (symmetric) encryption

A **private-key encryption scheme** is a tuple of probabilistic polynomial-time algorithms \((Gen, Enc, Dec)\) and sets \(K, M, C\) such that

- the **key generation algorithm** \(Gen\) receives a security parameter \(\ell\) and outputs a key \(K \leftarrow Gen(\ell)\), with \(K \in K\), key length \(|K| \geq \ell\);

- the **encryption algorithm** \(Enc\) maps a key \(K\) and a plaintext message \(M \in M = \{0, 1\}^m\) to a ciphertext message \(C \leftarrow Enc_K(M)\);

- the **decryption algorithm** \(Dec\) maps a key \(K\) and a ciphertext message \(C \in C = \{0, 1\}^n\) (\(n \geq m\)) to a plaintext message \(M := Dec_K(C)\);

- for all \(\ell, K \leftarrow Gen(\ell)\), and \(M \in \{0, 1\}^m\): \(Dec_K(Enc_K(M)) = M\).

**Notes:**

A “polynomial-time algorithm” has constants \(a, b, c\) such that the runtime is always less than \(a \cdot \ell^b + c\) if the input is \(\ell\) bits long. (think Turing machine)

Technicality: we supply the security parameter \(\ell\) to \(Gen\) here in unary encoding (as a sequence of \(\ell\) “1” bits: \(1^\ell\)), merely to remain compatible with the notion of “input size” from computational complexity theory. In practice, \(Gen\) usually simply picks \(\ell\) random bits \(K \in R\{0, 1\}^\ell\).

A non-secure pseudo-random number generator

**Example (insecure)**

Linear congruential generator with secret parameters \((a, b, R_0)\):

\[
R_{i+1} = (aR_i + b) \mod m
\]

**Attack:** guess some plain text (e.g., known file header), obtain for example \((R_1, R_2, R_3)\), then solve system of linear equations over \(\mathbb{Z}_m\):

\[
\begin{align*}
R_2 & \equiv aR_1 + b \pmod{m} \\
R_3 & \equiv aR_2 + b \pmod{m}
\end{align*}
\]

**Solution:**

\[
\begin{align*}
a & \equiv (R_2 - R_3)/((R_1 - R_2) \pmod{m}) \\
b & \equiv R_2 - R_1(R_2 - R_3)/(R_1 - R_2) \pmod{m}
\end{align*}
\]

Multiple solutions if \(gcd(R_1 - R_2, m) \neq 1\): resolved using \(R_4\) or just by trying all possible values.

Security definitions for encryption schemes

We define security via the rules of a game played between two players:

- a challenger, who uses an encryption scheme \(\Pi = (Gen, Enc, Dec)\)
- an adversary \(A\), who tries to demonstrate a weakness in \(\Pi\).

Most of these games follow a simple pattern:

1. the challenger uniformly picks at random a secret bit \(b \in R\{0, 1\}\)
2. \(A\) interacts with the challenger according to the rules of the game
3. At the end, \(A\) has to output a bit \(b'\).

The outcome of such a game \(X_{A,\Pi}(\ell)\) is either

- \(b = b'\) ⇒ \(A\) won the game, we write \(X_{A,\Pi}(\ell) = 1\)
- \(b \neq b'\) ⇒ \(A\) lost the game, we write \(X_{A,\Pi}(\ell) = 0\)

**Advantage**

One way to quantify \(A\)'s ability to guess \(b\) is

\[
Adv_{X_{A,\Pi}(\ell)} = |\mathbb{P}(b = 1 \text{ and } b' = 1) - \mathbb{P}(b = 0 \text{ and } b' = 1)|
\]
Negligible advantage

Security definition

An encryption scheme \( \Pi \) is considered “\( X \) secure” if for all probabilistic polynomial-time (PPT) adversaries \( A \) there exists a “negligible” function \( \text{negl} \) such that

\[
P(\text{\text{X} } A, \Pi(\ell)) = 1 < \frac{1}{2} + \text{negl}(\ell).
\]

Some authors prefer the equivalent definition with

\[
\text{Adv}_{\text{\text{X} } A, \Pi}(\ell) < \text{negl}(\ell).
\]

Negligible functions

A function \( \text{negl}(\ell): \mathbb{N} \rightarrow \mathbb{R} \) is “negligible” if, as \( \ell \to \infty \), it converges faster to zero than \( \frac{1}{\text{poly}(\ell)} \) does for any polynomial \( \text{poly}(\ell) \).

**In practice:** We want \( \text{negl}(\ell) \) to drop below a small number (e.g., \( 2^{-80} \) or \( 2^{-100} \)) for modest key lengths \( \ell \) (e.g., \( \log_{10} \ell \approx 2 \ldots \)). Then no realistic opponent will have the computational power to repeat the game often enough to win at least once more than what is expected from random guessing.

“Computationally infeasible”

With good cryptographic primitives, the only form of possible cryptanalysis should be an exhaustive search of all possible keys (brute force attack).

The following numbers give a rough idea of the limits involved:

Let’s assume we can later this century produce VLSI chips with 10 GHz clock frequency and each of these chips costs 10 $ and can test in a single clock cycle 100 keys. For 10 million $, we could then buy the chips needed to build a machine that can test \( 10^{18} \approx 2^{60} \) keys per second.

Such a hypothetical machine could break an 80-bit key in 7 days on average. For a 128-bit key it would need over \( 10^{12} \) years, that is over \( 100 \times \) the age of the universe.

**Rough limit of computational feasibility:** \( 2^{80} \) iterations (i.e., \( < 2^{60} \) feasible with effort, but \( > 2^{100} \) certainly not)

For comparison:

- The fastest key search effort using thousands of Internet PCs (RC5-64, 2002) achieved in the order of \( 2^{37} \) keys per second.
- Since January 2018, the Bitcoin network has been searching through about \( 10^{19} \approx 2^{63} \) cryptographic hash values per second, mostly using ASICs.

Indistinguishability in the presence of an eavesdropper

Private-key encryption scheme \( \Pi = (\text{Gen}, \text{Enc}, \text{Dec}) \), \( \mathcal{M} = \{0, 1\}^m \), security parameter \( \ell \).

Experiment/game PrivK_{\text{eav}}(\ell):

\[
\begin{align*}
1^\ell & \xrightarrow{} b \in \{0, 1\} \quad K \leftarrow \text{Gen}(1^\ell) \\
\text{challenger} & \quad C \leftarrow \text{Enc}_K(M_b) \\
A & \quad M_0, M_1 \\
A & \quad C \\
A & \quad b' \\
\end{align*}
\]

**Setup:**

1. The challenger generates a bit \( b \in \{0, 1\} \) and a key \( K \leftarrow \text{Gen}(1^\ell) \).
2. The adversary \( A \) is given input \( 1^\ell \)

**Rules for the interaction:**

1. The adversary \( A \) outputs a pair of messages: \( M_0, M_1 \in \{0, 1\}^m \).
2. The challenger computes \( C \leftarrow \text{Enc}_K(M_b) \) and returns \( C \) to \( A \)

Finally, \( A \) outputs \( b' \). If \( b' = b \) then \( A \) has succeeded \( \Rightarrow \text{PrivK}_{\text{eav}}(\ell) = 1 \)

**Definition:** A private-key encryption scheme \( \Pi \) has indistinguishable encryption in the presence of an eavesdropper if for all probabilistic, polynomial-time adversaries \( A \) there exists a negligible function \( \text{negl} \) such that

\[
P(\text{PrivK}_{\text{eav}}(\ell) = 1) \leq \frac{1}{2} + \text{negl}(\ell)
\]

**In other words:** as we increase the security parameter \( \ell \), we quickly reach the point where no eavesdropper can do significantly better than just randomly guessing \( b \).
Pseudo-random generator I

\[ G : \{0,1\}^n \to \{0,1\}^{e(n)} \] where \( e(\cdot) \) is a polynomial (expansion factor)

**Definition**

\( G \) is a pseudo-random generator if both

1. \( e(n) > n \) for all \( n \) (expansion)
2. for all probabilistic, polynomial-time distinguishers \( D \) there exists a negligible function \( \text{negl} \) such that
   \[ |\Pr[D(r) = 1] - \Pr[D(G(s)) = 1]| \leq \text{negl}(n) \]

where both \( r \in \{0,1\}^{e(n)} \) and the seed \( s \in \{0,1\}^n \) are chosen at random, and the probabilities are taken over all coin tosses used by \( D \) and for picking \( r \) and \( s \).

Pseudo-random generator II

A brute-force distinguisher \( D \) would enumerate all \( 2^n \) possible outputs of \( G \), and return 1 if the input is one of them.

It would achieve
\[ \Pr[D(G(s)) = 1] = \frac{1}{2^n} \]
\[ \Pr[D(r) = 1] = \frac{2}{2^n} \]
the difference of which converges to 1, which is not negligible.

But a brute-force distinguisher has a exponential run-time \( \Omega(2^n) \), and is therefore excluded!

We do not know how to prove that a given algorithm is a pseudo-random generator, but there are many algorithms that are widely believed to be.

Some constructions are pseudo-random generators if another well-studied problem is not solvable in polynomial time.

Encrypting using a pseudo-random generator

We define the following fixed-length private-key encryption scheme:

\[ \Pi_{\text{PRG}} = (\text{Gen}, \text{Enc}, \text{Dec}) : \]

Let \( G \) be a pseudo-random generator with expansion factor \( e(\cdot) \),
\( K = \{0,1\}^\ell, M = C = \{0,1\}^{e(\ell)} \)

- **Gen**: on input \( 1^\ell \) chose \( K \in R \{0,1\}^\ell \) randomly
- **Enc**: \( C := G(K) \oplus M \)
- **Dec**: \( M := G(K) \oplus C \)

Such constructions are known as "stream ciphers".

We can prove that \( \Pi_{\text{PRG}} \) has "indistinguishable encryption in the presence of an eavesdropper" assuming that \( G \) is a pseudo-random generator:

Claim: If \( \Pi_{\text{PRG}} \) did not have indistinguishability in the presence of an eavesdropper, there would be an adversary \( A \) for which
\[ e(\ell) := \Pr[\text{PrivK}^{\text{eav}}_{A,\Pi_{\text{PRG}}} = 1] - \frac{1}{2} \]
is not negligible.

Use that \( A \) to construct a distinguisher \( D \) for \( G \):

- receive input \( W \in \{0,1\}^{e(\ell)} \)
- pick \( b \in R \{0,1\} \)
- run \( A(1^\ell) \) and receive from it \( M_0, M_1 \in \{0,1\}^{e(\ell)} \)
- return \( C := W \oplus M_b \) to \( A \)
- receive \( b' \) from \( A \)
- return 1 if \( b' = b \), otherwise return 0

Now, what is \( |\Pr[D(r) = 1] - \Pr[D(G(K)) = 1]| \)?

Security proof for a stream cipher
Security proofs through reduction

Some key points about this style of “security proof”:

- We have not shown that the encryption scheme \( \Pi_{\text{PRG}} \) is “secure”. (We don’t know how to do this!)
- We have shown that \( \Pi_{\text{PRG}} \) has one particular type of security property, if one of its building blocks \( G \) has another one.
- We have “reduced” the security of construct \( \Pi_{\text{PRG}} \) to another problem \( X \):

\[
\begin{array}{c}
\text{Reduction} \\
\text{instance of } \Pi \\
\text{to } X
\end{array}
\]

Here: \( X = \text{distinguishing output of } G \) from random string

- We have shown how to turn any successful attack on \( \Pi_{\text{PRG}} \) into an equally successful attack on its underlying building block \( G \).
- “Successful attack” means finding a polynomial-time probabilistic adversary algorithm that succeeds with non-negligible success probability in winning the game specified by the security definition.

Security proofs through reduction

In the end, the provable security of some cryptographic construct (e.g., \( \Pi_{\text{PRG}} \), some mode of operation, some security protocol) boils down to these questions:

- What do we expect from the construct?
- What do we expect from the underlying building blocks?
- Does the construct introduce new weaknesses?
- Does the construct mitigate potential existing weaknesses in its underlying building blocks?

Security for multiple encryptions

Private-key encryption scheme \( \Pi = (\text{Gen}, \text{Enc}, \text{Dec}), M = \{0,1\}^m \), security parameter \( \ell \).

Experiment/game \( \text{PrivK}^{\text{mult}}_{A,\Pi}(\ell) \):

\[
\begin{array}{c}
b \leftarrow \{0,1\} \\
K \leftarrow \text{Gen}(1^\ell) \\
C \leftarrow \text{Enc}_K(M_b) \\
\text{adversary}
\end{array}
\]

Setup:

1. The challenger generates a bit \( b \in \{0,1\} \) and a key \( K \leftarrow \text{Gen}(1^\ell) \).
2. The adversary \( A \) is given input \( 1^\ell \)

Rules for the interaction:

1. The adversary \( A \) outputs two sequences of \( t \) messages: \( M_0^1, M_0^2, \ldots, M_0^t \) and \( M_1^1, M_1^2, \ldots, M_1^t \), where all \( M_i^j \in \{0,1\}^m \).
2. The challenger computes \( C^i \leftarrow \text{Enc}_K(M_b^i) \) and returns \( C^1, C^2, \ldots, C^t \) to \( A \)

Finally, \( A \) outputs \( b' \). If \( b' = b \) then \( A \) has succeeded \( \Rightarrow \text{PrivK}^{\text{mult}}_{A,\Pi}(\ell) = 1 \)
Security for multiple encryptions (cont’d)

**Definition:** A private-key encryption scheme Π has indistinguishable multiple encryptions in the presence of an eavesdropper if for all probabilistic, polynomial-time adversaries A there exists a negligible function negl, such that

$$\Pr[\text{PrivK}_{A,\Pi}^{\text{m}}(\ell) = 1] \leq 1/2 + \text{negl}(\ell)$$

Same definition as for indistinguishable encryptions in the presence of an eavesdropper, except for referring to the multi-message eavesdropping experiment PrivK_{A,\Pi}^{\text{m}}(\ell).

**Example:** Does our stream cipher ΠPRG offer indistinguishable multiple encryptions in the presence of an eavesdropper?

Adversary A outputs four messages, and returns $$b' = 1$$ iff $$\Pr[\text{PrivK}_{A,\Pi}^{\text{m}}(\ell) = 1] = \text{negl}(\ell)$$.

Actually: Any encryption scheme is going to fail here!

Securing a stream cipher for multiple encryptions II

**Unsynchronized mode**

Some PRGs have two separate inputs, a key K and an “initial vector” IV. The private key K remains constant, while IV is freshly chosen at random for each message, and sent along with the message.

For each i: $$IV_i \in \{0,1\}^n, \quad C_i := (IV_i, G(K, IV_i) \oplus M_i)$$

But: what exact security properties do we expect of a G with IV input?

This question leads us to a new security primitive and associated security definition: pseudo-random functions and CPA security.

Securing a stream cipher for multiple encryptions I

How can we still use a stream cipher if we want to encrypt multiple messages $$M_1, M_2, \ldots, M_t$$ using a pseudo-random generator G?

**Synchronized mode**

Let the PRG run for longer to produce enough output bits for all messages:

$$G(K) = R_1 \parallel R_2 \parallel \ldots \parallel R_t, \quad C_i = R_i \oplus M_i$$

∥ is concatenation of bit strings

- convenient if $$M_1, M_2, \ldots, M_t$$ all belong to the same communications session and G is of a type that can produce long enough output
- requires preservation of internal state of G across sessions

**Security against chosen-plaintext attacks (CPA)**

Private-key encryption scheme Π = (Gen, Enc, Dec), $$M = \{0,1\}^m$$, security parameter $$\ell$$.

Experiment/game PrivK_{A,\Pi}^{\text{cpa}}(\ell):

1. The challenger generates a bit $$b \in \{0,1\}$$ and a key $$K \leftarrow \text{Gen}(1^\ell)$$.  
2. The adversary A is given input $$1^\ell$$.
3. Rules for the interaction:
   1. The adversary A is given oracle access to EncK:
      - A outputs $$M^1$$, gets EncK($$M^1$$), outputs $$M^2$$, gets EncK($$M^2$$), . . . 
   2. The adversary A outputs a pair of messages: $$M_0, M_1 \in \{0,1\}^m$$.
   3. The challenger computes $$C \leftarrow \text{Enc}_K(M_i)$$ and returns C to A.
   4. The adversary A continues to have oracle access to EncK.

Finally, A outputs $$b'$$. If $$b' = b$$ then A has succeeded $$\Rightarrow$$ PrivK_{A,\Pi}^{\text{cpa}}(\ell) = 1$$.
Security against chosen-plaintext attacks (cont’d)

Definition: A private-key encryption scheme \( \Pi \) has \textit{indistinguishable multiple encryptions under a chosen-plaintext attack} ("is CPA-secure") if for all probabilistic, polynomial-time adversaries \( A \) there exists a negligible function \( \text{negl} \), such that

\[
\Pr[\text{PrivK}_{\Pi, A}(\ell) = 1] \leq \frac{1}{2} + \text{negl}(\ell)
\]

Advantages:

- Eavesdroppers can often observe their own text being encrypted, even where the encrypter never intended to provide an oracle. (WW2 story: Midway Island/AF, server communication).
- CPA security provably implies security for multiple encryptions.
- CPA security allows us to build a variable-length encryption scheme simply by using a fixed-length one many times.

Random functions and permutations

\textbf{Random function}

Consider all possible functions of the form

\[
f : \{0, 1\}^m \rightarrow \{0, 1\}^n
\]

How often do you have to toss a coin to fill the value table of such a function \( f \) with random bits?

How many different such \( f \) are there?

An \( m \)-bit to \( n \)-bit \textit{random function} \( f \) is one that we have picked uniformly at random from all these possible functions.

\textbf{Random permutation}

Consider all possible permutations of the form

\[
g : \{0, 1\}^n \mapsto \{0, 1\}^n
\]

How many different such \( g \) are there?

An \( n \)-bit to \( n \)-bit \textit{random permutation} \( g \) is one that we have picked uniformly at random from all these possible permutations.

Pseudo-random functions and permutations

\textbf{Basic idea:}

A \textit{pseudo-random function (PRF)} is a fixed, efficiently computable function

\[
F : \{0, 1\}^k \times \{0, 1\}^m \rightarrow \{0, 1\}^n
\]

that (compared to a random function) depends on an additional input parameter \( K \in \{0, 1\}^k \), the key. Each choice of \( K \) leads to a function

\[
F_K : \{0, 1\}^m \rightarrow \{0, 1\}^n
\]

For typical key lengths (e.g., \( k, m \geq 128 \)), the set of all possible functions \( F_K \) will be a tiny subset of the set of all possible random functions \( f \).

For a secure pseudo-random function \( F \) there must be no practical way to distinguish between \( F_K \) and a corresponding random function \( f \) for anyone who does not know key \( K \).

We can similarly define a keyed \textit{pseudo-random permutation}.

In some proofs, in the interest of simplicity, we will only consider PRFs with \( k = m = n \).

Pseudo-random function (formal definition)

\[
F : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n \quad \text{efficient, keyed, length preserving}
\]

\textbf{Definition}

\( F \) is a \textit{pseudo-random function} if for all probabilistic, polynomial-time distinguishers \( D \) there exists a negligible function \( \text{negl} \) such that

\[
\left| \Pr[D(F_K(1^n) = 1) = 1] - \Pr[D(f(1^n) = 1) = 1] \right| \leq \text{negl}(n)
\]

where \( K \in \mathbb{R} \{0, 1\}^n \) is chosen uniformly at random and \( f \) is chosen uniformly at random from the set of functions mapping \( n \)-bit strings to \( n \)-bitstrings.

\textbf{Notation:} \( D^{f(\cdot)} \) means that algorithm \( D \) has "oracle access" to function \( f \).

\textbf{How does this differ from a pseudo-random generator?}

The distinguisher of a pseudo-random generator examines a string. Here, the distinguisher examines entire functions \( F_K \) and \( f \).

Any description of \( f \) would be at least \( n \cdot 2^n \) bits long and thus cannot be read in polynomial time. Therefore we can only provide oracle access to the distinguisher (i.e., allow \( D \) to query \( f \) a polynomial number of times).
CPA-secure encryption using a pseudo-random function

We define the following fixed-length private-key encryption scheme:

\[ \Pi_{PRF} = (\text{Gen}, \text{Enc}, \text{Dec}) : \]

Let \( F \) be a pseudo-random function.

- **Gen**: on input 1\(^k\) choose \( K \in \{0,1\}^k \) randomly
- **Enc**: read \( K \in \{0,1\}^k \) and \( M \in \{0,1\}^{|m|} \), choose \( R \in \{0,1\}^{|r|} \) randomly, then output \( C := (R, F_K(R) \oplus M) \)
- **Dec**: read \( K \in \{0,1\}^k \), \( C = (R, S) \in \{0,1\}^{2|c|} \), then output \( M := F_K(R) \oplus S \)

**Strategy for proving \( \Pi_{PRF} \) to be CPA secure:**

1. Show that a variant scheme \( \tilde{\Pi} \) in which we replace \( F_K \) with a random function \( f \) is CPA secure (just not efficient).
2. Show that replacing \( f \) with a pseudo-random function \( F_K \) cannot make it insecure, by showing how an attacker on the scheme using \( F_K \) can be converted into a distinguisher between \( f \) and \( F_K \), violating the assumption that \( F_K \) is a pseudo-random function.

**Security proof for encryption scheme \( \Pi_{PRF} \) (cont’d)**

Assume we have an attacker \( A \) against \( \Pi_{PRF} \) with non-negligible

\[ \epsilon(t) = \mathbb{P}(\text{PrivK}_{A, \Pi_{PRF}}(t) = 1) - \frac{1}{2} \]

Its performance against \( \tilde{\Pi} \) is also limited by

\[ \mathbb{P}(\text{PrivK}_{A, \tilde{\Pi}}(t) = 1) \leq \frac{1}{2} + \frac{q(t)}{2^t} \]

Combining those two equations we get

\[ \mathbb{P}(\text{PrivK}_{A, \Pi_{PRF}}(t) = 1) - \mathbb{P}(\text{PrivK}_{A, \tilde{\Pi}}(t) = 1) \geq \epsilon(t) - \frac{q(t)}{2^t} \]

which is not negligible either, allowing us to distinguish \( f \) from \( F_K \): Build distinguisher \( D^O \) using oracle \( O \) to play \( \text{PrivK}_{A, \Pi_{PRF}}(t) \) with \( A \):

1. Run \( A(1^t) \) and for each of its oracle queries \( M^i \) pick \( R^i \in \{0,1\}^{|r|} \), then return \( C^i := (R^i, O(R^i) \oplus M^i) \) to \( A \).
2. When \( A \) outputs \( M_0, M_1 \), pick \( b \in \{0,1\} \) and \( R_C \in \{0,1\}^{|r|} \), then return \( C := (R_C, O(R_C) \oplus M_b) \) to \( A \).
3. Continue answering \( A \)'s encryption oracle queries. When \( A \) outputs \( b' \), output 1 if \( b' = b \), otherwise 0.

**Security proof for encryption scheme \( \Pi_{PRF} \) (cont’d)**

How effective is this \( D^O \)?

1. **If \( D^O \)'s oracle is \( F_K \):** \( A \) effectively plays \( \text{PrivK}_{A, \Pi_{PRF}}(t) \) because if \( K \) is chosen randomly, \( D^O \) behaves towards \( A \) just like \( \Pi_{PRF} \), and therefore

\[ \mathbb{P}(D^O(1^t) = 1) = \mathbb{P}(\text{PrivK}_{A, \Pi_{PRF}}(t) = 1) \]

2. **If \( D^O \)'s oracle is \( f \):** likewise, \( A \) effectively plays \( \text{PrivK}_{A, \tilde{\Pi}}(t) \) and therefore

\[ \mathbb{P}(D^O(1^t) = 1) = \mathbb{P}(\text{PrivK}_{A, \tilde{\Pi}}(t) = 1) \]

if \( f \in_R \{0,1\}^{(0,1)^t} \) is chosen uniformly at random.

All combined the difference

\[ \mathbb{P}(D^O(1^t) = 1) - \mathbb{P}(D^O(1^t) = 1) \geq \epsilon(t) - \frac{q(t)}{2^t} \]

not being negligible implies that \( F_K \) is not a pseudo-random function, which contradicts the assumption, so \( \Pi_{PRF} \) is CPA secure.  

---

Katz/Lindell, pp 90–93
Pseudo-random permutation

$F : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ efficient, keyed, length preserving

$F_K$ is a pseudo-random permutation if

- for every key $K$, there is a 1-to-1 relationship for input and output
- $F_K$ and $F_K^{-1}$ can be calculated with polynomial-time algorithms
- there is no polynomial-time distinguisher that can distinguish $F_K$ (with randomly picked $K$) from a random permutation.

Note: Any pseudo-random permutation is also a pseudo-random function. A random function $f$ looks to any distinguisher just like a random permutation until it finds a collision $x \neq y$ with $f(x) = f(y)$. The probability for finding one in polynomial time is negligible (“birthday problem”).

A strong pseudo-random permutation remains indistinguishable even if the distinguisher has oracle access to the inverse.

Definition: $F$ is a strong pseudo-random permutation if for all polynomial-time distinguishers $D$ there exists a negligible function $\text{negl}$ such that

$$\left| \mathbb{P}(DF_K(.),F_K^{-1}(1^n) = 1) - \mathbb{P}(DF(.),F^{-1}(1^n) = 1) \right| \leq \text{negl}(n)$$

where $K \in \mathbb{R} \{0,1\}^n$ is chosen uniformly at random, and $f$ is chosen uniformly at random from the set of permutations on $n$-bit strings.

Iterating a random function

$f : \{1,\ldots,n\} \rightarrow \{1,\ldots,n\} \quad n^n$ such functions, pick one at random

Functional graph: vertices $\{1,\ldots,n\}$, directed edges $(i, f(i))$

Several expected values for $n \rightarrow \infty$, random $u \in \mathbb{R} \{1,\ldots,n\}$:

- tail length $E(t(u)) = \sqrt{\pi n/8}$
- cycle length $E(c(u)) = \sqrt{\pi n/8}$
- rho-length $E(t(u) + c(u)) = \sqrt{\pi n/2}$
- predecessors $E(|\{v| f^i(v) = u \land i > 0\}|) = \sqrt{\pi n/8}$
- edges of component containing $u$: $2n/3$

If $f$ is a random permutation: no trees, expected cycle length $(n + 1)/2$

Flajolet/Odlyzko: Random mapping statistics, EUROCRYPT’89, LNCS 434.

Probability of collision / Birthday problem

With 23 random people in a room, there is a 0.507 chance that two share a birthday. Surprised?

We throw $b$ balls into $n$ bins, selecting each bin uniformly at random.

With what probability do at least two balls end up in the same bin?

Expected number of balls thrown before first collision: $\sqrt{\pi n}$ (for $n \rightarrow \infty$)

Approximation formulas: http://cseweb.ucsd.edu/~mihir/cse207/v-birthday.pdf

- Historic ciphers
- Perfect secrecy
- Semantic security
- Block ciphers
- Modes of operation
- Message authenticity
- Authenticated encryption
- Secure hash functions
- Secure hash applications
- Key distribution problem
- Number theory and group theory
- Discrete logarithm problem
- RSA trapdoor permutation
- Digital signatures
Block ciphers

Practical, efficient algorithms that try to implement a pseudo-random permutation $E$ (and its inverse $D$) are called "block ciphers":

$E : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$

$D : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$

with $D_K(E_K(M)) = M$ for all $K \in \{0, 1\}^k$, $M \in \{0, 1\}^n$.

Alphabet size: $2^n$, size of key space: $2^k$.

Examples: AES, Camellia: $k, n = 128$ bit; DES, PRESENT: $n = 64$ bit

Implementation strategies:
- Confusion – complex relationship between key and ciphertext
- Diffusion – remove statistical links between plaintext and ciphertext
- Prevent adaptive chosen-plaintext attacks, including differential and linear cryptanalysis
- Product cipher: iterate many rounds of a weaker permutation
- Feistel structure, substitution/permutation network, key-dependent s-boxes, mix incompatible groups, transpositions, linear transformations, arithmetic operations, non-linear substitutions, . . .

Feistel structure I

Problem: Build a pseudo-random permutation $E_K : \{0, 1\}^n \leftrightarrow \{0, 1\}^n$ (invertible) using pseudo-random functions $f_{K,i} : \{0, 1\}^k \rightarrow \{0, 1\}^k$ (non-invertible) as building blocks.

Solution: Split the plaintext block $M$ ($n$ bits) into two halves $L$ and $R$ ($n/2$ bits each):

$M = L_0||R_0$

Then apply the non-invertible function $f_K$ in each round $i$ alternatingly to one of these halves, and XOR the result onto the other half, respectively:

$L_i = L_{i-1} \oplus f_{K,i}(R_{i-1})$ and $R_i = R_{i-1}$ for odd $i$

$L_i = R_{i-1} \oplus f_{K,i}(L_{i-1})$ and $L_i = L_{i-1}$ for even $i$

After applying rounds $i = 1, \ldots, r$, concatenate the two halves to form the ciphertext block $C$:

$E_K(M) = C = L_r||R_r$

Feistel structure II

$r = 3$ rounds:

```
L_0
L_1
L_2
L_3
\oplus
\oplus
\oplus
\oplus
f_{K,1}
f_{K,2}
f_{K,3}

R_0
R_1
R_2
R_3
```

Feistel structure III

Decryption:

```
L_0
L_1
L_2
L_3
\oplus
\oplus
\oplus
\oplus
f_{K,1}
f_{K,2}
f_{K,3}

R_0
R_1
R_2
R_3
```
Decryption works backwards \((i = r, \ldots, 1)\), undoing round after round, starting from the ciphertext:

\[
\begin{align*}
L_{i-1} &= L_i \oplus f_{K,i}(R_i) \quad \text{and} \quad R_{i-1} = R_i \quad \text{for odd } i \\
R_{i-1} &= R_i \oplus f_{K,i}(L_i) \quad \text{and} \quad L_{i-1} = L_i \quad \text{for even } i
\end{align*}
\]

This works because the Feistel structure is arranged such that during decryption of round \(i\), the input value for \(f_{K,i}\) is known, as it formed half of the output bits of round \(i\) during encryption.

**Luby–Rackoff result**

If \(f\) is a pseudo-random function, then \(r = 3\) Feistel rounds build a pseudo-random permutation and \(r = 4\) rounds build a strong pseudo-random permutation.

M. Luby, C. Rackoff: How to construct pseudorandom permutations from pseudorandom functions. CRYPTO'85, LNCS 218, http://www.springerlink.com/content/27t7330g746q2168/

The round function \(f\) expands the 32-bit input to 48 bits, XORs this with a 48-bit subkey, and applies eight carefully designed 6-bit to 4-bit substitution tables (“s-boxes”). The expansion function \(E\) makes sure that each sbox shares one input bit with its left and one with its right neighbour.

The key schedule of DES breaks the key into two 28-bit halves, which are left shifted by two bits in most rounds (only one bit in round 1,2,9,16) before 48 bits are selected as the subkey for each round.

In 1977, the US government standardized a block cipher for unclassified data, based on a proposal by an IBM team led by Horst Feistel.

DES has a block size of 64 bits and a key size of 56 bits. The relatively short key size and its limited protection against brute-force key searches immediately triggered criticism, but this did not prevent DES from becoming the most commonly used cipher for banking networks and numerous other applications for more than 25 years.

DES uses a 16-round Feistel structure. Its round function \(f\) is much simpler than a good pseudo-random function, but the number of iterations increases the complexity of the resulting permutation sufficiently.

DES was designed for hardware implementation such that the same circuit can be used with only minor modification for encryption and decryption. It is not particularly efficient in software.

Strengthening DES

Two techniques have been widely used to extend the short DES key size:

**DESX** \(2 \times 64 + 56 = 184\) bit keys:

\[
\text{DESX}_{K_1, K_2, K_3}(M) = K_1 \oplus \text{DES}_{K_2}(M \oplus K_3)
\]

**Triple DES (TDES)** \(3 \times 56 = 168\)-bits keys:

\[
\text{TDES}_K(M) = \text{DES}_{K_3}(\text{DES}^{-1}_{K_2}(\text{DES}_{K_1}(M)))
\]

\[
\text{TDES}^{-1}_K(C) = \text{DES}^{-1}_{K_1}(\text{DES}_{K_2}(\text{DES}^{-1}_{K_3}(C)))
\]

Where key size is a concern, \(K_1 = K_3\) is used \(\Rightarrow 112\) bit key. With \(K_1 = K_2 = K_3\), the TDES construction is backwards compatible to DES.

Double DES would be vulnerable to a meet-in-the-middle attack that requires only \(2^{57}\) iterations and \(2^{57}\) blocks of storage space: the known \(M\) is encrypted with \(2^{56}\) different keys, the known \(C\) is decrypted with \(2^{56}\) keys and a collision among the stored results leads to \(K_1\) and \(K_2\).

Neither extension fixes the small alphabet size of \(2^64\).

Advanced Encryption Standard (AES)

In November 2001, the US government published the new Advanced Encryption Standard (AES), the official DES successor with 128-bit block size and either 128, 192 or 256 bit key length. It adopted the “Rijndael” cipher designed by Joan Daemen and Vincent Rijmen, which offers additional block/key size combinations.

Each of the 9–13 rounds of this substitution-permutation cipher involves:

- an 8-bit s-box applied to each of the 16 input bytes
- permutation of the byte positions
- column mix, where each of the four 4-byte vectors is multiplied with a \(4 \times 4\) matrix in \(\text{GF}(2^8)\)
- XOR with round subkey

The first round is preceded by another XOR with a subkey, the last round lacks the column-mix step.

Software implementations usually combine the first three steps per byte into 16 8-bit \(\rightarrow 32\)-bit table lookups.

http://csrc.nist.gov/encryption/aes/
http://www.iaik.tu-graz.ac.at/research/krypto/AES/
Recent CPUs with AES hardware support: Intel/AMD x86 AES-NI instructions, VIA PadLock.

AES round

Illustration by John Savard, http://www.quadibloc.com/crypto/co040401.htm
Electronic Code Book (ECB) I

ECB is the simplest mode of operation for block ciphers (DES, AES). The message $M$ is cut into $m$ $n$-bit blocks:

$$M_1 \| M_2 \| \ldots \| M_m = M \| \text{padding}$$

Then the block cipher $E_K$ is applied to each $n$-bit block individually:

$$C_i = E_K(M_i) \quad i = 1, \ldots, m$$

$$C = C_1 \| C_2 \| \ldots \| C_m$$

Electronic Code Book (ECB) II

**Warning:**

Like any deterministic encryption scheme, Electronic Code Book (ECB) mode is not CPA secure. Therefore, repeated plaintext messages (or blocks) can be recognised by the eavesdropper as repeated ciphertext. If there are only few possible messages, an eavesdropper might quickly learn the corresponding ciphertext.

**Another problem:**

Plaintext block values are often not uniformly distributed, for example in ASCII encoded English text, some bits have almost fixed values. As a result, not the entire input alphabet of the block cipher is utilised, which simplifies for an eavesdropper building and using a value table of $E_K$.


Electronic Code Book (ECB) III

Plain-text bitmap:

![Plain-text bitmap](image)

DES-ECB encrypted:

![DES-ECB encrypted](image)

Randomized encryption

Any CPA secure encryption scheme must be randomized, meaning that the encryption algorithm has access to an $r$-bit random value that is not predictable to the adversary:

$$\text{Enc} : \{0,1\}^k \times \{0,1\}^r \times \{0,1\}^l \rightarrow \{0,1\}^m$$

$$\text{Dec} : \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^l$$

receives in addition to the $k$-bit key and $l$-bit plaintext also an $r$-bit random value, which it uses to ensure that repeated encryption of the same plaintext is unlikely to result in the same $m$-bit ciphertext.

With randomized encryption, the ciphertext will be longer than the plaintext: $m > l$, for example $m = r + l$.

Given a fixed-length pseudo-random function $F$, we could encrypt a variable-length message $M = M_1 \| M_2 \| \ldots \| M_n$ by applying $F_{\text{PRF}}$ to its individual blocks $M_i$, and the result will still be CPA secure:

$$\text{Enc}_K(M) = (R_1, E_K(R_1) \oplus M_1, R_2, E_K(R_2) \oplus M_2, \ldots, R_n, E_K(R_n) \oplus M_n)$$

But this doubles the message length!

Several efficient “modes of operation” have been standardized for use with block ciphers to provide CPA-secure encryption schemes for arbitrary-length messages.
Cipher Block Chaining (CBC) I

The Cipher Block Chaining mode is one way of constructing a CPA-secure randomized encryption scheme from a block cipher $E_K$.

1. Pad the message $M$ and split it into $m$ $n$-bit blocks, to match the alphabet of the block cipher used:

\[ M_1 \| M_2 \| \ldots \| M_m = M \| \text{padding} \]

2. Generate a random, unpredictable $n$-bit initial vector (IV) $C_0$.

3. Starting with $C_0$, XOR the previous ciphertext block into the plaintext block before applying the block cipher:

\[ C_i = E_K(M_i \oplus C_{i-1}) \quad \text{for } 0 < i \leq m \]

4. Output the $(m+1) \times n$-bit cipher text

\[ C = C_0 \| C_1 \| \ldots \| C_m \]

(which starts with the random initial vector)

| $M_i$ | $E_K$ | $C_i$ |

Plain-text bitmap:

![Plain-text bitmap](https://example.com/plain-text-bitmap.png)

DES-CBC encrypted:

![DES-CBC encrypted](https://example.com/des-cbc-encrypted.png)

Cipher Block Chaining (CBC) II

The input of the block cipher $E_K$ is now uniformly distributed.

Expect a repetition of block cipher input after around $\sqrt{2^n} \approx 2^{n/2}$ blocks have been encrypted with the same key $K$, where $n$ is the block size in bits (→ birthday paradox). Change $K$ well before that.

| $M_1$ | $E_K$ | $C_1$ |
| $M_2$ | $E_K$ | $C_2$ |
| $\ldots$ | $E_K$ | $C_m$ |

Cipher Feedback Mode (CFB)

As in CBC, $C_0$ is a randomly selected, unpredictable initial vector, the entropy of which will propagate through the entire ciphertext.

This variant has three advantages over CBC that can help to reduce latency:

- The blockcipher step needed to derive $C_i$ can be performed before $M_i$ is known.
- Incoming plaintext bits can be encrypted and output immediately; no need to wait until another $n$-bit block is full.
- No padding of last block needed.
Output Feedback Mode (OFB)

Output Feedback Mode is a stream cipher seeded by the initial vector:

1. Split the message into \( m \) blocks (blocks \( M_1, \ldots, M_{m-1} \) each \( n \)-bit long, \( M_m \) may be shorter, no padding required):
   \[
   M_1 \| M_2 \| \ldots \| M_m = M
   \]
2. Generate a unique \( n \)-bit initial vector (IV) \( C_0 \).
3. Start with \( R_0 = C_0 \), then iterate
   \[
   \begin{align*}
   R_i &= E_K(R_{i-1}) \\
   C_i &= M_i \oplus R_i 
   \end{align*}
   \]
   for \( 0 < i \leq m \). From \( R_m \) use only the leftmost bits needed for \( M_m \).
4. Output the cipher text \( C = C_0 \| C_1 \| \ldots \| C_m \)

Again, the key \( K \) should be replaced before in the order of \( 2^n \) \( n \)-bit blocks have been generated.

Unlike with CBC or CFB, the IV does not have to be unpredictable or random (it can be a counter), but it must be very unlikely that the same IV is ever used again or appears as another value \( R_i \), while the same key \( K \) is still used.

Counter Mode (CTR)

This mode is also a stream cipher. It obtains the pseudo-random bit stream by encrypting an easy to generate sequence of mutually different blocks \( T_1, T_2, \ldots, T_m \), such as the block counter \( i \) plus some offset \( O \), encoded as an \( n \)-bit binary value:

\[
C_i = M_i \oplus E_K(T_i), \quad T_i = (O + i), \quad \text{for } 0 < i \leq m
\]

Choose \( O \) such that probability of reusing any \( T_i \) under the same \( K \) is negligible. Send offset \( O \) as initial vector \( C_0 = \langle O \rangle \).

Notation \( \langle i \rangle \) here means "\( n \)-bit binary representation of integer \( i \)", where \( n \) is block length of \( E_K \).

Advantages:

- allows fast random access
- both encryption and decryption can be parallelized
- low latency
- no padding required
- no risk of short cycles

Today, Counter Mode is generally preferred over CBC, CFB, and OFB. Alternatively, the \( T_i \) can also be generated by a maximum-length linear-feedback shift register (replacing the operation \( O + i \) in \( Z_{2^n} \) with \( O(x) \cdot x^i \) in \( GF(2^n) \) to avoid slow carry bits).

Security against chosen-ciphertext attacks (CCA)

Private-key encryption scheme \( \Pi = (Gen, Enc, Dec) \), \( M = \{0,1\}^m \), security parameter \( \ell \).

Experiment/game \( PrivK_{A}^{\text{CCA}}(\ell) \):

\[
1^\ell \xrightarrow{} b \in \{0,1\} \quad \xleftarrow{} K \leftarrow Gen(1^\ell) \quad \xleftarrow{} C_0 \leftarrow Enc_K(M_0) \quad \xleftarrow{} M' \leftarrow Dec_K(C'_0) \quad b' \xrightarrow{} A \xleftarrow{} \text{adversary}
\]

Setup:

- handling of \( \ell \), \( b \), \( K \) as before

Rules for the interaction:

1. The adversary \( A \) is given oracle access to \( Enc_K \) and \( Dec_K \):
   - \( A \) outputs \( M' \), gets \( Enc_K(M') \), outputs \( C' \), gets \( Dec_K(C') \), . . .
2. The adversary \( A \) outputs a pair of messages: \( M_0, M_1 \in \{0,1\}^m \).
3. The challenger computes \( C \leftarrow Enc_K(M_b) \) and returns \( C \) to \( A \).
4. The adversary \( A \) continues to have oracle access to \( Enc_K \) and \( Dec_K \) but is not allowed to ask for \( Dec_K(C) \).

Finally, \( A \) outputs \( b' \). If \( b' = b \) then \( A \) has succeeded \( \Rightarrow PrivK_{A}^{\text{CCA}}(\ell) = 1 \).
Malleability

We call an encryption scheme $(\text{Gen}, \text{Enc}, \text{Dec})$ malleable if an adversary can modify the ciphertext $C$ in a way that causes a predictable/useful modification to the plaintext $M$.

**Example:** stream ciphers allow adversary to XOR the plaintext $M$ with arbitrary value $X$:

- Sender:\quad $C = \text{Enc}_K(M) = (R, F_K(R) \oplus M)$
- Adversary:\quad $C' = (R, (F_K(R) \oplus M) \oplus X)$
- Recipient:\quad $M' = \text{Dec}_K(C') = F_K(R) \oplus ((F_K(R) \oplus M) \oplus X)$

\[ M \oplus X \]

Malleable encryption schemes are usually not CCA secure.

CBC, OFB, and CTR are all malleable and not CCA secure.

Malleability is not necessarily a bad thing. If carefully used, it can be an essential building block to privacy-preserving technologies such as digital cash or anonymous electronic voting schemes.

Homomorphic encryption schemes are malleable by design, providing anyone not knowing the key a means to transform the ciphertext of $M$ into a valid encryption of $f(M)$ for some restricted class of transforms $f$.

MAC security definition: existential unforgeability

Message authentication code $\Pi = (\text{Gen}, \text{Mac}, \text{Vrfy})$, $M = \{0,1\}^*$, security parameter $\ell$.

**Experiment/game Mac-forge$_\mathcal{A},\Pi(\ell)$:**

\[
\begin{align*}
1^\ell & \quad \left\{ \begin{array}{l}
K \leftarrow \text{Gen}(1^\ell) \\
T^i \leftarrow \text{Mac}_K(M^i)
\end{array} \right. \\
b & \quad \left\{ \begin{array}{l}
M^1, M^2, \ldots, M^\ell \\
T^1, T^2, T^\ell \\
M, T
\end{array} \right. \\
\mathcal{A} & \quad \left\{ \begin{array}{l}
M^1, M^2, \ldots, M^\ell \\
M, T
\end{array} \right. \\
1^\ell & \quad \left\{ \begin{array}{l}
b := \text{Vrfy}_K(M, T)
\end{array} \right.
\end{align*}
\]

- challenger generates random key $K \leftarrow \text{Gen}(1^\ell)$
- adversary $\mathcal{A}$ is given oracle access to $\text{Mac}_K(\cdot)$; let $Q = \{M^1, \ldots, M^\ell\}$ denote the set of queries that $\mathcal{A}$ asks the oracle
- adversary outputs $(M, T)$
- the experiment outputs 1 if $\text{Vrfy}_K(M, T) = 1$ and $M \not\in Q$

**Definition:** A message authentication code $\Pi = (\text{Gen}, \text{Mac}, \text{Vrfy})$ is existentially unforgeable under an adaptive chosen-message attack ("secure") if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function $\text{negl}$ such that

\[
P(\text{Mac-forge}_{\mathcal{A},\Pi}(\ell) = 1) \leq \text{negl}(\ell)
\]

MACs versus security protocols

MACs prevent adversaries forging new messages. But adversaries can still

1. replay messages seen previously ("pay £1000ns", old CCTV image)
2. drop or delay messages ("smartcard revoked")
3. reorder a sequence of messages
4. redirect messages to different recipients

A security protocol is a higher-level mechanism that can be built using MACs, to prevent such manipulations. This usually involves including into each message additional data before calculating the MAC, such as

- nonces
  - message sequence counters
  - message timestamps and expiry times
  - random challenge from the recipient
  - MAC of the previous message

- identification of source, destination, purpose, protocol version
- "heartbeat" (regular message to confirm sequence number)

Security protocols also need to define unambiguous syntax for such message fields, delimiting them securely from untrusted payload data.
Stream authentication

Alice and Bob want to exchange a sequence of messages \(M_1, M_2, \ldots\). They want to verify not just each message individually, but also the integrity of the entire sequence received so far.

One possibility: Alice and Bob exchange a private key \(K\) and then send

\[
\begin{align*}
A \to B &: (M_1, T_1) \quad \text{with } T_1 = \text{Mac}_K(M_1, 0) \\
B \to A &: (M_2, T_2) \quad \text{with } T_2 = \text{Mac}_K(M_2, T_1) \\
A \to B &: (M_3, T_3) \quad \text{with } T_3 = \text{Mac}_K(M_3, T_2) \\
& \vdots \\
B \to A &: (M_{2i}, T_{2i}) \quad \text{with } T_{2i} = \text{Mac}_K(M_{2i}, T_{2i-1}) \\
A \to B &: (M_{2i+1}, T_{2i+1}) \quad \text{with } T_{2i+1} = \text{Mac}_K(M_{2i+1}, T_{2i}) \\
& \vdots
\end{align*}
\]

Mallory can still delay messages or replay old ones. Including in addition unique transmission timestamps in the messages (in at least \(M_1\) and \(M_2\)) allows the recipient to verify their “freshness” (using a secure, accurate local clock).

MAC using a pseudo-random function

Let \(F\) be a pseudo-random function.

- **Gen**: on input \(1^l\) choose \(K \in_R \{0, 1\}^l\) randomly
- **Mac**: read \(K \in \{0, 1\}^l\) and \(M \in \{0, 1\}^m\), then output \(T := F_K(M) \in \{0, 1\}^n\)
- **Vrfy**: read \(K \in \{0, 1\}^l, M \in \{0, 1\}^m, T \in \{0, 1\}^n\), then output 1 iff \(T = F_K(M)\).

If \(F\) is a pseudo-random function, then \((\text{Gen}, \text{Mac}, \text{Vrfy})\) is existentially unforgeable under an adaptive chosen message attack.

MAC using a block cipher: CBC-MAC

Blockcipher \(E : \{0, 1\}^\ell \times \{0, 1\}^m \rightarrow \{0, 1\}^m\)

\[
\begin{align*}
M_1 & \quad M_2 & \quad \ldots & \quad M_n \\
E_K & \quad E_K & \quad \ldots & \quad E_K \\
\oplus & \quad \oplus & \quad \ldots & \quad \oplus \\
\text{CBC-MAC}_{E_K}(M)
\end{align*}
\]

Similar to CBC: \(IV = 0^m\), last ciphertext block serves as tag.

Provides existential unforgeability, but only for fixed message length \(n\):

Adversary asks oracle for \(T^1 := \text{CBC-MAC}_{E_K}(M^1) = E_K(M^1)\) and then presents \(M = M^1||T^1\) and \(T := \text{CBC-MAC}_{E_K}(M) = E_K((M^1 \oplus T^1) \oplus E_K(M^1)) = E_K((M^1 \oplus T^1) \oplus T^1) = E_K(M^1) = T^1\).

Variable-length MAC using a block cipher: ECBC-MAC

Blockcipher \(E : \{0, 1\}^\ell \times \{0, 1\}^m \rightarrow \{0, 1\}^m\)

Padding: \(M||10^p\) where \(p = m - (|M| \mod m) - 1\)

\[
\begin{align*}
M_1 & \quad M_2 & \quad \ldots & \quad M_n \\
E_{K_1} & \quad E_{K_1} & \quad \ldots & \quad E_{K_1} \\
\oplus & \quad \oplus & \quad \ldots & \quad \oplus \\
\text{ECBC-MAC}_{E_{K_1}, K_2}(M)
\end{align*}
\]

Disadvantages:

- up to two additional applications of block cipher
- need to rekey block cipher
- added block if \(m\) divides \(|M|\)
Variable-length MAC using a block cipher: CMAC

Blockcipher $E: \{0,1\}^\ell \times \{0,1\}^m \to \{0,1\}^m$ (typically AES: $m = 128$)

Derive subkeys $K_1, K_2 \in \{0,1\}^m$ from key $K \in \{0,1\}^\ell$:

- $K_0 := E_K(0)$
- if $\text{msb}(K_0) = 0$ then $K_1 := (K_0 \ll 1) \oplus J$ else $K_1 := (K_0 \ll 1)$
- if $\text{msb}(K_1) = 0$ then $K_2 := (K_1 \ll 1) \oplus J$ else $K_2 := (K_1 \ll 1)$

This merely clocks a linear-feedback shift register twice, or equivalently multiplies a value in $GF(2^m)$ twice with $x$. $J$ is a fixed constant (generator polynomial), $\ll$ is a left shift.

CMAC algorithm:

\[
M_1 \| M_2 \| \ldots \| M_n := M \\
\text{if } r = m \text{ then } M_n := K_1 \oplus M_n \\
\text{else } M_n := K_2 \oplus (M_n \| 10^{m-r-1})
\]

return CBC-MAC$_K(M_1 \| M_2 \| \ldots \| M_n)$

Provides existential unforgeability, without the disadvantages of ECBC.

NIST SP 800-38B, RFC 4493

A one-time MAC (Carter-Wegman)

The only MAC scheme is very fast and unconditionally secure, but only if the key is used to secure only a single message.

Let $F$ be a large finite field (e.g. $\mathbb{Z}_{2^{128}+51}$ or $GF(2^{128})$).

- Pick a random key pair $K = (K_1, K_2) \in F^2$
- Split padded message $P$ into blocks $P_1, \ldots, P_m \in F$
- Evaluate the following polynomial over $F$ to obtain the MAC:

\[
\text{OT-MAC}_{K_1, K_2}(P) = K_1^{m+1} + P_m K_1^m + \cdots + P_2 K_1^2 + P_1 K_1 + K_2
\]

Converted into a computationally secure many-time MAC:

- Pseudo-random function/permutation $E_K: F \to F$
- Pick per-message random value $R \in F$
- $\text{CW-MAC}_{K_1, K_2}(P) = (R, K_1^{m+1} + P_m K_1^m + \cdots + P_2 K_1^2 + P_1 K_1 + E_K(R))$

Birthday attack against CBC-MAC, ECBC-MAC, CMAC

Let $E$ be an $m$-bit block cipher, used to build MAC$_K$ with $m$-bit tags.

Birthday/collision attack:

- Make $t = \sqrt{2^m}$ oracle queries for $T := \text{MAC}_K(\langle i \rangle \| R_i \| \langle 0 \rangle)$ with $R_i \in \{0,1\}^m$, $1 \leq i \leq t$.
- Here $\langle i \rangle \in \{0,1\}^m$ is the $m$-bit binary integer notation for $i$.
- Look for collision $T^i = T^j$ with $i \neq j$.
- Ask oracle for $T^i := \text{MAC}_K(\langle i \rangle \| R_i \| \langle 1 \rangle)$.
- Present $M := \langle j \rangle \| R_j \| \langle 1 \rangle$ and $T := T^i = \text{MAC}_K(M)$.

The same intermediate value $C_2$ occurs while calculating the MAC of $\langle i \rangle \| R_i \| \langle 0 \rangle, \langle j \rangle \| R_j \| \langle 0 \rangle$.

Possible workaround:
Truncate MAC result to less than $m$ bits, such that adversary cannot easily spot collisions in $C_2$ from $C_1$.

Solution: big enough $m$. 

Ciphertext integrity

Private-key encryption scheme $\Pi = (Gen, Enc, Dec)$, Dec can output error: ⊥. Experiment/game $Cl_{A,N}(\ell)$:

1. challenger generates random key $K \leftarrow Gen(1^\ell)$
2. adversary $A$ is given oracle access to $Enc_K(\cdot)$, let $Q = \{C^1, \ldots, C^t\}$ denote the set of query answers that $A$ got from the oracle
3. adversary outputs $C$
4. the experiment outputs 1 if $Dec_K(C) \neq \perp$ and $C \not\in Q$.

**Definition:** An encryption scheme $\Pi = (Gen, Enc, Dec)$ provides ciphertext integrity if for all probabilistic polynomial-time adversaries $A$ there exists a negligible function $negl$ such that

$$P(Cl_{A,N}(\ell) = 1) \leq negl(\ell)$$

Combining encryption and message authentication

**Warning:** Not every way of combining a CPA-secure encryption scheme (to achieve privacy) and a secure message authentication code (to prevent forgery) will necessarily provide CPA security:

- **Encrypt-and-authenticate:** $(Enc_{K_E}(M), \text{Mac}_{K_M}(M))$
- **Likely to be CPA secure:** MAC may leak information about $M$.

- **Authenticate-then-encrypt:** $Enc_{K_E}(M || \text{Mac}_{K_M}(M))$
- **May not be CPA secure:** the recipient first decrypts the received message with $Dec_{K_E}$, then parses the result into $M$ and $\text{Mac}_{K_M}(M)$ and finally tries to verify the latter. A malleable encryption scheme, combined with a parser that reports syntax errors, may reveal information about $M$.

- **Encrypt-then-authenticate:** $(Enc_{K_E}(M), \text{Mac}_{K_M}(Enc_{K_E}(M)))$
- **Secure:** provides both CCA security and existential unforgeability.

**Note:** CCA security alone does not imply existential unforgeability.

Autenticated encryption

**Definition:** An encryption scheme $\Pi = (Gen, Enc, Dec)$ provides authenticated encryption if it provides both CPA security and ciphertext integrity.

Such an encryption scheme will then also be CCA secure.

**Example:**

Private-key encryption scheme $\Pi_E = (Gen_E, Enc, Dec)$

Message authentication code $\Pi_M = (Gen_M, \text{Mac, Vrfy})$

Encryption scheme $\Pi' = (Gen', Enc', Dec')$:
1. $Gen'(1^\ell) := (K_E, K_M)$ with $K_E \leftarrow Gen_E(1^\ell)$ and $K_M \leftarrow Gen_M(1^\ell)$
2. $Enc'_{(K_E, K_M)}(M) := (C, T)$ with $C \leftarrow Enc_{K_E}(M)$ and $T \leftarrow \text{Mac}_{K_M}(C)$
3. $Dec'$ on input of $(K_E, K_M)$ and $(C, T)$ first check if $Vrfy_{K_M}(C, T) = 1$. If yes, output $Dec_{K_E}(C)$, if no output ⊥.

If $\Pi_E$ is a CPA-secure private-key encryption scheme and $\Pi_M$ is a secure message authentication code with unique tags, then $\Pi'$ is a CCA-secure private-key encryption scheme.

A message authentication code has unique tags, if for every $K'$ and every $M$ there exists a unique value $T$, such that $Vrfy_K(M, T) = 1$.

Padding oracle

TLS record protocol:

Recipient steps: CBC decryption, then checks and removes padding, finally checks MAC.

Padding: append $n$ times byte $n$ ($1 \leq n \leq 16$)

Padding syntax error and MAC failure (used to be) distinguished in error messages.

<table>
<thead>
<tr>
<th>$C_0 = IV$</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$D_K$</td>
<td>$D_K$</td>
<td>$D_K$</td>
</tr>
<tr>
<td>$M_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_3</td>
<td></td>
<td>\text{pad}$</td>
<td></td>
</tr>
</tbody>
</table>

If the recipient does not even attempt to decrypt $M$ unless the MAC has been verified successfully, this method can also prevent some side-channel attacks.
Padding oracle (cont’d)

Attacker has \( C_0, \ldots, C_3 \) and tries to get \( M_2 \):

- truncate ciphertext after \( C_2 \)
- \( a = \) actual last byte of \( M_2 \), \( g = \) attacker’s guess of \( a \) (try all \( g \in \{0, \ldots, 255\} \))
- XOR the last byte of \( C_1 \) with \( g \oplus 0x01 \)
- last byte of \( M_2 \) is now \( a \oplus g \oplus 0x01 \)
- \( g = a \): padding correct ⇒ MAC failed error
- \( g \neq a \): padding syntax error (high prob.)

Then try \( 0x02 \) \( 0x02 \) and so on.

Serge Vaudenay: Security flaws induced by CBC padding, EUROCRYPT 2002

Galois Counter Mode (GCM)

CBC and CBC-MAC used together require different keys, resulting in two encryptions per block of data.

Galois Counter Mode is a more efficient authenticated encryption technique that requires only a single encryption, plus one XOR \( \oplus \) and one multiplication \( \otimes \), per block of data:

\[
\begin{align*}
C_i &= M_i \oplus E_K(O + i) \\
G_i &= (G_{i-1} \oplus C_i) \otimes H, \quad G_0 = A \otimes H, \quad H = E_K(0) \\
\text{GMAC}_{E_K}(A, C) &= ((G_n \oplus (\text{len}(A)\|\text{len}(C))) \otimes H) \oplus E_K(O)
\end{align*}
\]

\( A \) is authenticated, but not encrypted (e.g., message header).

The multiplication \( \otimes \) is over the Galois field \( \text{GF}(2^{128}) \): block bits are interpreted as coefficients of binary polynomials of degree 127, and the result is reduced modulo \( x^{128} + x^7 + x^2 + x + 1 \).

This is like 128-bit modular integer multiplication, but without carry bits, and therefore faster in hardware.


1 Historic ciphers
2 Perfect secrecy
3 Semantic security
4 Block ciphers
5 Modes of operation
6 Message authenticity
7 Authenticated encryption
8 Secure hash functions
9 Secure hash applications
10 Key distribution problem
11 Number theory and group theory
12 Discrete logarithm problem
13 RSA trapdoor permutation
14 Digital signatures
Secure hash functions

A secure, collision-resistant hash function is designed to make it infeasible for an adversary who knows the implementation of the hash function to find any collision

\[ h(x) = h(y) \text{ with } x \neq y \]

Examples for applications of secure hash functions:
- message digest for efficient calculation of digital signatures
- fast message-authentication codes (HMAC)
- tamper-resistant checksums of files
- git commit identifiers
- P2P file sharing identifiers
- key derivation functions
- password verification
- hash chains (e.g., Bitcoin, timestamping services)
- commitment protocols

Collision resistance – a formal definition

A hash function is a pair of probabilistic polynomial-time (PPT) algorithms \((\mathcal{G}, \mathcal{H})\) where

- \(\mathcal{G}\) reads a security parameter \(1^n\) and outputs a key \(s\).
- \(\mathcal{H}\) reads key \(s\) and input string \(x \in \{0,1\}^\ast\) and outputs \(H_s(x) \in \{0,1\}^\ell(n)\) (where \(n\) is a security parameter implied by \(s\))

Formally define collision resistance using the following game:

1. Challenger generates a key \(s = \mathcal{G}(1^n)\)
2. Challenger passes \(s\) to adversary \(\mathcal{A}\)
3. \(\mathcal{A}\) replies with \(x, x'\)
4. \(\mathcal{A}\) has found a collision iff \(H_s(x) = H_s(x')\) and \(x \neq x'\)

A hash function \((\mathcal{G}, \mathcal{H})\) is collision resistant if for all PPT adversaries \(\mathcal{A}\) there is a negligible function \(\negl\) such that

\[ \Pr[\mathcal{A} \text{ finds a collision}] \leq \negl(n) \]

Recall “negligible function” (Security I): approaches zero faster than any polynomial, e.g. \(2^{-n}\).

A fixed-length compression function is only defined on \(x \in \{0,1\}^\ell(n)\) with \(\ell(n) > \ell(n)\).
Unkeyed hash functions

Commonly used collision-resistant hash functions (SHA-256, etc.) do not use a key \( s \). They are fixed functions of the form \( h : \{0,1\}^* \rightarrow \{0,1\}^t \).

Why do we need \( s \) in the security definition?

Any fixed function \( h \) where the size of the domain (set of possible input values) is greater than the range (set of possible output values) will have collisions \( x, x' \). There always exists a constant-time adversary \( A \) that just outputs these hard-wired values \( x, x' \).

Therefore, a complexity-theoretic security definition must depend on a key \( s \) (and associated security parameter \( 1^n \)). Then \( H \) becomes a recipe for defining ever new collision-resistant fixed functions \( H_s \).

So in practice, \( s \) is a publicly known fixed constant, embedded in the secure hash function \( h \).

Also, without any security parameter \( n \), we could not use the notion of a negligible function.

Merkle–Damgård construction

**Wanted:** variable-length hash function (Gen, \( H \)).

**Given:** \((\text{Gen}, C)\), a fixed-length hash function with \( C: \{0,1\}^2n \rightarrow \{0,1\}^n \) ("compression function")

**Input of \( H \):** key \( s \), string \( x \in \{0,1\}^L \) with length \( L < 2^n \)

\begin{enumerate}
\item Pad \( x \) to length divisible by \( n \) by appending "0" bits, then split the result into \( B = \lceil \frac{L}{n} \rceil \) blocks of length \( n \) each:
\[
x\|0^{n\lceil \frac{L}{n} \rceil - L} = x_1\|x_2\|x_3\| \ldots \|x_{B-1}\|x_B
\]
\item Append a final block \( x_{B+1} = \langle L \rangle \), which contains the \( n \)-bit binary representation of input length \( L = |x| \).
\item Set \( z_0 := 0^n \) (initial vector, IV)
\item Compute \( z_i := C_s(z_{i-1}\|x_i) \) for \( i = 1, \ldots, B + 1 \)
\item Output \( H_s(x) := z_{B+1} \)
\end{enumerate}

Weaker properties implied by collision resistance

**Second-preimage resistance**

For a given \( s \) and output value \( y \), it is infeasible for any polynomial-time adversary to find \( x' \) with \( H_s(x') = y \) (except with negligible probability).

If there existed a PPT adversary \( A \) that can break the second-preimage resistance of \( H_s \), then \( A \) can also break its collision resistance. Therefore, collision resistance implies second-preimage resistance.

**Preimage resistance**

For a given \( s \) and input value \( x \), it is infeasible for any polynomial-time adversary to find \( s \) with \( H_s(x) = y \) (except with negligible probability).

If there existed a PPT adversary \( A \) that can break the pre-image resistance of \( H_s \), then \( A \) can also break its second-preimage resistance (with high probability). Therefore, either collision resistance or second-preimage resistance imply preimage resistance.

How?

Note: collision resistance does not prevent \( H_s \) from leaking information about \( x \) (\( \to \) CPA).

\[
x\|0^n\lceil \frac{L}{n} \rceil - L = x_1\|x_2\|x_3\| \ldots \|x_{B-1}\|x_B
\]

\[
x'\|0^n\lceil \frac{L}{n} \rceil - L' = x'_1\|x'_2\|x'_3\| \ldots \|x'_{B-1}\|x'_{B'}
\]
Merkle–Damgård construction – security proof

If the fixed-length compression function \( C \) is collision resistant, so will be the variable-length hash function \( H \) resulting from the Merkle–Damgård construction.

**Proof outline:**
Assume \( C \) is collision resistant, but \( H \) is not, because some PPT adversary \( A \) outputs \( x \neq x' \) with \( H_s(x) = H_s(x') \).

Let \( x_1, \ldots, x_B \) be the \( n \)-bit blocks of padded \( L \)-bit input \( x \), and \( x_1', \ldots, x_B' \) be those of \( L' \)-bit input \( x' \), and \( x_{B+1} = \langle L \rangle \), \( x_{B'+1} = \langle L' \rangle \).

**Case \( L \neq L' \):** Then \( x_{B+1} \neq x_{B'+1} \) but \( H_s(x) = H_s(x') = C_s(z_B || x_{B+1}) = C_s(z_B' || x_{B'+1}) = z_{B'+1} = H_s(x') \), which is a collision in \( C_s \).

**Case \( L = L' \):** Now \( B = B' \). Let \( i \in \{1, \ldots, B + 1\} \) be the largest index where \( z_{i-1} || x_i \neq z'_{i-1} || x'_i \). (Such \( i \) exists as due to \( |x| = |x'| \) and \( x \neq x' \) there will be at least one \( 1 \leq j \leq B \) with \( x_j \neq x'_j \).) Then \( z_k = z'_k \) for all \( k \in \{i, \ldots, B + 1\} \) and \( z_i = C_s(z_{i-1} || x_i) = C_s(z'_{i-1} || x'_i) = z'_i \) is a collision in \( C_s \).

So \( C_s \) was not collision resistant, invalidating the assumption.

SHA-1 structure

Merkle–Damgård construction, block length \( n = 512 \) bits.

**Compression function:**
- Input = 160 bits = five 32-bit registers A–E
- each block = 16 32-bit words \( W_0, \ldots, W_{15} \)
- LFSR extends that sequence to 80 words: \( W_{16}, \ldots, W_{79} \)
- 80 rounds, each fed one \( W_i \)
- Round constant \( K_i \) and non-linear function \( F_i \) change every 20 rounds.
- four 32-bit additions \( \oplus \) and two 32-bit rotations per round, 2–5 32-bit Boolean operations for \( F \).
- finally: 32-bit add round 0 input to round 79 output (Davies–Meyer)

**One round:**

```
A B C D E
\[ F \]
```

```
\[ W_i \]
\[ K_i \]
```

Random oracle model

Many applications of secure hash functions have no security proof that relies only on the collision resistance of the function used.

The known security proofs require instead a much stronger assumption, the strongest possible assumption one can make about a hash function:

**Random oracle**
- A random oracle \( H \) is a device that accepts arbitrary length strings \( X \in \{0,1\}^* \) and consistently outputs for each a value \( H(X) \in \{0,1\}^t \) which it chooses uniformly at random.
- Once it has chosen an \( H(X) \) for \( X \), it will always output that same answer for \( X \) consistently.
- Parties can privately query the random oracle (nobody else learns what anyone queries), but everyone gets the same answer if they query the same value.
- No party can infer anything about \( H(X) \) other than by querying \( X \).
Ideal cipher model

An random-oracle equivalent can be defined for block ciphers:

**Ideal cipher**

Each key $K \in \{0,1\}^\ell$ defines a random permutation $E_K$, chosen uniformly at random out of all $(2^n)!$ permutations. All parties have oracle access to both $E_K(X)$ and $E_K^{-1}(X)$ for any $(K, X)$. No party can infer any information about $E_K(X)$ (or $E_K^{-1}(X)$) without querying its value for $(K, X)$.

We have encountered random functions and random permutations before, as a tool for defining pseudo-random functions/permutations. Random oracles and ideal ciphers are different:

If a security proof is made “in the random oracle model”, then a hash function is replaced by a random oracle or a block cipher is replaced by an ideal cipher.

In other words, the security proof makes much stronger assumptions about these components: they are not just indistinguishable from random functions/permutations by any polynomial-time distinguisher, they are actually assumed to be random functions/permutations.

For some fixed pair $i, j$ with $i > j$, what is the probability of $c_i = c_j$?

A collision at query $i$ can only occur as one of these two query results:

- $E_K(X_i) = c_j \oplus X_i$
- $E_K^{-1}(Y_i) = c_j \oplus Y_i$

Each query will reveal a new uniformly distributed $\ell$-bit value, except that it may be constrained by (at most) $i - 1$ previous query results (since $E_K$ must remain a permutation).

Therefore, the ideal cipher $E$ will answer query $i$ by uniformly choosing a value out of at least $2^\ell - (i - 1)$ possible values.

Therefore, each of the above two possibilities for reaching $c_i = c_j$ can happen with probability no higher than $1/(2^\ell - (i - 1))$.

With $i \leq q < 2^{\ell/2}$ and $\ell > 1$, we have

$$P(c_i = c_j) \leq \frac{1}{2^\ell - (i - 1)} \leq \frac{1}{2^{\ell/2} - \ell/2} \leq \frac{1}{2^{\ell/2}}$$

There are $(\frac{q}{2}) < q^2/2$ pairs $j < i \leq q$, so the collision probability after $q$ queries cannot be more than $\frac{2q^2}{2^{\ell/2}} = \frac{q^2}{2^\ell}$. 

Davies–Meyer construction – security proof

If $E$ is modeled as an ideal cipher, then $C$ is a collision-resistant hash function. Any attacker $A$ making $q < 2^{\ell/2}$ oracle queries to $E$ finds a collision with probability not higher than $q^2/2^{\ell}$. (negligible)

**Proof:** Attacker $A$ tries to find $(K, X), (K', X')$ with $E_K(X) \oplus X = E_K'(X') \oplus X'$. We assume that, before outputting $(K, X), (K', X')$, $A$ has previously made queries to learn $E_K(X)$ and $E_K'(X')$. We also assume (wlog) $A$ never makes redundant queries, so having learnt $Y = E_K(X)$, $A$ will not query $E_K^{-1}(Y)$ and vice versa.

The $i$-th query $(K_i, X_i)$ to $E$ only reveals

$$c_i = C_i(K_i, X_i) = E_K(K_i) \oplus X_i$$

A query to $E^{-1}$ instead would only reveal $E_K^{-1}(Y_i) = X_i$ and therefore

$$c_i = C_i(K_i, X_i) = Y_i \oplus E_K^{-1}(Y_i)$$

$A$ needs to find $c_i = c_j$ with $i > j$.

Random oracle model – controversy

Security proofs that replace the use of a hash function with a query to a random oracle (or a block cipher with an ideal cipher) remain controversial.

**Cons**

- Real hash algorithms are publicly known. Anyone can query them privately as often as they want, and look for shortcuts.
- No good justification to believe that proofs in the random oracle model say anything about the security of a scheme when implemented with practical hash functions (or pseudo-random functions/permutations).
- No good criteria known to decide whether a practical hash function is “good enough” to instantiate a random oracle.

**Pros**

- A random-oracle model proof is better than no proof at all.
- Many efficient schemes (especially for public-key crypto) only have random-oracle proofs.
- No history of successful real-world attacks against schemes with random-oracle security proofs.
- If such a scheme were attacked successfully, it should still be fixable by using a better hash function.
Sponge functions

Another way to construct a secure hash function $H(M) = Z$:

\[(r + c)\text{-bit internal state, XOR } r\text{-bit input blocks at a time, stir with pseudo-random permutation } f, \text{ output } r\text{-bit output blocks at a time.}\]

Versatile: secure hash function (variable input length) and stream cipher (variable output length)

Advantage over Merkle–Damgård: internal state > output, flexibility.

SHA-3

Latest NIST secure hash algorithm

- Sponge function with $b = r + c = 1600 = 5 	imes 5 	imes 64$ bits of state
- Standardized (SHA-2 compatible) output sizes:
  $\ell \in \{224, 256, 384, 512\}$ bits
- Internal capacity: $c = 2\ell$
- Input block size: $r = b - 2\ell \in \{1152, 1088, 832, 576\}$ bits
- Padding: append $10^*1$ to extend input to next multiple of $r$

NIST also defined two related extendable-output functions (XOFs), SHAKE128 and SHAKE256, which accept arbitrary-length input and can produce arbitrary-length output. PRBG with 128 or 256-bit security.

SHA-3 standard: permutation-based hash and extendable-output functions. August 2015.

Duplex construction

A variant of the sponge construction, proposed to provide

- authenticated encryption (basic idea: $\sigma_i = C_i = M_i \oplus Z_{i-1}$)
- reseedable pseudo-random bit sequence generator
  (for post-processing and expanding physical random sources)

http://sponge.noekeon.org/SpongeDuplex.pdf

“Birthday attacks”

If a hash function outputs $\ell$-bit words, an attacker needs to try only different input values, before there is a better than 50% chance of finding a collision.

Computational security

Attacks requiring $2^{128}$ steps considered infeasible $\implies$ use hash function that outputs $\ell = 256$ bits (e.g., SHA-256). If only second pre-image resistance is a concern, shorter $\ell = 128$-bit may be acceptable.

Finding useful collisions

An attacker needs to generate a large number of plausible input plaintexts to find a practically useful collision. For English plain text, synonym substitution is one possibility for generating these:

A: Mallory is a {good,hardworking} and {honest,loyal} {employee,worker}
B: Mallory is a {lazy,difficult} and {lying,malicious} {employee,worker}

Both A and B can be phrased in $2^6$ variants each $\implies$ $2^6$ pairs of phrases.

With a 64-bit hash over an entire letter, we need only such sentences for a good chance to find a collision in steps.
A normal search for an $\ell$-bit collision uses $O(2^{\ell/2})$ memory and time.

Algorithm for finding a collision with $O(1)$ memory and $O(2^{\ell/2})$ time:

**Input:** $H : \{0,1\}^* \to \{0,1\}^\ell$

**Output:** $x \neq x'$ with $H(x) = H'(x)$

$x_0 \leftarrow \{0,1\}^{\ell+1}$

$x' := x := x_0$

$i := 0$

**loop**

$i := i + 1$

$x := H(x)$ // $x = H^i(x_0)$

$x' := H(H(x'))$ // $x' = H^{i+1}(x_0)$

until $x = x'$

**loop 1:** ends when $x'$ overtakes $x$ for the first time

⇒ $x'$ now $i$ steps ahead of $x$

⇒ $i$ is now an integer multiple of the cycle length

**loop 2:** $x$ back at start, $x'$ is $i$ steps ahead, same speed

⇒ meet at cycle entry point

**Basic idea:**

- Tortoise $x$ goes at most once round the cycle, hare $x'$ at least once
- loop 1: ends when $x'$ overtakes $x$ for the first time
- loop 2: $x$ back at start, $x'$ is $i$ steps ahead, same speed

With probability $\frac{1}{2}$ the resulting $x, x'$ ($H(g(x)) = H(g(x'))$) will differ in the last bit ⇒ collision between two texts with different meanings.

**Wikipedia:** Cycle detection

---

**Constructing meaningful collisions**

Tortoise-hare algorithm gives no direct control over content of $x, x'$.

**Solution:**

Define a text generator function $g : \{0,1\}^\ell \to \{0,1\}^*$, e.g.

- $g(0000) = \text{Mallory is a good and honest employee}$
- $g(0001) = \text{Mallory is a lazy and lying employee}$
- $g(0010) = \text{Mallory is a good and honest worker}$
- $g(0011) = \text{Mallory is a lazy and lying worker}$
- $g(0100) = \text{Mallory is a good and loyal employee}$
- $g(0101) = \text{Mallory is a lazy and malicious employee}$
- $\ldots$
- $g(1111) = \text{Mallory is a difficult and malicious worker}$

Then apply the tortoise-hare algorithm to $H(x) = h(g(x))$, if $h$ is the hash function for which a meaningful collision is required.

**Hash and MAC**

A secure hash function can be combined with a fixed-length MAC to provide a variable-length MAC $\text{Mac}_H(m)$.

More formally:

Let $\Pi = (\text{Mac}, \text{Vrfy})$ be a MAC for messages of length $\ell(n)$ and let $\Pi_H = (\text{Gen}_H, H)$ be a hash function with output length $\ell(n)$. Then define variable-length MAC $\Pi' = (\text{Gen}', \text{Mac}', \text{Vrfy}')$ as:

- $\text{Gen}'$: Read security parameter $1^n$, choose uniform $k \in \{0,1\}^n$, run $s := \text{Gen}_H(1^n)$ and return $(k, s)$.
- $\text{Mac}'$: read key $(k, s)$ and message $m \in \{0,1\}^*$, return tag $\text{Mac}_H(m)$.
- $\text{Vrfy}'$: read key $(k, s)$, message $m \in \{0,1\}^*$, tag $t$, return $\text{Vrfy}_H(k, m, t)$.

If $\Pi$ offers existential unforgeability and $\Pi_H$ is collision resistant, then $\Pi'$ will offer existential unforgeability.

**Proof outline:** If an adversary used $\text{Mac}'$ to get tags on a set $Q$ of messages, and then can produce a valid tag for $m^* \not\in Q$, then there are two cases:

- $\exists m \in Q$ with $H_s(m) = H_s(m^*) ⇒ H_s$ not collision resistant
- $\forall m \in Q: H_s(m) \neq H_s(m^*) ⇒ \text{Mac} \text{ failed existential unforgeability}$
Hash-based message authentication code

Initial idea: hash a message \( M \) prefixed with a key \( K \) to get
\[
\text{MAC}_K(M) = h(K||M)
\]

This construct is secure in the random oracle model (where \( h \) is a random function). It is also generally considered secure with fixed-length \( m \)-bit messages \( M \in \{0,1\}^m \) or with sponge-function based hash algorithm \( h \), such as SHA-3.

**Danger:** If \( h \) uses the Merkle–Damgård construction, an adversary can call the compression function again on the MAC to add more blocks to \( M \), and obtain the MAC of a longer \( M' \) without knowing the key!

To prevent such a message-extension attack, variants like
\[
\text{MAC}_K(M) = h(h(K||M))
\]
or
\[
\text{MAC}_K(M) = h(K||h(M))
\]
could be used to terminate the iteration of the compression function in a way that the adversary cannot continue. \( \Rightarrow \) HMAC

Secure commitment

**Proof of prior knowledge**

You have today an idea that you write down in message \( M \). You do not want to publish \( M \) yet, but you want to be able to prove later that you knew \( M \) already today. Initial idea: you publish \( h(M) \) today.

**Danger:** if the entropy of \( M \) is small (e.g., \( M \) is a simple choice, a PIN, etc.), there is a high risk that your adversary can invert the collision-resistant function \( h \) successfully via brute-force search.

**Solution:**
- Pick (initially) secret \( N \in \{0,1\}^{128} \) uniformly at random.
- Publish \( h(N,M) \) (as well as \( h \) and \( |N| \)).
- When the time comes to reveal \( M \), also reveal \( N \).

You can also commit yourself to message \( M \), without yet revealing it’s content, by publishing \( h(N,M) \).

Applications: online auctions with sealed bids, online games where several parties need to move simultaneously, etc.

Tuple \((N,M)\) means any form of unambiguous concatenation, e.g., \( N||M \) if length \( |N| \) is agreed.

HMAC

HMAC is a standard technique widely used to form a message-authentication code using a Merkle–Damgård-style secure hash function \( h \), such as MD5, SHA-1 or SHA-256:
\[
\text{HMAC}_K(x) = h(K \oplus \text{opad}||h(K \oplus \text{ipad}||x))
\]

Fixed padding values \( \text{ipad} \), \( \text{opad} \) extend the key to the input size of the compression function, to permit precomputation of its first iteration.

**Merkle tree**


Leaves contain hash values of files \( F_0, \ldots, F_{k-1} \). Each inner node contains the hash of its children. Only root \( h_0 \) (and number \( k \) of files) needs to be stored securely.

Advantages of tree (over naive alternative \( h_0 = h(F_0, \ldots, F_{k-1}) \)):
- Update of a file \( F_i \) requires only \( O(\log k) \) recalculations of hash values along path from \( h(F_i) \) to root (not rereading every file).
- Verification of a file requires only reading \( O(\log k) \) values in all direct children of nodes in path to root (not rereading every node).

\[
\begin{align*}
  h_0 &= h(h_1, h_2) \\
  h_1 &= h(h_3, h_4) \\
  h_3 &= h(h_7, h_8) \\
  h_7 &= h(F_0) \\
  h_4 &= h(h_9, h_{10}) \\
  h_9 &= h(F_1) \\
  h_{10} &= h(F_2) \\
  h_8 &= h(F_3) \\
  h_{11} &= h(F_4) \\
  h_{12} &= h(F_5) \\
  h_{13} &= h(F_6) \\
  h_{14} &= h(F_7) \\
  h_2 &= h(h_5, h_6) \\
  h_5 &= h(h_{11}, h_{12}) \\
  h_{15} &= h(h_{13}, h_{14}) \\
  h_6 &= h(h_{13}, h_{14}) \\
  h_{16} &= h(F_k)
\end{align*}
\]
One-time passwords from a hash chain

Generate hash chain: \( \{h\) is preimage resistant, with ASCII output\)

\[
\begin{align*}
R_0 & \leftarrow \text{random} \\
R_1 & := h(R_0) \\
\vdots \\
R_{n-1} & := h(R_{n-2}) \\
R_n & := h(R_{n-1})
\end{align*}
\]

Equivalently: \( R_i := h(h(. . . h(R_0) . . .)) = h^i(R_0) \) \( 0 < i \leq n \)

Store last chain value \( H := R_n \) on the host server. Give the remaining list \( R_{n-1}, R_{n-2}, \ldots, R_0 \) as one-time passwords to the user.

When user enters password \( R_i \), compare \( h(R_i) \frown H \). If they match:
- Update \( H := R_i \) on host
- grant access to user


Broadcast stream authentication

Alice sends to a group of recipients a long stream of messages \( M_1, M_2, \ldots, M_n \). They want to verify Alice’s signature on each packet immediately upon arrival, but it is too expensive to sign each message.

Alice calculates

\[
\begin{align*}
C_1 & = h(C_2, M_1) \\
C_2 & = h(C_3, M_2) \\
C_3 & = h(C_4, M_3) \\
\vdots \\
C_{n-2} & = h(C_{n-1}, M_{n-2}) \\
C_{n-1} & = h(C_n, M_{n-1}) \\
C_n & = h(0, M_n)
\end{align*}
\]

and then broadcasts the stream

\[
C_1, \text{Sign}(C_1), (C_2, M_1), (C_3, M_2), \ldots, (0, M_n).
\]

Only the first check value is signed, all other packets are bound together in a hash chain that is linked to that single signature.

Problem: Alice needs to know \( M_n \) before she can start to broadcast \( C_1 \). Solution: TESLA

Timed Efficient Stream Loss-tolerant Authentication

TESLA uses a hash chain to authenticate broadcast data, without any need for a digital signature for each message.

Timed broadcast of data sequence \( M_1, M_2, \ldots, M_n \):

- \( t_0 \): \( \text{Sign}(R_0), R_0 \) where \( R_0 = h(R_1) \)
- \( t_1 \): \( \text{Mac}_{R_0}(M_1, M_1, R_1) \) where \( R_1 = h(R_2) \)
- \( t_2 \): \( \text{Mac}_{R_1}(M_2, M_2, R_2) \) where \( R_2 = h(R_3) \)
- \( t_3 \): \( \text{Mac}_{R_2}(M_3, M_3, R_3) \) where \( R_3 = h(R_4) \)
- \( t_4 \): \( \text{Mac}_{R_3}(M_4, M_4, R_4) \) where \( R_4 = h(R_5) \)
- \( \ldots \)

Each \( R_i \) is revealed at a pre-agreed time \( t_i \). The MAC for \( M_i \) can only be verified after \( t_{i+1} \) when key \( R_{i+1} \) is revealed.

By the time the MAC key is revealed, everyone has already received the MAC, therefore the key can no longer be used to spoof the message.

Hash chains, block chains, time-stamping services

Clients continuously produce transactions \( M_i \) (e.g., money transfers).

**Block-chain time-stamping service:** receives client transactions \( M_i \), may order them by dependency, validates them (payment covered by funds?), batches them into groups

\[
\begin{align*}
G_1 & = (M_1, M_2, M_3) \\
G_2 & = (M_4, M_5, M_6, M_7) \\
G_3 & = (M_8, M_9) \\
\vdots \\
G_i & = (M_{2i-1}, M_{2i}, M_{2i+1})
\end{align*}
\]

and then publishes the hash chain (with timestamps \( t_i \))

\[
\begin{align*}
B_1 & = (G_1, t_1, 0) \\
B_2 & = (G_2, t_2, h(B_1)) \\
B_3 & = (G_3, t_3, h(B_2)) \\
\vdots \\
B_i & = (G_i, t_i, h(B_{i-1}))
\end{align*}
\]
New blocks are broadcast to and archived by clients. Clients can
- verify that \( t_{i-1} \leq t_i \leq \text{now} \)
- verify \( h(B_{i-1}) \)
- frequently compare latest \( h(B_i) \) with other clients
to ensure consensus that
- each client sees the same serialization order of the same set of
  validated transactions
- every client receives the exact same block-chain data
- nobody can later rewrite the transaction history

The Bitcoin cryptocurrency is based on a decentralized block-chain:
- accounts identified by single-use public keys
- each transaction signed with the payer’s private key
- new blocks broadcast by “miners”, who are allowed to mint
  themselves new currency as incentive for operating the service
- issuing rate of new currency is limited by requirement for miners to
  solve cryptographic puzzle (adjust a field in each block such that
  \( h(B_i) \) has a required number of leading zeros, currently \( \approx 68 \) bits)

Password-based key derivation

Human-selected secrets (PINs, passwords, pass-phrases) usually have much
lower entropy than the > 80 bits desired for cryptographic keys.

Typical password search list: “dictionary of 64k words, 4k suffixes, 64 prefixes and 4 alteration
rules for a total of 2^38 passwords” http://phrack.sourceforge.net/tables.php

Machine-generated random strings encoded for keyboard entry (hexadecimal,
base64, etc.) still lack the full 8 bits per byte entropy of a random binary string
(e.g. only < 96 graphical characters per byte from keyboard).

Workarounds:
- Preferably generate keys with a true random bit generator.
- Ask user to enter a text string longer than the key size.
- Avoid or normalize visually similar characters: 0OQ/1lI/A4/Z2/S5/VU/nu
- Use a secure hash function to condense the passphrase to key length.
- Use a deliberately slow hash function, e.g. iterate \( C \) times.
- Use a per-user random salt value \( S \) to personalize the hash function
  against pre-computed dictionary attacks.

PBKDF2 iterates HMAC \( C \) times for each output bit.

Typical values: \( S \in \{0, 1\}^{128}, 10^3 < C < 10^7 \)

Recommendation for password-based key derivation. NIST SP 800-132, December 2010.

Key derivation functions

A secret key \( K \) should only ever be used for one single purpose, to prevent one
application of \( K \) being abused as an oracle for compromising another one.

Any cryptographic system may involve numerous applications for keys (for encryption
systems, message integrity schemes, etc.)

A key derivation function (KDF) extends a single multi-purpose key \( K \) (which may
have been manually configured by a user, or may have been the result of a
key-agreement protocol) into \( k \) single-purpose keys \( K_1, K_2, \ldots, K_k \).

Requirements:
- Use a one-way function, such that compromise of one derived key \( K_i \) does not
  also compromise the master key \( K \) or any other derived keys \( K_j (j \neq i) \).
- Use an entropy-preserving function, i.e. \( H(K_i) \approx \min{H(K)}, |K_i| \)
- Include a unique application identifier \( A \) (e.g., descriptive text string, product
  name, domain name, serial number), to minimize the risk that someone else
  accidentally uses the same derived keys for another purpose.

Secure hash functions work well for this purpose, especially those with arbitrary-length
output (e.g., SHA-3). Split their output bit sequence into the keys needed:

\[
K_1 || K_2 || \ldots || K_k = h(A, K)
\]

Hash functions with fixed output-length (e.g., SHA-256) may have to be called
multiple times, with an integer counter:

\[
K_1 || K_2 = h(A, K, (1)), K_3 || K_4 = h(A, K, (2)), \ldots
\]

ISO/IEC 11770-6

Password storage

Servers that authenticate users by password need to store some
information to verify that password.

Avoid saving a user’s password \( P \) as plaintext. Save the output of a
secure hash function \( h(P) \) instead, to help protect the passwords after
theft of the database. Verify a password by comparing it’s hash against
that in the database record.

Better: hinder dictionary attacks by adding a random salt value \( S \) and by
iterating the hash function \( C \) times to make it computationally more
expensive. The database record then stores

\[
(S, h^C(P, S))
\]
or similar.

Standard password-based key derivation functions, such as PBKDF2 or
Argon2, can also be used to verify passwords.

Argon2 is deliberately designed to be memory intensive to discourage fast ASIC implementations.
Inverting unsalted password hashes: time–memory trade-off

Target: invert \( h(p) \), where \( p \in P \) is a password from an assumed finite set \( P \) of passwords (e.g., \( h = \text{MD5} \), \(|P| = \approx 2^{53} \) 8-char ASCII strings).

Idea: define “reduction” function \( r : \{0, 1\}^{128} \rightarrow P \), then iterate \( h(r(\cdot)) \)

For example: convert input from base-2 to base-96 number, output first 8 “digits” as printable ASCII characters, interpret DEL as string terminator.

\[
m \left\{ \begin{array}{c} 0 \rightarrow x_0 \rightarrow p_1 \rightarrow h(x_1) \rightarrow p_2 \rightarrow h \cdots \rightarrow x_{n-1} \rightarrow p_n \rightarrow h(x_n) \Rightarrow L[x_n] := x_0 \\ \end{array} \right.
\]

Problem: Once \( mn \gg \sqrt{|P|} \) there are many collisions, the \( x_0 \rightarrow x_n \) chains merge, loop and overlap, covering \( P \) very inefficiently.


Inverting unsalted password hashes: “rainbow tables”

Target: invert \( h(p) \), where \( p \in P \) is a password from an assumed finite set \( P \) of passwords (e.g., \( h = \text{MD5} \), \(|P| = \approx 2^{53} \) 8-char ASCII strings).

Idea: define a “rainbow” of \( n \) reduction functions \( r_i : \{0, 1\}^{128} \rightarrow P \), then iterate \( h(r_i(\cdot)) \) to avoid loops. (For example: \( r_i(x) := r(h(x\|i)) \)).

\[
m \left\{ \begin{array}{c} 0 \rightarrow x_0 \rightarrow p_1 \rightarrow h(x_1) \rightarrow p_2 \rightarrow h \cdots \rightarrow x_{n-1} \rightarrow p_n \rightarrow h(x_n) \Rightarrow L[x_n] := x_0 \\ \end{array} \right.
\]


Other applications of secure hash functions

- deduplication – quickly identify in a large collection of files duplicates, without having to compare all pairs of files, just compare the hash of each file content.
- file identification – in a peer-to-peer filesharing network or cluster file system, identify each file by the hash of its content.
- distributed version control systems (git, mercurial, etc.) – name each revision via a hash tree of all files in that revision, along with the hash of the parent revision(s). This way, each revision name securely identifies not only the full content, but its full revision history.
Key distribution problem

In a group of $n$ participants, there are $n(n-1)/2$ pairs who might want to communicate at some point, requiring $O(n^2)$ private keys to be exchanged securely in advance.

This gets quickly unpractical if $n \gg 2$ and if participants regularly join and leave the group.

Alternative 1: introduce an intermediary “trusted third party”

Kerberos

An extension of the Needham–Schroeder protocol is now widely used in corporate computer networks between desktop computers and servers, in the form of Kerberos and Microsoft’s Active Directory. $K_{AS}$ is generated from $A$’s password (hash function).

Extensions include:

- timestamps and nonces to prevent replay attacks
- a “ticket-granting ticket” is issued and cached at the start of a session, replacing the password for a limited time, allowing the password to be instantly wiped from memory again.
- a pre-authentication step ensures that $S$ does not reply with anything encrypted under $K_{AS}$ unless the sender has demonstrated knowledge of $K_{AS}$, to hinder offline password guessing.
- mechanisms for forwarding and renewing tickets
- support for a federation of administrative domains (“realms”)

Problem: ticket message enables eavesdropper off-line dictionary attack.

Key distribution problem: other options

Alternative 2: hardware security modules + conditional access

A trusted third party generates a global key $K$ and embeds it securely in tamper-resistant hardware tokens (e.g., smartcard)

Every participant receives such a token, which also knows the identity of its owner and that of any groups they might belong to.

Each token offers its holder authenticated encryption operations $\text{Enc}_K(\cdot)$ and $\text{Dec}_K(A, \cdot)$.

Each encrypted message $\text{Enc}_K(A, M)$ contains the name of the intended recipient $A$ (or the name of a group to which $A$ belongs).

$A$’s smartcard will only decrypt messages addressed this way to $A$.

Commonly used for “broadcast encryption”, e.g., pay-TV, navigation satellites.

Alternative 3: Public-key cryptography

Find an encryption scheme where separate keys can be used for encryption and decryption.

Publish the encryption key: the “public key”

Keep the decryption key: the “secret key”

Some form of trusted third party is usually still required to certify the correctness of the published public keys, but it is no longer directly involved in establishing a secure connection.
Public-key encryption

A **public-key encryption scheme** is a tuple of PPT algorithms (Gen, Enc, Dec) such that

- the **key generation algorithm** Gen receives a security parameter $\ell$ and outputs a pair of keys $(PK, SK) \leftarrow \text{Gen}(1^\ell)$, with key lengths $|PK| \geq \ell$, $|SK| \geq \ell$;
- the **encryption algorithm** Enc maps a public key $PK$ and a plaintext message $M \in M$ to a ciphertext message $C \leftarrow \text{Enc}_{PK}(M)$;
- the **decryption algorithm** Dec maps a secret key $SK$ and a ciphertext $C$ to a plaintext message $M := \text{Dec}_{SK}(C)$, or outputs $\bot$;
- for all $\ell$, $(PK, SK) \leftarrow \text{Gen}(1^\ell)$: $\text{Dec}_{SK}(\text{Enc}_{PK}(M)) = M$.

In practice, the message space $M$ may depend on $PK$.

In some practical schemes, the condition $\text{Dec}_{SK}(\text{Enc}_{PK}(M)) = M$ may fail with negligible probability.

Security against chosen-ciphertext attacks (CCA)

Public-key encryption scheme $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$

**Experiment/game PubK^{cca}_{A,\Pi}(\ell):**

```
\begin{align*}
&\begin{array}{c}
1^{\ell} \\
\text{(PK, SK) \leftarrow \text{Gen}(1^{\ell})} \\
M_0, M_1 \\
C \leftarrow \text{Enc}_{PK}(M_0) \\
\text{adversary}
\end{array}

&\begin{array}{c}
1^{\ell} \\
1^{\ell} \\
b \in \{0, 1\} \\
M_0, M_1 \\
C \\
C_{t+1} \neq C_1, \ldots, C_{t+2} = \ldots = C_t \\
b' \\
b
\end{array}
\end{align*}
```

**Setup:**
1. The challenger generates a bit $b \in \{0, 1\}$ and a key pair $(PK, SK) \leftarrow \text{Gen}(1^\ell)$.
2. The adversary $A$ is given input $1^{\ell}$
3. The adversary $A$ outputs a pair of messages: $M_0, M_1 \in \{0, 1\}^m$.
4. The challenger computes $C \leftarrow \text{Enc}_{PK}(M_b)$ and returns $C$ to $A$
5. Finally, $A$ outputs $b'$. If $b' = b$ then $A$ has succeeded $\Rightarrow \text{PubK}^{cca}_{A,\Pi}(\ell) = 1$

Note that unlike in PrivK^{cca} we do not need to provide $A$ with any oracle access: here $A$ has access to the encryption key $PK$ and can evaluate $\text{Enc}_{PK}(\cdot)$ itself.

Security against chosen-plaintext attacks (cont’d)

**Definition:** A public-key encryption scheme $\Pi$ has indistinguishable encryptions under a chosen-plaintext attack ("is CPA-secure") if for all probabilistic, polynomial-time adversaries $A$ there exists a negligible function $\text{negl}$, such that

$$\mathbb{P}(\text{PubK}^{cpa}_{A,\Pi}(\ell) = 1) \leq \frac{1}{2} + \text{negl}(\ell)$$

**Definition:** A public-key encryption scheme $\Pi$ has indistinguishable encryptions under a chosen-ciphertext attack ("is CCA-secure") if for all probabilistic, polynomial-time adversaries $A$ there exists a negligible function $\text{negl}$, such that

$$\mathbb{P}(\text{PubK}^{cca}_{A,\Pi}(\ell) = 1) \leq \frac{1}{2} + \text{negl}(\ell)$$

**What about ciphertext integrity / authenticated encryption?**

Since the adversary has access to the public encryption key $PK$, there is no useful equivalent notion of authenticated encryption for a public-key encryption scheme.
Integer division with remainder

For every integer \( a \) and positive integer \( b \) there exist unique integers \( q \) and \( r \) with \( a = qb + r \) and \( 0 \leq r < b \).

The modulo operator performs integer division and outputs the remainder:

\[
a \mod b = r \implies 0 \leq r < b \land \exists q \in \mathbb{Z} : a - qb = r
\]

Examples: \( 7 \mod 5 = 2, \) \( -1 \mod 10 = 9 \)

If

\[
a \mod n = b \mod n
\]

we say that "\( a \) and \( b \) are congruent modulo \( n \)", and also write

\[
a \equiv b \pmod{n}
\]

This implies \( n \mid (a - b) \). Being congruent modulo \( n \) is an equivalence relationship:

- reflexive: \( a \equiv a \pmod{n} \)
- symmetric: \( a \equiv b \pmod{n} \implies b \equiv a \pmod{n} \)
- transitive: \( a \equiv b \pmod{n} \land b \equiv c \pmod{n} \implies a \equiv c \pmod{n} \)

Number theory: integers, divisibility, primes, \( \text{gcd} \)

Set of integers: \( \mathbb{Z} := \{ \ldots, -2, -1, 0, 1, 2, \ldots \} \quad a, b \in \mathbb{Z} \)

If there exists \( c \in \mathbb{Z} \) such that \( ac = b \), we say "\( a \) divides \( b \)" or "\( a \mid b \)."

- if \( 0 < a \) then \( a \) is a "divisor" of \( b \)
- if \( 1 < a < b \) then \( a \) is a "factor" of \( b \)
- if \( a \) does not divide \( b \), we write "\( a \nmid b \)"

If integer \( p > 1 \) has no factors (only 1 and \( p \) as divisors), it is "prime", otherwise it is "composite". Primes: \( 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, \ldots \)

- every integer \( n > 1 \) has a unique prime factorization \( n = \prod p_i^{e_i} \), with primes \( p_i \) and positive integers \( e_i \)

The greatest common divisor \( \text{gcd}(a, b) \) is the largest \( c \) with \( c \mid a \) and \( c \mid b \).

- examples: \( \text{gcd}(18, 12) = 6, \) \( \text{gcd}(15, 9) = 3, \) \( \text{gcd}(15, 8) = 1 \)
- if \( \text{gcd}(a, b) = 1 \) we say \( a \) and \( b \) are "relatively prime"
- \( \text{gcd}(a, b) = \text{gcd}(b, a) \)
- if \( c \mid ab \) and \( \text{gcd}(a, c) = 1 \) then \( c \mid b \)
- if \( a \mid n \) and \( b \mid n \) and \( \text{gcd}(a, b) = 1 \) then \( ab \mid n \)

Modular arithmetic

Addition, subtraction, and multiplication work the same under congruence modulo \( n \):

If \( a \equiv a' \pmod{n} \) and \( b \equiv b' \pmod{n} \) then

\[
a + b \equiv a' + b' \pmod{n} \\
-a - b \equiv a' - b' \pmod{n} \\
ab \equiv a'b' \pmod{n}
\]

Associative, commutative and distributive laws also work the same:

\[
a(b + c) \equiv ab + ac \equiv ca + ba \pmod{n}
\]

When evaluating an expression that is reduced modulo \( n \) in the end, we can also reduce any intermediate results. Example:

\[
(a - bc) \mod n = (((a \mod n) - (b \mod n)(c \mod n)) \mod n) \mod n
\]

Reduction modulo \( n \) limits intermediate values to

\[
\mathbb{Z}_n := \{0, 1, 2, \ldots, n - 1\},
\]

the "set of integers modulo \( n \)."

Staying within \( \mathbb{Z}_n \) helps to limit register sizes and can speed up computation.
Groups

A group \((G, \cdot)\) is a set \(G\) and an operator \(\cdot : G \times G \to G\) that have

- **Closure**: for all \(a, b \in G\), \(a \cdot b \in G\).
- **Associativity**: for all \(a, b, c \in G\), \((a \cdot b) \cdot c = a \cdot (b \cdot c)\).
- **Neutral Element**: there exists a \(e \in G\) such that for all \(a \in G\), \(a \cdot e = e \cdot a = a\).
- **Inverse Element**: for each \(a \in G\) there exists some \(b \in G\) such that \(a \cdot b = b \cdot a = e\).

If \(a \cdot b = b \cdot a\) for all \(a, b \in G\), the group is called commutative (or abelian).

Examples of abelian groups:
- \((\mathbb{Z}, +)\), \((\mathbb{R}, +)\), \((\mathbb{R} \setminus \{0\}, \cdot)\)
- \((\mathbb{Z}_n, +)\) - set of integers modulo \(n\) with addition \(a + b := (a + b) \mod n\)
- \((\{0, 1\}^*, \oplus)\) where \(a_1 a_2 \ldots a_n \oplus b_1 b_2 \ldots b_n = c_1 c_2 \ldots c_n\) with \((a_i + b_i) \mod 2 = c_i\) (for all \(1 \leq i \leq n\), \(a_i, b_i, c_i \in \{0, 1\}\)) “bit-wise XOR”

If there is no inverse element for each element, \((G, \cdot)\) is a monoid instead.

Examples of monoids:
- \((\mathbb{Z}, \cdot)\) - set of integers under multiplication
- \((\{0, 1\}^*, \|\|)\) - set of variable-length bit strings under concatenation

Euclid’s algorithm

\[
gcd(21, 15) = gcd(15, 21 \mod 15) = gcd(15, 6) = gcd(6, 15 \mod 6) = gcd(6, 3) = -2 \times 21 + 3 \times 15
\]

Euclid’s algorithm

Euclidean algorithm: \((\text{WLOG } a \geq b > 0, \text{ since } gcd(a, b) = gcd(b, a))\)

\[
gcd(a, b) = \begin{cases} b, & \text{if } b \mid a \\ gcd(b, a \mod b), & \text{otherwise} \end{cases}
\]

For all positive integers \(a, b\), there exist integers \(x, y\) such that \(gcd(a, b) = ax + by\).

Euclid’s extended algorithm also provides \(x, y\): \((\text{WLOG } a \geq b > 0)\)

\[
\begin{align*}
egcd(a, b, x, y) := \\
& \begin{cases} 
(b, 0, 1), & \text{if } b \mid a \\
(d, y, x - yq), & \text{otherwise}, \\
\quad \text{with } (d, x, y) := egcd(b, r), \\
\quad \text{where } a = qb + r, 0 \leq r < b
\end{cases}
\end{align*}
\]

Permutations and groups

**Permutation groups**

A set \(P\) of permutations over a finite set \(S\) forms a group under concatenation if

- **Closure**: for any pair of permutations \(g, h : S \leftrightarrow S\) in \(P\) their concatenation \(g \circ h : x \mapsto g(h(x))\) is also in \(P\).
- **Neutral element**: the identity function \(x \mapsto x\) is in \(P\).
- **Inverse element**: for each permutation \(g \in P\), the inverse permutation \(g^{-1}\) is also in \(P\).

Note that function composition is associative: \(f \circ (g \circ h) = (f \circ g) \circ h\)

The set of all permutations of a set \(S\) forms a permutation group called the “symmetric group” on \(S\). Non-trivial symmetric groups \(|S| > 1\) are not abelian.

**Each group is isomorphic to a permutation group**

Given a group \((G, \cdot)\), map each \(g \in G\) to a function \(f_g : x \mapsto x \cdot g\). Since \(g^{-1} \in G\), \(f_g\) is a permutation, and the set of all \(f_g\) for \(g \in G\) forms a permutation group isomorphic to \(G\). (“Cayley’s theorem”)

**Encryption schemes are permutations.**

Which groups can be used to form encryption schemes?
Subgroups

\((\mathbb{H}, \bullet)\) is a subgroup of \((\mathbb{G}, \bullet)\) if

- \(\mathbb{H}\) is a subset of \(\mathbb{G}\) (\(\mathbb{H} \subseteq \mathbb{G}\))
- the operator \(\bullet\) on \(\mathbb{H}\) is the same as on \(\mathbb{G}\)
- \((\mathbb{H}, \bullet)\) is a group, that is
  - for all \(a, b \in \mathbb{H}\) we have \(a \bullet b \in \mathbb{H}\)
  - each element of \(\mathbb{H}\) has an inverse element in \(\mathbb{H}\)
  - the neutral element of \((\mathbb{G}, \bullet)\) is also in \(\mathbb{H}\).

Examples of subgroups

- \((n\mathbb{Z}, +)\) with \(n\mathbb{Z} := \{ni | i \in \mathbb{Z}\} = \{\ldots, -2n, -n, 0, n, 2n, \ldots\}\) – the set of integer multiples of \(n\) is a subgroup of \((\mathbb{Z}, +)\)
- \((\mathbb{R}^+, \cdot)\) – the set of positive real numbers is a subgroup of \((\mathbb{R} \setminus \{0\}, \cdot)\)
- \((\mathbb{Q}, +)\) is a subgroup of \((\mathbb{R}, +)\), which is a subgroup of \((\mathbb{C}, +)\)
- \((\mathbb{Q} \setminus \{0\}, \cdot)\) is a subgroup of \((\mathbb{R} \setminus \{0\}, \cdot)\), etc.
- \((\{0, 2, 4, 6\}, +)\) is a subgroup of \((\mathbb{Z}_8, +)\)

Rings

A ring \((\mathbb{R}, \boxplus, \boxtimes)\) is a set \(\mathbb{R}\) and two operators \(\boxplus : \mathbb{R} \times \mathbb{R} \to \mathbb{R}\) and \(\boxtimes : \mathbb{R} \times \mathbb{R} \to \mathbb{R}\) such that

- \((\mathbb{R}, \boxplus)\) is an abelian group
- \((\mathbb{R}, \boxtimes)\) is a monoid
- \(a \boxtimes (b \boxtimes c) = (a \boxtimes b) \boxtimes (a \boxtimes c)\) and \((a \boxplus b) \boxtimes c = (a \boxtimes b) \boxtimes c\) \(\text{(distributive law)}\)

If also \(a \boxtimes b = b \boxtimes a\), then we have a commutative ring.

Examples for rings:

- \((\mathbb{Z}[x], +, \cdot)\), where
  \[
  \mathbb{Z}[x] := \left\{ \sum_{i=0}^{n} a_i x^i \big| a_i \in \mathbb{Z}, n \geq 0 \right\}
  \]
  is the set of polynomials with variable \(x\) and coefficients from \(\mathbb{Z}\)
  - commutative
- \(\mathbb{Z}_n[x]\) – the set of polynomials with coefficients from \(\mathbb{Z}_n\)
- \((\mathbb{R}^{n \times n}, +, \cdot)\) – \(n \times n\) matrices over \(\mathbb{R}\) – not commutative

Notations used with groups

When the definition of the group operator is clear from the context, it is often customary to use the symbols of the normal arithmetic addition or multiplication operators ("+", "\times", ",", "\cdot") for the group operation.

There are two commonly used alternative notations:

- "Additive" group: think of group operator as a kind of "\(+\)"
  - write 0 for the neutral element and \(-g\) for the inverse of \(g \in \mathbb{G}\).
  - write \(g \cdot i := g \cdot g \cdot \ldots \cdot g\) \((g \in \mathbb{G}, i \in \mathbb{Z})\) \(i\) times

- "Multiplicative" group: think of group operator as a kind of "\(\times\)"
  - write 1 for the neutral element and \(g^{-1}\) for the inverse of \(g \in \mathbb{G}\).
  - write \(g^i := g \cdot g \cdot \ldots \cdot g\) \((g \in \mathbb{G}, i \in \mathbb{Z})\) \(i\) times

Fields

A field \((\mathbb{F}, \boxplus, \boxtimes)\) is a set \(\mathbb{F}\) and two operators \(\boxplus : \mathbb{F} \times \mathbb{F} \to \mathbb{F}\) and \(\boxtimes : \mathbb{F} \times \mathbb{F} \to \mathbb{F}\) such that

- \((\mathbb{F}, \boxplus)\) is an abelian group with neutral element \(0\_\mathbb{F}\)
- \((\mathbb{F} \setminus \{0\_\mathbb{F}\}, \boxtimes)\) is also an abelian group with neutral element \(1\_\mathbb{F} \neq 0\_\mathbb{F}\)
- \(a \boxplus (b \boxtimes c) = (a \boxplus b) \boxtimes (a \boxtimes c)\) and \((a \boxplus b) \boxtimes c = (a \boxtimes b) \boxtimes c\) \(\text{(distributive law)}\)

In other words: a field is a commutative ring where each element except for the neutral element of the addition has a multiplicative inverse.

Field means: division works, linear algebra works, solving equations, etc.

Examples for fields:

- \((\mathbb{Q}, +, \cdot)\)
- \((\mathbb{R}, +, \cdot)\)
- \((\mathbb{C}, +, \cdot)\)
Finite fields (Galois fields)

Ring \( \mathbb{Z}_n \)

Set of integers modulo \( n \) is \( \mathbb{Z}_n := \{0, 1, \ldots, n - 1\} \)

When we refer to \((\mathbb{Z}_n, +)\) or \((\mathbb{Z}_n, \cdot)\), we apply after each addition or multiplication a reduction modulo \( n \). (No need to write out "mod \( n \)" each time.)

We add/subtract the integer multiple of \( n \) needed to get the result back into \( \mathbb{Z}_n \).

\( (\mathbb{Z}_n, +) \) is an abelian group:
- neutral element of addition is 0
- the inverse element of \( a \in \mathbb{Z}_n \) is \( n - a \equiv -a \) (mod \( n \))

\( (\mathbb{Z}_n, \cdot) \) is a monoid:
- neutral element of multiplication is 1
- the inverse element of \( a \in \mathbb{Z}_n \) is \( n, a \) (with \( n \) prime), or as \( \mathbb{F}_p \).

From now on, when we refer to \( \mathbb{Z}_n \), we usually imply that we work with the commutative ring \((\mathbb{Z}_n, +, \cdot)\).

Examples in \( \mathbb{Z}_8 \): \( 4 + 3 = 2, 4 \cdot 2 = 3, 4^2 = 1 \)

Finite fields (Galois fields)

\((\mathbb{Z}_p, +, \cdot)\) is a finite field with \( p \) elements, where \( p \) is a prime number.

Also written as \( \text{GF}(p) \), the "Galois field of order \( p \)" or as \( \mathbb{F}_p \).

We can also construct finite fields \( \text{GF}(p^n) \) with \( p^n \) elements:

- **Elements**: polynomials over variable \( x \) with degree less than \( n \) and coefficients from the finite field \( \mathbb{Z}_p \)
- **Modulus**: select an irreducible polynomial \( T(x) \in \mathbb{Z}_p[x] \) of degree \( n \)
  \[ T(x) = c_0 x^n + \cdots + c_2 x^2 + c_1 x + c_0 \]
  where \( c_i \in \mathbb{Z}_p \) for all \( 0 \leq i \leq n \). An irreducible polynomial cannot be factored into two other polynomials from \( \mathbb{Z}_p[x] \setminus \{0, 1\} \).
- **Addition**: \( \oplus \) is normal polynomial addition (i.e., pairwise addition of the coefficients in \( \mathbb{Z}_p \))
- **Multiplication**: \( \otimes \) is normal polynomial multiplication, then divide by \( T(x) \) and take the remainder (i.e., multiplication modulo \( T(x) \)).

**Theorem**: any finite field has \( p^n \) elements \( (p \text{ prime}, \ n > 0) \)

**Theorem**: all finite fields of the same size are isomorphic

Division in \( \mathbb{Z}_n \)

In ring \( \mathbb{Z}_n \), element \( a \) has a multiplicative inverse \( a^{-1} \) (with \( aa^{-1} = 1 \)) if and only if \( \gcd(n, a) = 1 \).

In this case, the extended Euclidian algorithm gives us

\[ nx + ay = 1 \]

and since \( nx = 0 \) in \( \mathbb{Z}_n \) for all \( x \), we have \( ay = 1 \).

Therefore \( y = a^{-1} \) is the inverse needed for dividing by \( a \).

- We call the set of all elements in \( \mathbb{Z}_n \) that have a multiplicative inverse the "multiplicative group" of \( \mathbb{Z}_n \):
  \[ \mathbb{Z}_n^* := \{ a \in \mathbb{Z}_n \mid \gcd(n, a) = 1 \} \]

- If \( p \) is prime, then \( (\mathbb{Z}_p^*, \cdot) \) with
  \[ \mathbb{Z}_p^* = \{1, \ldots, p - 1\} \]
  is a group, and \( (\mathbb{Z}_p, +, \cdot) \) is a (finite) field, that is every element except 0 has a multiplicative inverse.

**Example**: Multiplicative inverses of \( \mathbb{Z}_7 \):

\[ 1 \cdot 1 = 1, 2 \cdot 4 = 1, 3 \cdot 5 = 1, 4 \cdot 2 = 1, 5 \cdot 3 = 1, 6 \cdot 6 = 1 \]

GF(2\(^n\))

GF(2) is particularly easy to implement in hardware:

- additive = subtraction = XOR gate
- multiplication = AND gate
- division can only be by 1, which merely results in the first operand

Of particular practical interest in modern cryptography are larger finite fields of the form GF(2\(^n\)) (also written as \( \mathbb{F}_{2^n} \)):

- Polynomials are represented as bit words, each coefficient = 1 bit.
- Addition/subtraction is implemented via bit-wise XOR instruction.
- Multiplication and division of binary polynomials is like binary integer multiplication and division, but *without carry-over bits*. This allows the circuit to be clocked much faster.

Recent Intel/AMD CPUs have added instruction PCLMULQDQ for 64 × 64-bit carry-less multiplication. This helps to implement arithmetic in GF(2\(^{64}\)) or GF(2\(^{128}\)) more efficiently.
GF($2^8$) example

The finite field GF($2^8$) consists of the 256 polynomials of the form

$$c_7x^7 + \cdots + c_2x^2 + c_1x + c_0 \quad c_i \in \{0, 1\}$$

each of which can be represented by the byte $c_7c_6c_5c_4c_3c_2c_1c_0$.

As modulus we chose the irreducible polynomial

$$T(x) = x^8 + x^4 + x^3 + x + 1 \quad \text{or} \quad 1001011011$$

Example operations:

\begin{enumerate}
  \item $(x^2 + x^2 + x + 1) \oplus (x^7 + x^6 + 1) = x^6 + x^5 + x$
    
    or equivalently 10100011 01100001 = 01100010

  \item $(x^6 + x^4 + 1) \otimes_T (x^2 + 1) = [(x^6 + x^4 + 1)(x^2 + 1)] \bmod T(x) = (x^8 + x^4 + x^2 + x + 1) \bmod (x^8 + x^4 + x^3 + x + 1) = (x^8 + x^4 + x^2 + 1) \oplus (x^7 + x^4 + x^3 + x + 1) = x^3 + x^2 + x$
    
    or equivalently 01010001 \otimes_T 00001010 = 10001010 \oplus 1000101101 = 00001110
\end{enumerate}

Proofs

\begin{enumerate}
  \item In any group $(G, \cdot)$ with $a, b, c \in G$ we have $ac = bc \Rightarrow a = b$.
    
    Proof: $ac = bc \Rightarrow (ac)c^{-1} = (bc)c^{-1} \Rightarrow a(cc^{-1}) = b(cc^{-1}) \Rightarrow a \cdot 1 = b \cdot 1 \Rightarrow a = b$.

  \item Let $G$ be an abelian group of order $m$ with elements $g_1, \ldots, g_m$. We have
    
    $$g_1 \cdot g_2 \cdots g_m = (gg_1) \cdot (gg_2) \cdots (gg_m)$$
    
    for arbitrary fixed $g \in G$, because $gg_1 = gg_2, \Rightarrow g_1 = g_2$ (see 1), which implies that each of the $(gg_1)$ is distinct, and since there are only $m$ elements of $G$, the right-hand side of the above equation is just a permutation of the left-hand side. Now pull out the $g$:
    
    $$g_1 \cdot g_2 \cdots g_m = (gg_1) \cdot (gg_2) \cdots (gg_m) = g^m \cdot g_1 \cdot g_2 \cdots g_m \Rightarrow g^m = 1.$$
    
    (Not shown here: $g^m = 1$ also holds for non-commutative groups.)
    
    Also: $g^m = 1 \Rightarrow g^e = g^e \cdot (g^m)^n = g^{e-nm} = g^{e \bmod m}$ for any $n \in \mathbb{Z}$.

  \item Likewise: $i = \text{ord}(g) \Rightarrow g^i = 1 \Rightarrow g^i = g^i \cdot (g^i)^n = g^{i+n} = g^{i \bmod i}$ for any $n \in \mathbb{Z}$.

  \item Let $i = \text{ord}(g)$.
    
    "If $z \equiv y \pmod{i}$ then $x \equiv y \pmod{i}$ means $x = y + ki$ for some $k \in \mathbb{Z}$. Since $(x-y) \bmod i < i$, but $i$ is the smallest positive integer with $g^i = 1$, we must have $(x-y) \bmod i = 0 \Rightarrow x \equiv y \pmod{i}$.

  \item $g^m = 1 \Rightarrow g^0 \therefore m \equiv 0 \pmod{\text{ord}(g)}$ from 3 and so $\text{ord}(g)|m$.

  \item $(g^i)^n = g^{in} \text{ mod } m = g^i = g$ means that $g \mapsto g^i$ is indeed the inverse of $g \mapsto g^d$ if $ed \bmod m = 1$.
\end{enumerate}

Katz/Lindell, sections 8.1 and 8.3

Finite groups

Let $(G, \cdot)$ be a group with a finite number of elements $|G|$.

Practical examples here: $(\mathbb{Z}_n, +)$, $(\mathbb{Z}_n^*, \cdot)$, (GF($2^n$), $\oplus$), (GF($2^n$) \setminus \{0\}, $\otimes$)

Terminology:

\begin{itemize}
  \item The order of a group $G$ is its size $|G|$
  \item order of group element $g$ in $G$ is
    
    $$\text{ord}_G(g) = \min \{ i > 0 \mid g^i = 1 \}.$$ 

  \end{itemize}

Useful facts regarding any element $g \in G$ in a group of order $m = |G|$:

\begin{enumerate}
  \item $g^m = 1, g^e = g^{e \bmod m}$
  \item $g^y = g^x \Leftrightarrow x \equiv y \pmod{\text{ord}(g)}$
  \item $g^y = g^x \Rightarrow x \equiv y \pmod{\text{ord}(g)}$

  \item $\text{ord}(g) \mid m$ “Lagrange’s theorem”
  \item if $\gcd(e, m) = 1$ then $g \mapsto g^e$ is a permutation, and $g \mapsto g^d$ its inverse (i.e., $g^{ed} = g$ if $ed \bmod m = 1$
\end{enumerate}

Cyclic groups

Let $G$ be a finite (multiplicative) group of order $m = |G|$.

For $g \in G$ consider the set

$$\langle g \rangle := \{ g^0, g^1, g^2, \ldots \}$$

Note that $|\langle g \rangle| = \text{ord}(g)$ and $\langle g \rangle = \{ g^0, g^1, g^2, \ldots, g^{\text{ord}(g)-1} \}$.

Definitions:

\begin{itemize}
  \item We call $g$ a generator of $G$ if $\langle g \rangle = G$.
  \item We call $G$ cyclic if it has a generator.
\end{itemize}

Useful facts:

\begin{enumerate}
  \item Every cyclic group of order $m$ is isomorphic to $(\mathbb{Z}_m, +)$.
  \item $(g^i)$ is a subgroup of $G$ (subset, a group under the same operator)
  \item If $|G|$ is prime, then $G$ is cyclic and all $g \in G \setminus \{1\}$ are generators.
\end{enumerate}

Katz/Lindell, section 8.3
How to find a generator?

Let $\mathbb{G}$ be a cyclic (multiplicative) group of order $m = |\mathbb{G}|$.

- If $m$ is prime, any non-neutral element is a generator. Done.
  But $|\mathbb{Z}_p^*| = p - 1$ is not prime (for $p > 3$).
- Directly testing for $|\langle g \rangle| \not= m$ is infeasible for crypto-sized $m$.
- Fast test: if $m = \prod p_i^{e_i}$ is composite, then $g \in \mathbb{G}$ is a generator if and only if $g^{m/p_i} \not= 1$ for all $i$.
- Sampling a polynomial number of elements of $\mathbb{G}$ for the above test will lead to a generator in polynomial time (of $\log_2 m$) with all but negligible probability.

$\Rightarrow$ Make sure you pick a group of an order with known prime factors.

One possibility for $\mathbb{Z}_p^*$ (commonly used):

- Chose a “strong prime” $p = 2q + 1$, where $q$ is also prime
  $\Rightarrow |\mathbb{Z}_p^*| = p - 1 = 2q$ has prime factors 2 and $q$.

$(\mathbb{Z}_p^*, \cdot)$ is a cyclic group

For every prime $p$ there exists a generator $g \in \mathbb{Z}_p^*$ such that

$$\mathbb{Z}_p^* = \{g^i \mod p \mid 0 \leq i \leq p - 2\}$$

Note that this does not follow from fact 3 on slide 168: $\mathbb{Z}_p^*$ is of order $p - 1$, which is even (for $p > 3$); not prime.

Example in $\mathbb{Z}_7$:

- $(1^0, 1^1, 1^2, 1^3, 1^4, 1^5) = (1, 1, 1, 1, 1)$
- $(2^0, 2^1, 2^2, 2^3, 2^4, 2^5) = (1, 2, 4, 1, 2, 4)$
- $(3^0, 3^1, 3^2, 3^3, 3^4, 3^5) = (1, 3, 2, 6, 4, 5)$
- $(4^0, 4^1, 4^2, 4^3, 4^4, 4^5) = (1, 4, 2, 1, 4, 2)$
- $(5^0, 5^1, 5^2, 5^3, 5^4, 5^5) = (1, 5, 4, 6, 2, 3)$
- $(6^0, 6^1, 6^2, 6^3, 6^4, 6^5) = (1, 6, 1, 6, 1, 6)$

- $3$ and $5$ are generators of $\mathbb{Z}_7^*$
- $1, 2, 4, 6$ generate subgroups of $\mathbb{Z}_7$: $\{1\}, \{1, 2, 4\}, \{1, 2, 4\}, \{1, 6\}$
- $\text{ord}(1) = 1, \text{ord}(2) = 3, \text{ord}(3) = 6, \text{ord}(4) = 3, \text{ord}(5) = 6, \text{ord}(6) = 2$

For every prime $p$ every element $g \in \mathbb{Z}_p \setminus \{0\}$ is a generator:

$$\mathbb{Z}_p = \langle g \rangle = \{g \cdot i \mod p \mid 0 \leq i \leq p - 1\}$$

Note that this follows from fact 3 on slide 168: $\mathbb{Z}_p$ is of order $p$, which is prime.

Example in $\mathbb{Z}_7$:

- $(1 \cdot 0, 1 \cdot 1, 1 \cdot 2, 1 \cdot 3, 1 \cdot 4, 1 \cdot 5, 1 \cdot 6) = (0, 1, 2, 3, 4, 5, 6)$
- $(2 \cdot 0, 2 \cdot 1, 2 \cdot 2, 2 \cdot 3, 2 \cdot 4, 2 \cdot 5, 2 \cdot 6) = (0, 2, 4, 6, 1, 3, 5)$
- $(3 \cdot 0, 3 \cdot 1, 3 \cdot 2, 3 \cdot 3, 3 \cdot 4, 3 \cdot 5, 3 \cdot 6) = (0, 3, 6, 2, 5, 1, 4)$
- $(4 \cdot 0, 4 \cdot 1, 4 \cdot 2, 4 \cdot 3, 4 \cdot 4, 4 \cdot 5, 4 \cdot 6) = (0, 4, 1, 5, 2, 6, 3)$
- $(5 \cdot 0, 5 \cdot 1, 5 \cdot 2, 5 \cdot 3, 5 \cdot 4, 5 \cdot 5, 5 \cdot 6) = (0, 5, 3, 1, 6, 4, 2)$
- $(6 \cdot 0, 6 \cdot 1, 6 \cdot 2, 6 \cdot 3, 6 \cdot 4, 6 \cdot 5, 6 \cdot 6) = (0, 6, 5, 4, 3, 2, 1)$

- All the non-zero elements of $\mathbb{Z}_7$ are generators
- $\text{ord}(0) = 1, \text{ord}(1) = \text{ord}(2) = \text{ord}(3) = \text{ord}(4) = \text{ord}(5) = \text{ord}(6) = 7$

Fermat’s and Euler’s theorem

**Fermat’s little theorem: (1640)**

$$p \text{ prime and } \gcd(a, p) = 1 \Rightarrow a^{p-1} \mod p = 1$$

Recall from Lagrange’s theorem: for $a \in \mathbb{Z}_p^*$, $\text{ord}(a)(p - 1)$ since $|\mathbb{Z}_p^*| = p - 1$.

**Euler’s phi function:**

$$\varphi(n) = |\mathbb{Z}_n^*| = |\{a \in \mathbb{Z}_n \mid \gcd(n, a) = 1\}|$$

- Example: $\varphi(12) = |\{1, 5, 7, 11\}| = 4$
- primes $p, q$:
  $$\varphi(p) = p - 1$$
  $$\varphi(p^k) = p^{k-1}(p - 1)$$
  $$\varphi(pq) = (p - 1)(q - 1)$$

- $\gcd(a, b) = 1 \Rightarrow \varphi(ab) = \varphi(a)\varphi(b)$

**Euler’s theorem: (1763)**

$$\gcd(a, n) = 1 \iff a^{\varphi(n)} \mod n = 1$$

Recall from Lagrange’s theorem: for $a \in \mathbb{Z}_n^*$, $\gcd(a, n) = 1$ since $|\mathbb{Z}_n^*| = \varphi(n)$.
Chinese remainder theorem

**Definition:** Let \((\mathbb{G}, \bullet)\) and \((\mathbb{H}, \circ)\) be two groups. A function \(f : \mathbb{G} \to \mathbb{H}\) is an isomorphism from \(\mathbb{G}\) to \(\mathbb{H}\) if

- \(f\) is a 1-to-1 mapping (bijection)
- \(f(g_1 \bullet g_2) = f(g_1) \circ f(g_2)\) for all \(g_1, g_2 \in \mathbb{G}\)

**Chinese remainder theorem:**

For any \(p, q\) with \(\gcd(p, q) = 1\) and \(n = pq\), the mapping

\[
f : \mathbb{Z}_n \leftrightarrow \mathbb{Z}_p \times \mathbb{Z}_q, \quad f(x) = (x \mod p, x \mod q)
\]

is an isomorphism, both from \(\mathbb{Z}_n\) to \(\mathbb{Z}_p \times \mathbb{Z}_q\) and from \(\mathbb{Z}_n^*\) to \(\mathbb{Z}_p^* \times \mathbb{Z}_q^*\).

**Inverse:** To get back from \(x_p, x_q\) to \(x\), we use the extended algorithm to find \(a, b\) such that \(ap + bq = 1\), and then \(x = (x_p, x_qap) \mod n\).

**Application:** arithmetic operations on \(\mathbb{Z}_n\) can instead be done on both \(\mathbb{Z}_p\) and \(\mathbb{Z}_q\) after this mapping, which may be faster.

**Example:** \(n = pq = 3 \times 5 = 15\)

<table>
<thead>
<tr>
<th>(x \mod 3)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x \mod 5)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Taking roots in \(\mathbb{Z}_p^*\)

If \(x^e = c\) in \(\mathbb{Z}_p\), then \(x\) is the “\(e^{th}\) root of \(c\), or \(x = c^{1/e}\).

**Case 1:** \(\gcd(e, p-1) = 1\)

Find \(d\) with \(de = 1\) in \(\mathbb{Z}_{p-1}\) (Euclid’s extended), then \(c^{1/e} = c^d\) in \(\mathbb{Z}_p^*\).

**Proof:** \((c^e)^d = c^{de} = c^{de \mod (p-1)} = c^1 = c\).

**Case 2:** \(e = 2\) (taking square roots)

\(\gcd(2, p-1) \neq 1\) if \(p\) odd prime ⇒ Euclid’s extended alg. no help here.

- If \(p\) mod 4 = 3 and \(c \in \mathbb{Z}_p^*\) is a quadratic residue: \(\sqrt{c} = c^{(p+1)/4}\)

**Proof:** \((x^{(p-1)/2})^2 = c^{(p-1)/2} = (c^{p+1}/4)^2 = c\).

- If \(p\) mod 4 = 1 this can also be done efficiently (details omitted).

**Application:** solve quadratic equations \(ax^2 + bx + c = 0\) in \(\mathbb{Z}_p\)

**Solution:** \(x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}\)

**Algorithms:** \(\sqrt{b^2 - 4ac}\) as above, \((2a)^{-1}\) using Euclid’s extended

Quadratic residues in \((\mathbb{Z}_p^*, \cdot)\)

In \(\mathbb{Z}_p^*\), the squaring of an element, \(x \mapsto x^2\) is a 2-to-1 function:

\[y = x^2 = (-x)^2\]

**Example in \(\mathbb{Z}_7^*:\)

\((1^2, 2^2, 3^2, 4^2, 5^2, 6^2) = (1, 4, 2, 2, 4, 1)\)

If \(y\) is the square of a number in \(\mathbb{Z}_p\), then \(y\) is a “quadratic residue”.

**Example:** \(\mathbb{Z}_7^*\) has 3 quadratic residues: \(\{1, 2, 4\}\).

If \(p\) is an odd prime: \(\mathbb{Z}_p^*\) has \((p-1)/2\) quadratic residues.

\(\mathbb{Z}_p^*\) would have one more: 0

**Euler’s criterion:**

\[c^{(p-1)/2} \mod p = 1 \iff c\text{ is a quadratic residue in }\mathbb{Z}_p^*\]

**Example in \(\mathbb{Z}_7^*: (7-1)/2 = 3,\) \(\{1^3, 2^3, 3^3, 4^3, 5^3, 6^3\} = (1, 1, 6, 1, 6, 6)\)

\[c^{(p-1)/2}\] is also called the Legendre symbol

Working in subgroups of \(\mathbb{Z}_p^*\)

How can we construct a cyclic finite group \(\mathbb{G}\) where all non-neutral elements are generators?

Recall that \(\mathbb{Z}_p^*\) has \(q = (p-1)/2\) quadratic residues, exactly half of its elements.

**Quadratic residue:** an element that is the square of some other element.

Choose \(p\) to be a strong prime, that is where \(q\) is also prime.

**Application:** solve quadratic equations \(ax^2 + bx + c = 0\) in \(\mathbb{Z}_p\)

**Solution:** \(x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}\)

**Algorithms:** \(\sqrt{b^2 - 4ac}\) as above, \((2a)^{-1}\) using Euclid’s extended

**Chinese Remainder Theorem:** If \(n\) is composite, then we know how to test whether \(c^{1/e}\) exists, and how to compute it efficiently, only if we know the prime factors of \(n\). Basic idea: apply Chinese Remainder Theorem, then apply above techniques for \(\mathbb{Z}_p^*\).
Modular exponentiation

In cyclic group \((G, \cdot)\) (e.g., \(G = \mathbb{Z}_q^*)\):

How do we calculate \(g^e\) efficiently? (\(g \in G, e \in \mathbb{N}\))

**Naive algorithm:** \(g^e = g \cdot g \cdot \ldots \cdot g\)

Far too slow for crypto-size \(e\) (e.g., \(e \approx 2^{256}\))!

**Square and multiply algorithm:**

Binary representation: \(e = \sum_{i=0}^{n} e_i \cdot 2^i\), \(n = \lfloor \log_2 e \rfloor\), \(e_i = \lfloor \frac{e}{2^i} \rfloor \mod 2\)

Computation:

\[
\begin{align*}
g^0 &:= g, \\
g^{2^i} &:= \left( g^{2^{i-1}} \right)^2 \\
g^e &:= \prod_{i=0}^{n} \left( g^{2^i} \right)^{e_i} 
\end{align*}
\]

**Side-channel vulnerability:** the if statement leaks the binary representation of \(e\). “Montgomery’s ladder” is an alternative algorithm with fixed control flow.

**Discrete logarithm problem**

Let \((G, \cdot)\) be a given cyclic group of order \(q = |G|\) with given generator \(g\) \((G = \{g^0, g^1, \ldots, g^{q-1}\})\). The “discrete logarithm problem (DLP)” is finding for a given \(y \in G\) the number \(x \in \mathbb{Z}_q\) such that

\[
g^x = g \cdot g \cdot \ldots \cdot g = y
\]

If \((G, \cdot)\) is clear from context, we can write \(x = \log_g y\). For any \(x'\) with \(g^{x'} = y\), we have \(x = x' \mod q\). Discrete logarithms behave similar to normal logarithms: \(\log_g 1 = 0\) (if 1 is the neutral element of \(G\)), \(\log_g h^r = (r \cdot \log_g h) \mod q\), and \(\log_g h_1 h_2 = (\log_g h_1 + \log_g h_2) \mod q\).

For cryptographic applications, we require groups with

- a probabilistic polynomial-time group-generation algorithm \(G(1^l)\) that outputs a description of \(G\) with \(|\log_G |G|\| = \ell\);
- a description that defines how each element of \(G\) is represented uniquely as a bit pattern;
- efficient (polynomial time) algorithms for \(\cdot\), for picking an element of \(G\) uniformly at random, and for testing whether a bit pattern represents an element of \(G\);
- Hard discrete logarithm problems

The discrete logarithm experiment \(DLog_{G,A}(\ell)\):

1. Run \(G(1^l)\) to obtain \((G, q, g)\), where \(G\) is a cyclic group of order \(q\) \((2^{l-1} < q \leq 2^{\ell})\) and \(g\) is a generator of \(G\)
2. Choose uniform \(h \in G\).
3. Give \((G, q, g, h)\) to \(A\), which outputs \(x \in \mathbb{Z}_q\)
4. Return 1 if \(g^x = h\), otherwise return 0

We say “the discrete-logarithm problem is hard relative to \(G\)” if for all probabilistic polynomial-time algorithms \(A\) there exists a negligible function \(\text{negl}\), such that \(P(DLog_{G,A}(\ell) = 1) \leq \text{negl}(\ell)\).
Diffie–Hellman problems

Let \((G, \cdot)\) be a cyclic group of order \(q = |G|\) with generator \(g\) \((G = \{g^0, g^1, \ldots, g^{q-1}\})\). Given elements \(h_1, h_2 \in G\), define

\[
DH(h_1, h_2) := g^{\log, h_1 \cdot \log, h_2}
\]

that is if \(g^{x_1} = h_1\) and \(g^{x_2} = h_2\), then \(DH(h_1, h_2) = g^{x_1 \cdot x_2} = h_1^{x_2} = h_2^{x_1}\).

These two problems are related to the discrete logarithm problem:

- **Computational Diffie–Hellman (CDH) problem**: the adversary is given uniformly chosen \(h_1, h_2 \in G\) and has to output \(DH(h_1, h_2)\).
  The problem is hard if for all PPT \(A\) we have \(|P(A(G, g, g^x, g^y)) = g^{xy}| \leq \text{neg}(t)\).

- **Decision Diffie–Hellman (DDH) problem**: the adversary is given \(h_1, h_2 \in G\) chosen uniformly at random, plus another value \(h' \in G\), which is either equal to \(DH(h_1, h_2)\), or was chosen uniformly at random, and has to decide which of the two cases applies.
  The problem is hard if for all PPT \(A\) and uniform \(x, y, z \in \mathbb{Z}\) we have
  \[
  |P(A(G, g, g^x, g^y, g^z)) = 1) - P(A(G, g, g^x, g^y, g^{xy})) = 1)| \leq \text{neg}(t).
  \]

If the discrete-logarithm problem is not hard for \(G\), then neither will be the CDH problem, and if the latter is not hard, neither will be the DDH problem.

Discrete logarithm algorithms

Several generic algorithms are known for solving the discrete logarithm problem for any cyclic group \(G\) of order \(q\):

- **Trivial brute-force algorithm**: try all \(g^i\), time \(|\langle g \rangle| = \text{ord}(g) \leq q\).

- **Pollard–Hellman algorithm**: if \(q\) is not prime, and has a known (or easy to determine) factorization, then this algorithm reduces the discrete-logarithm problem for \(G\) to discrete-logarithm problems for prime-order subgroups of \(G\).

- **Shank’s baby-step/giant-step algorithm**: requires \(O(\sqrt{q} \cdot \text{polylog}(q))\) time and \(O(\sqrt{q})\) memory.

- **Pollard’s rho algorithm**: requires \(O(\sqrt{q} \cdot \text{polylog}(q))\) time and \(O(1)\) memory.

- **Choose \(G\) to have a prime order \(q\), and make \(q\) large enough such that no adversary can be expected to execute \(\sqrt{q}\) steps** (e.g. \(q \gg 2^{200}\)).

Diffie–Hellman key exchange

How can two parties achieve message confidentiality who have no prior shared secret and no secure channel to exchange one?

Select a cyclic group \(G\) of order \(q\) and a generator \(g \in G\), which can be made public and fixed system wide. \(A\) generates \(x\) and \(B\) generates \(y\), both chosen uniformly at random out of \(\{1, \ldots, q-1\}\). Then they exchange two messages:

\[
A \to B: \quad g^x \\
B \to A: \quad g^y
\]

Now both can form \((g^y)^x = (g^x)^y = g^{xy}\) and use a hash \(h(g^{xy})\) as a shared private key (e.g. with an authenticated encryption scheme).

The eavesdropper faces the computational Diffie–Hellman problem of determining \(g^{xy}\) from \(g^x, g^y\) and \(g\).

The DH key exchange is secure against a passive eavesdropper, but not against middleperson attacks, where \(g^x\) and \(g^y\) are replaced by the attacker with other values.


Baby-step/giant-step algorithm

Given generator \(g \in G\) \((|G| = q)\) and \(y \in G\), find \(x \in \mathbb{Z}_q\) with \(g^x = y\).

- **Powers of \(g\) form a cycle** 1 = \(g^0, g^1, g^2, \ldots, g^{q-1}, g^0 = 1\), and \(y = g^x\) sits on this cycle.

- **Go around cycle in “giant steps” of \(n = \lceil \sqrt{q} \rceil\)**:

  \[
g^0, g^n, g^{2n}, \ldots, g^{\lceil q/n \rceil n}
  \]

  Store all values encountered in a lookup table \(L[y^{kn}] := k\).
  Memory: \(\sqrt{q}\), runtime: \(\sqrt{q}\), \(O(\text{times log. lookup table insertion})\)

- **Go around cycle in “baby steps”, starting at \(y\)**

  \[
y \cdot g^1, y \cdot g^2, \ldots, y \cdot g^n
  \]

  until we find one of these values in the table \(L\): \(L[y \cdot g^i] = k\).
  Runtime: \(\sqrt{q}\) \(O(\text{times log. table lookup})\)

- **Now we know \(y \cdot g^i = g^{kn}\), therefore \(y = g^{kn-1}\) and can return \(x := (kn - i) \mod q = \log_g y\)**.

Compare with time–memory tradeoff on slide 137.
Discrete logarithm algorithms for $\mathbb{Z}_p^*$

The Index Calculus Algorithm computes discrete logarithms in the cyclic group $\mathbb{Z}_p^*$. Unlike the generic algorithms, it has sub-exponential runtime

$$2^{O(\sqrt{\log p} \log \log p)}$$

Therefore, prime $p$ bit-length in cyclic group $\mathbb{Z}_p^*$ has to be much longer than a symmetric key of equivalent attack cost. In contrast, the bit-length of the order $q$ of the subgroup used merely has to be doubled.

Elliptic-curve groups over $\mathbb{Z}_p^*$ or $\text{GF}(p^n)$ exist that are not believed to be vulnerable to the Index Calculus Algorithm.

Equivalent key lengths: (NIST)

<table>
<thead>
<tr>
<th>RSA</th>
<th>Discrete logarithm problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>private key length</td>
<td>factoring $n = pq$</td>
</tr>
<tr>
<td>80 bits</td>
<td>1024 bits</td>
</tr>
<tr>
<td>112 bits</td>
<td>2048 bits</td>
</tr>
<tr>
<td>128 bits</td>
<td>3072 bits</td>
</tr>
<tr>
<td>192 bits</td>
<td>7680 bits</td>
</tr>
<tr>
<td>256 bits</td>
<td>15360 bits</td>
</tr>
</tbody>
</table>

Schnorr groups (proofs)

Let $p = rq + 1$ with $p, q$ prime and $\mathbb{G} = \langle h^r \mod p | h \in \mathbb{Z}_p^* \rangle$. Then

1. $\mathbb{G}$ is a subgroup of $\mathbb{Z}_p^*$.
   Proof: $\mathbb{G}$ is closed under multiplication, as for all $x, y \in \mathbb{G}$ we have $x^y \mod p = (x^y \mod p)^1 \mod p \in \mathbb{G}$ as $(x \mod p) \in \mathbb{Z}_p^*$.
   In addition, $\mathbb{G}$ includes the neutral element $1^1 = 1$.
   For each $h^r$, it also includes the inverse element $(h^{-1})^r \mod p$.

2. $\mathbb{G}$ has $q = (p-1)/r$ elements.
   Proof: The idea is to show that the function $f_r: \mathbb{Z}_r^* \to \mathbb{G}$ with $f_r(a) = a^r \mod p$ is an $r$-to-1 function, and then since $|\mathbb{Z}_r^*| = p - 1$ this will show that $|\mathbb{G}| = q = (p-1)/r$.
   Let $g$ be a generator of $\mathbb{Z}_r^*$ such that $(g^0, g^1, \ldots, g^{p-2}) = \mathbb{Z}_r^*$. Under what condition for $i, j$ is $(i^j)^r \equiv (g^i)^r \mod p$? $(g^i)^r \equiv (g^{i+jr}) \mod p \iff r | i$, so by $\iff (p-1)(r - i) \iff rq(i - j) \iff q(i - j)$.
   For any fixed $j \in \{0, \ldots, p-2\} = \mathbb{Z}_r^*$, what values of $i \in \mathbb{Z}_r^*$, $q(i - j)$, and how many such values are there? For each $j$, there are exactly the $r$ different values of $i$ such that $i \equiv j \mod (p-1)$. This makes $f_r$ an $r$-to-1 function.

3. For any $h \in \mathbb{Z}_p^*$, $h^r$ is either 1 or a generator of $\mathbb{G}$.
   Proof: $h \in \mathbb{G}$ by definition and $[\mathbb{G}]$ prime $\Rightarrow \text{ord}_{\mathbb{G}}(h^r) \in \{1, [\mathbb{G}]\}$ (Lagrange).

4. $h \in \mathbb{G} \iff h \in \mathbb{Z}_p^* \land h^q \mod p = 1$. (Useful security check!)
   Proof: Let $h = g^i$ with $(g^i) = \mathbb{Z}_p^*$ and $0 \leq i < p - 1$. Then $h^q \mod p = 1 \Rightarrow g^{iq} \mod p = 1 \Rightarrow g^{iq} \mod (p - 1) = 0 \Rightarrow q | i \Rightarrow r | i.$

Elliptic-curve groups

Schnorr groups – working in subgroups of $\mathbb{Z}_p^*$

**Schnorr group:** cyclic subgroup $\mathbb{G} = \langle g \rangle \subset \mathbb{Z}_p^*$ with prime order $q = |\mathbb{G}| = (p-1)/r$, where $(p, q, g)$ are generated with:

1. Choose primes $p \gg q$ with $q = r + 1$ for $r \in \mathbb{N}$
2. Choose $1 < h < p$ with $h^r \mod p \neq 1$
3. Use $g := h^r \mod p$ as generator for $\mathbb{G} = \langle g \rangle = \{ h^r \mod p | h \in \mathbb{Z}_p^* \}$

Advantages:

- Select bit-length of $p$ and $q$ independently, based on respective security requirements (e.g. 128-bit security: 3072-bit $p$, 256-bit $q$)
- Difficulty of Discrete Logarithm problem over $\mathbb{G} \subseteq \mathbb{Z}_p^*$ with order $q = |\mathbb{G}|$ depends on both $p$ (subexponentially) and $q$ (exponentially).
- Some operations faster than if $\log_2 q \approx \log_2 p$
- Square-and-multiply exponentiation $g^r \mod p$ (with $x < q$) run-time $\sim \log_2 x < \log_2 q$
- Prime order $q$ has several advantages:
  - Simple choice of generator (pick any element $\neq 1$)
  - $\mathbb{G}$ has no (non-trivial) subgroups $\Rightarrow$ no small subgroup confinement attacks
  - $q$ with small prime factors can make Decision Diffie–Hellman problem easy to solve (Exercise 28)

Compare with slide 176 where $r = 2$. 

Elliptic curves are sets of 2-D coordinates $(x, y)$ with

$$y^2 = x^3 + Ax + B$$

plus one additional “point at infinity” $O$.

Group operation $P_1 + P_2$: draw line through curve points $P_1, P_2$, intersect with curve to get third point $P_3$, then negate the $y$ coordinate of $P_3$ to get $P_1 + P_2$.

Neutral element: $O$ – intersects any vertical line. Inverse: $-(x, y) = (x, -y)$

Curve compression: for any given $x$, encoding $y$ requires only one bit
Elliptic-curve discrete-logarithm problem

The elliptic-curve operation is traditionally written as an additive group, so that the “exponentiation” of the elliptic-curve discrete-logarithm problem (ECDLP) becomes multiplication:

\[ x \cdot G = G + \cdots + G \quad x \in \mathbb{Z}_p \]

So the square-and-multiply algorithm becomes double-and-add, and Diffie–Hellman becomes DH(x \cdot G, y \cdot G) = xy \cdot G for x, y \in \mathbb{Z}_p.

Many curve parameters and cyclic subgroups for which ECDLP is believed to be hard have been proposed or standardised.

Example: NIST P-256

\[
p = 0 \ldots f7a1e79f4d8a69a6813e7b7d4a3e7b1d71fe2592136a9b4f1f2ba2b50f17d2f3917b9f8\]

\[
q = 0 \ldots 3d22551 \]

\[
A = 3\]

\[
B = 0 \ldots 123819336672627468359397546336387696598057789954576274983756986866b651d06b0a6b83b630e63b3c3e3d27d2604b\]

\[
G = (0x6b7d1f12e10c424f7b0c6e63a404277037f612debb33a0f4a13945d898c296, 0xc4fe342e21eaf7988e8e7e4a7c0f9e16b2b335763b15e5eb6b4068373f5f1f5)\]

Note: \( p = 2^{256} - 2^{224} + 2^{192} - 2^{96} - 1 \) and \( q \approx 2^{256} - 2^{224} - 2^{192} \) here are generalised resp. pseudo-Mersenne primes, for fast mod calculation on 32-bit CPUs and good use of the 256-bit space.

Elliptic-curve groups with prime order

How large are elliptic curves over \( \mathbb{Z}_p \)?

Equation \( y^2 = f(x) \) has two solutions if \( f(x) \) is a quadratic residue, and one solution if \( f(x) = 0 \). Half of the elements in \( \mathbb{Z}_p \) are quadratic residues, so expect around \( 2 \cdot (p-1)/2 + 1 = p \) points on the curve.

Hasse bound: \( p + 1 - 2\sqrt{p} \leq |E(\mathbb{Z}_p, A, B)| \leq p + 1 + 2\sqrt{p} \)

Actual group order: approximately uniformly spread over Hasse bound.

Elliptic curves became usable for cryptography with the invention of efficient algorithms for counting the exact number of points on them.

Generate a cyclic elliptic-curve group \((p, q, A, B, G)\) with:

1. Choose uniform \( n \)-bit prime \( p \)
2. Choose \( A, B \in \mathbb{Z}_p \) with \( 4A^3 + 27B^2 \neq 0 \mod p \), determine \( q = |E(\mathbb{Z}_p, A, B)| \), repeat until \( q \) is an \( n \)-bit prime
3. Choose \( G \in E(\mathbb{Z}_p, A, B) \setminus \{O\} \) as generator

Easy to find a point \( G = (x, y) \) on the curve: pick uniform \( x \in \mathbb{Z}_p \) until \( f(x) \) is a quadratic residue or 0, then set \( y = \sqrt{f(x)} \).

Commonly used standard curves

NIST FIPS 186-4 has standardized five such elliptic curves over integer field \((\mathbb{Z}_p)\) parameters: P-192, P-224, P-256, P-384, P-521.

Also: five random curves of the form \( y^2 + xy = x^3 + Ax + B \) over binary field \((GF(2^n))\) coordinates: B-163, B-233, B-283, B-409, B-571. The number of points on these curves is twice the order of the base point \( G \) ("cofactor 2").

And there are five Koblitz curves of the form \( y^2 + xy = x^3 + Ax + 1 \) \((A \in \{0, 1\}, \text{ with cofactors 4 or 2, resp.})\) also over \((GF(2^n))\): K-163, K-233, K-283, K-409, K-571. (Koblitz: \( A, B \in \{0, 1\} \Rightarrow \) faster.)

Some mistrust the NIST parameters for potentially having been carefully selected by the NSA, to embed a vulnerability. http://www.rsc-circuits.org.uk/rigid.html

Brainpool (RFC 5639): seven similar curves over \( \mathbb{Z}_p \), chosen by the German government.

The Standards for Efficient Cryptography Group SEC 2 specification lists eight curves over \( \mathbb{Z}_p \) (secp192k1, secp224k1, secp256k1, secp384k1, secp521k1) and 12 over \((GF(2^n))\) (sect163k1, ..., sect571k1).

The numbers indicate the bit length of one coordinate, i.e. roughly twice the equivalent symmetric-key strength.

ANSI X9.62 (and SEC 1) define a compact binary syntax for curve points.

Curve25519 was proposed by Daniel J. Bernstein in 2005 and has since become a highly popular P-256 alternative due to faster implementation, better resiliency against some implementation vulnerabilities (e.g., timing attacks), lack of patents and worries about NSA backdoors.
ElGamal encryption scheme

The DH key exchange requires two messages. This can be eliminated if everyone publishes their \( g^x \) as a public key in a sort of phonebook. Assume \( (G, \cdot, q, g) \) are fixed for all participants. 

A chooses secret key \( x \in \mathbb{Z}_q^* \) and publishes \( g^x \in G \) as her public key. B generates for each message a new nonce \( y \in \mathbb{Z}_q^* \) and then sends

\[
B \to A : \quad g^y, \ (g^x)^y \cdot M
\]

where \( M \in G \) is the message that \( B \) sends to \( A \) in this asymmetric encryption scheme. Then \( A \) calculates

\[
[(g^x)^y \cdot M] \cdot [(g^y)^{y^{-1}}] = M
\]

to decrypt \( M \).

In practice, this scheme is rarely used because of the difficulty of fitting blockciper-based authenticated encryption scheme that protects the confidentiality and integrity of the bulk of the message \( M \):

\[
B \to A : \quad g^y, \ \text{Enc}_K(M)
\]

Number theory: easy and difficult problems

**Easy:**
- given integer \( n, i \) and \( x \in \mathbb{Z}_n^* \): calculate \( x^{-1} \in \mathbb{Z}_n^* \) or \( x^i \in \mathbb{Z}_n^* \)
- given prime \( p \) and polynomial \( f(x) \in \mathbb{Z}_p[x] \):
  - find \( x \in \mathbb{Z}_p \) with \( f(x) = 0 \)
  - runtime grows linearly with the degree of the polynomial

**Difficult:**
- given safe prime \( p \), generator \( g \in \mathbb{Z}_p^* \) (or large subgroup):
  - given value \( a \in \mathbb{Z}_p^* \): find \( x \) such that \( a = g^x \).
    - \( \text{Discrete Logarithm Problem} \)
  - given values \( g', g'' \in \mathbb{Z}_p^* \): find \( g'' \).
    - \( \text{Computational Diffie–Hellman Problem} \)
  - given values \( g', g'' \in \mathbb{Z}_p^* \): tell whether \( z = g'' \).
    - \( \text{Decision Diffie–Hellman Problem} \)
- given a random \( n = p \cdot q \), where \( p \) and \( q \) are \( \ell \)-bit primes (\( \ell \geq 1024 \)):
  - find integers \( p \) and \( q \) such that \( n = p \cdot q \) in \( \mathbb{N} \)
    - \( \text{Factoring Problem} \)
  - given a polynomial \( f(x) \) of degree \( > 1 \):
    - find \( x \in \mathbb{Z}_n \) such that \( f(x) = 0 \) in \( \mathbb{Z}_n \)

“Textbook” RSA encryption

**Key generation**
- Choose random prime numbers \( p \) and \( q \) (each \( \approx 1024 \) bits long)
- \( n := pq \) (\( \approx 2048 \) bits = key length) \( \varphi(n) = (p - 1)(q - 1) \)
- pick integer values \( e, d \) such that: \( e d \mod \varphi(n) = 1 \)
- public key \( PK := (n, e) \)
- secret key \( SK := (n, d) \)

**Encryption**
- input plaintext \( M \in \mathbb{Z}_n^* \), public key \( (n, e) \)
- \( C := M^e \mod n \)

**Decryption**
- input ciphertext \( C \in \mathbb{Z}_n^* \), secret key \( (n, d) \)
- \( M := C^d \mod n \)

In \( \mathbb{Z}_n^* \):

\[
(M^e)^d = M^{ed} = M^{ed \mod \varphi(n)} = M^1 = M.
\]

Common implementation tricks to speed up computation:
- Choose small \( e \) with low Hamming weight (e.g., \( 3, 17, 2^{16} + 1 \)) for faster modular encryption
- Preserve factors of \( n \) in \( SK = (p, q, d) \), decryption in both \( \mathbb{Z}_p \) and \( \mathbb{Z}_q \), use Chinese remainder theorem to recover result in \( \mathbb{Z}_n \).
"Textbook" RSA is not secure

There are significant security problems with a naive application of the basic "textbook" RSA encryption function $C := P^e \mod n$:

- deterministic encryption: cannot be CPA secure
- malleability:
  - adversary intercepts C and replaces it with $C' := X^e \cdot C$
  - recipient decrypts $M' = \text{Dec}_{SK}(C') = X \cdot M \mod n$
- chosen-ciphertext attack recovers plaintext:
  - adversary intercepts C and replaces it with $C' := R^e \cdot C \mod n$
  - decryption oracle provides $M' = \text{Dec}_{SK}(C') = R \cdot M \mod n$
  - adversary recovers $M = M' \cdot R^{-1} \mod n$
- Small value of $M$ (e.g., 128-bit AES key), small exponent $e = 3$:
  - if $M^e < n$ then $C = M^e \mod n = M^e$ and then $M = \sqrt[3]{\text{C}}$ can be calculated efficiently in $\mathbb{Z}$ (no modular arithmetic!)
- many other attacks exist ...

Secure trapdoor permutations

Trapdoor permutation: $\Pi = (\text{Gen}, F, F^{-1})$

Experiment/game $\text{TDef}_{\mathcal{A},n}(\ell)$:

$\ell \xrightarrow{\text{1}} (PK, SK) \leftarrow \text{Gen}(\ell')$

$\begin{array}{c|c|c}
 & x \leftarrow \mathcal{X} & y := F_{PK}(x) \\
\hline
\text{challenger} & PK, y & \mathcal{A} \\
\hline
\text{adversary} & x' \\
\end{array}$

1. The challenger generates a key pair $(PK, SK) \leftarrow \text{Gen}(\ell')$ and a random value $x \in \mathcal{X}$ from the domain of $F_{PK}$.
2. The adversary $\mathcal{A}$ is given inputs $PK$ and $y := F_{PK}(x)$.
3. Finally, $\mathcal{A}$ outputs $x'$.

If $x' = x$ then $\mathcal{A}$ has succeeded: $\text{TDef}_{\mathcal{A},n}(\ell) = 1$.

A trapdoor permutation $\Pi$ is secure if for all probabilistic polynomial time adversaries $\mathcal{A}$ the probability of success $\mathbb{P}(\text{TDef}_{\mathcal{A},n}(\ell) = 1)$ is negligible.

While the definition of a trapdoor permutation resembles that of a public-key encryption scheme, its security definition does not provide the adversary any control over the input (plaintext).

Trapdoor permutations

A trapdoor permutation is a tuple of polynomial-time algorithms $(\text{Gen}, F, F^{-1})$ such that

- the key generation algorithm $\text{Gen}$ receives a security parameter $\ell$ and outputs a pair of keys $(PK, SK) \leftarrow \text{Gen}(\ell')$, with key lengths $|PK| \geq \ell$, $|SK| \geq \ell$
- the sampling function $F$ maps a public key $PK$ and a value $x \in \mathcal{X}$ to a value $y := F_{PK}(x) \in \mathcal{X}$
- the inverting function $F^{-1}$ maps a secret key $SK$ and a value $y \in \mathcal{X}$ to a value $x := F_{SK}^{-1}(y) \in \mathcal{X}$
- for all $\ell$, $(PK, SK) \leftarrow \text{Gen}(\ell')$, $x \in \mathcal{X}$: $F_{SK}^{-1}(F_{PK}(x)) = x$.

In practice, the domain $\mathcal{X}$ may depend on $PK$.

This looks almost like the definition of a public-key encryption scheme, the difference being
- $F$ is deterministic;
- the associated security definition.

Public-key encryption scheme from trapdoor permutation

Trapdoor permutation: $\Pi_{\text{TD}} = (\text{Gen}_{\text{TD}}, F, F^{-1})$ with $F_{PK} : \mathcal{X} \leftrightarrow \mathcal{X}$

Authentic. encrypt. scheme: $\Pi_{\text{AE}} = (\text{Gen}_{\text{AE}}, \text{Enc}, \text{Dec})$, key space $K$

Secure hash function $h : \mathcal{X} \rightarrow K$

We define the private-key encryption scheme $\Pi = (\text{Gen}', \text{Enc}', \text{Dec}')$:

- $\text{Gen}'$: output key pair $(PK, SK) \leftarrow \text{Gen}_{\text{TD}}(\ell')$
- $\text{Enc}'$: on input of plaintext message $M$, generate random $x \in \mathcal{X}$, $y := F(x)$, $K = h(x)$, $C \leftarrow \text{Enc}_{K}(M)$, output ciphertext $(y, C)$
- $\text{Dec}'$: on input of ciphertext message $C = (y, C)$, recover $K = h(F^{-1}(y))$, output $\text{Dec}_{K}(C)$

Encrypted message: $F(x), \text{Enc}_{h(x)}(M)$

The trapdoor permutation is only used to communicate a "session key" $h(x)$, the actual message is protected by a symmetric authenticated encryption scheme. The adversary $\mathcal{A}$ in the PubKCCA game has no influence over the input of $F$.

If hash function $h$ is replaced with a "random oracle" (something that just picks a random output value for each input from $\mathcal{X}$), the resulting public-key encryption scheme $\Pi'$ is CCA secure.
Using RSA as a CCA-secure encryption scheme

**Solution 1:** use only as trapdoor function to build encryption scheme

- Pick random value \( x \in \mathbb{Z}_n^* \)
- Ciphertext is \((x^e \mod n, \text{Enc}_{h(x)}(M))\), where \(\text{Enc}\) is from an authenticated encryption scheme

**Solution 2:** Optimal Asymmetric Encryption Padding

Make \( M \) (with zero padding) the left half, and a random string \( R \) the right half, of the input of a two-round Feistel cipher, using a secure hash function as the round function.

Interpret the result \((X, Y)\) as an integer \( M' \).

Then calculate \( C := M'^e \mod n \).

PKCS #1 v2.0

Practical pitfalls with implementing RSA

- low entropy of random-number generator seed when generating \( p \) and \( q \) (e.g., in embedded devices):
  - take public RSA modulus \( n_1 \) and \( n_2 \) from two devices
  - test \( \gcd(n_1, n_2) \neq 1 \Rightarrow \) if no, \( n_1 \) and \( n_2 \) share this number as a common factor
  - February 2012 experiments: worked for many public HTTPS keys

Lenstra et al.: Public keys, CRYPTO 2012
Heninger et al.: Mining your Ps and Qs, USENIX Security 2012.

One-time signatures

A simple digital signature scheme can be built using a one-way function \( h \) (e.g., secure hash function):

**Secret key:** 2\( n \) random bit strings \( R_{i,j} \) \((i \in \{0,1\}, 1 \leq j \leq n)\)

**Public key:** 2\( n \) bit strings \( h(R_{i,j}) \)

**Signature:** \((R_{b_1,1}, R_{b_2,2}, \ldots, R_{b_n,n})\), where \( h(M) = b_1b_2\ldots b_n \)
Digital Signature Algorithm (DSA)

Let \((G, q, g)\) be system-wide choices of a cyclic group \(G\) of order \(q\) with generator \(g\). In addition, we need two functions \(H : \{0, 1\}^* \rightarrow \mathbb{Z}_q\) and \(F : G \rightarrow \mathbb{Z}_q\) where \(H\) must be collision resistant. Both \(H\) and \(F\) are random oracles in security proofs, but common \(F\) not even preimage resistant.

**Key generation:** uniform secret key \(x \in \mathbb{Z}_q\), then public key \(y := g^x \in G\).

**Signing:** On input of a secret key \(x \in \mathbb{Z}_q\) and a message \(m \in \{0, 1\}^*\), first choose (for each message!) uniformly at random \(k \in \mathbb{Z}_q^*\) and set \(r := F(g^k)\). Then solve the linear equation

\[
k \cdot s - x \cdot r \equiv H(m) \pmod{q}
\]

for \(s := k^{-1} \cdot (H(m) + xr) \mod q\). If \(r = 0\) or \(s = 0\), restart with a fresh \(k\), otherwise output \(\text{Sign}(m) \leftarrow (r, s)\).

**Verification:** On input of public key \(y\), message \(m\), and signature \((r, s)\), verify equation (1) after both sides have been turned into exponents of \(g\):

\[
g^{sx}/g^{xr} = g^{H(m)}
\]

(2)

\[
(g^{sx})^r = g^{H(m)y^r}
\]

(3)

\[
g^k = g^{H(m)s^{-1}} \cdot g^{r^{-1}}
\]

(4)

\[
\implies \text{actually verify: } \quad r = F(g^{H(m)s^{-1}}) \cdot y^{s^{-1}}
\]

(5)

**DSA variants**

**ElGamal signature scheme**

The DSA idea was originally proposed by ElGamal with \(G = \mathbb{Z}_p^*\), \(\text{ord}(g) = q = p - 1\) and \(F(x) = x\).

Unless the \(p\) and \(q\) are chosen more carefully, ElGamal signatures can be vulnerable to forgery:

D. Bleichenbacher: Generating ElGamal signatures without knowing the secret key.

EUROCRYPT '96. [http://www.springerlink.com/link.asp?id=xbwmtv646gwlq7a](http://www.springerlink.com/link.asp?id=xbwmtv646gwlq7a)

**NIST DSA**

In 1993, the US government standardized the Digital Signature Algorithm, a modification of the ElGamal signature scheme where

- \(G\) is a prime-order subgroup of \(\mathbb{Z}_p^*\)
- prime number \(p\) (1024 bits), prime number \(q\) (160 bits) divides \(p - 1\)
- \(g = h(s^{-1}) \mod p\), with \(1 < h < p - 1\) so that \(g > 1\) (e.g., \(h = 2\))
- \(H\) is SHA-1
- \(F(x) = x \mod q\)

Generate key: random \(0 < x < q\), \(y := g^x \mod p\).

Signature \((r, s) := \text{Sign}_x(m)\): random \(0 < k < q\),

\[
r := (g^k \mod p) \mod q, \quad s := (k^{-1}(H(m) + x \cdot r)) \mod q
\]

Later versions of the DSA standard FIPS 186 added larger values for \((p, q, g)\), as well as ECDSA, where \(G\) is one of several elliptic-curve groups over \(\mathbb{Z}_p\) or \(\mathbb{G}(2^a)\) and \(F((x, y)) = x \mod q\).

Proper generation of \(k\) is important

DSA fails catastrophically if the adversary can ever guess \(k\):

\[
s \equiv k^{-1} \cdot (H(m) + xr) \quad \Rightarrow \quad x \equiv (k \cdot s - H(m)) \cdot r^{-1} \pmod{q}
\]

All that is needed for \(k\) to leak is two messages \(m \neq m'\) signed with the same \(k = k'\) (easily recognized from \(r = r' = F(g^k)\)):

\[
s \equiv k^{-1} \cdot (H(m) + xr)
\]

\[
 s' \equiv k^{-1} \cdot (H(m') + xr)
\]

\[
s - s' \equiv k^{-1} \cdot (H(m) - H(m'))
\]

\[
k \equiv (H(m) - H(m'))(s - s')^{-1} \pmod{q}
\]

Sony used a fixed \(k\) in firmware signatures for their PlayStation 3 (fail0verflow, 27th Chaos Communication Conf., Berlin 2010).

Without a good random-bit generator to generate \(k\), use e.g. \(k := \text{SHA}-3(x || m) \mod q\) (with hash output longer than \(q\)).
Public-key infrastructure I

Public key encryption and signature algorithms allow the establishment of confidential and authenticated communication links with the owners of public/secret key pairs.

Public keys still need to be reliably associated with identities of owners. In the absence of a personal exchange of public keys, this can be mediated via a trusted third party. Such a certification authority C issues a digitally signed public key certificate

\[ \text{Cert}_C(A) = (A, PK_A, T, L, N, \text{Sign}_{SK_C}(A, PK_A, T, L, N)) \]

in which C confirms that the public key \( PK_A \) belongs to entity A, starting at time \( T \) and that this confirmation is valid for the time interval \( L \), and all this has a serial number \( N \) and is digitally signed with C’s secret signing key \( SK_C \).

Anyone who knows C’s public key \( PK_C \) from a trustworthy source can use it to verify the certificate \( \text{Cert}_C(A) \) and obtain a trustworthy copy of A’s public key \( PK_A \) this way.

Public-key infrastructure II

We can use the operator \( \bullet \) to describe the extraction of A’s public key \( PK_A \) from a certificate \( \text{Cert}_C(A) \) with the certification authority public key \( PK_C \):

\[ PK_C \bullet \text{Cert}_C(A) = \begin{cases} PK_A & \text{if certificate valid} \\ \text{failure} & \text{otherwise} \end{cases} \]

The \( \bullet \) operation involves not only the verification of the certificate signature, but also the validity time and other restrictions specified in the signature. For instance, a certificate issued by C might contain a reference to an online certificate revocation list published by C, which lists the serial numbers \( N \) of all certificates of public keys that might have become compromised (e.g., the smartcard containing \( SK_A \) was stolen or the server storing \( SK_A \) was broken into) and whose certificates have not yet expired.

Public-key infrastructure III

Public keys can also be verified via several trusted intermediaries in a certificate chain:

\[ PK_{C_1} \bullet \text{Cert}_{C_2}(C_2) \bullet \text{Cert}_{C_2}(C_3) \bullet \cdots \bullet \text{Cert}_{C_n}(C_n) \bullet \text{Cert}_{C_1}(B) = PK_B \]

A has received directly a trustworthy copy of \( PK_{C_1} \) (which many implementations store locally as a certificate \( \text{Cert}_A(C_1) \) to minimise the number of keys that must be kept in tamper-resistant storage).

Certification authorities could be made part of a hierarchical tree, in which members of layer \( n \) verify the identity of members in layer \( n - 1 \) and \( n + 1 \). For example layer 1 can be a national CA, layer 2 the computing services of universities and layer 3 the system administrators of individual departments.

Practical example: A personally receives \( K_{C_1} \) from her local system administrator \( C_1 \), who confirmed the identity of the university’s computing service \( C_2 \) in \( \text{Cert}_{C_2}(C_2) \), who confirmed the national network operator \( C_4 \), who confirmed the IT department of B’s employer \( C_4 \) who finally confirms the identity of B. An online directory service allows A to retrieve all these certificates (plus related certificate revocation lists) efficiently.

In today’s Transport Layer Security (TLS) practice (HTTPS, etc.), most private users use their web-browser or operating-system vendor as their sole trusted source of \( PK_{C_1} \) root keys.

Outlook

Goals of this part of the course were

- introduce secure hash functions and some of their applications
- introduce some of the number-theory and abstract-algebra concepts behind the main public-key encryption and signature schemes, in particular the discrete logarithm problem, the Diffie-Hellman key exchange, RSA encryption, and the Digital Signature Algorithm

Modern cryptography is still a young discipline (born in the early 1980s), but well on its way from a collection of tricks to a discipline with solid theoretical foundations.

Some important concepts that we did not touch here for time reasons:

- password-authenticated key exchange
- identity-based encryption
- side-channel and fault attacks
- application protocols: electronic voting, digital cash, etc.
- secure multi-party computation
- post-quantum cryptography
Appendix

Confidentiality games at a glance

| PRIVK^{env} | Privacy env variability | 1^{e} | b \in_R \{0,1\} | \K \leftarrow \text{Gen}(1^{e}) | C \leftarrow \text{Enc}_{K(M_b)}(\text{challenger}) | M_0, M_1 | A \ \text{adversary} | 1^{e} | b' |
| PRIVK^{multi} | Privacy multi variability | 1^{e} | b \in_R \{0,1\} | \K \leftarrow \text{Gen}(1^{e}) | C \leftarrow \text{Enc}_{K(M_b)}(\text{challenger}) | M_0^l, M_1^l, \ldots, M_n^l | M_0^l, M_1^l, \ldots, M_n^l | A \ \text{adversary} | 1^{e} | b' |
| PRIVK^{conf} | Confidentiality variability | 1^{e} | b \in_R \{0,1\} | \K \leftarrow \text{Gen}(1^{e}) | C \leftarrow \text{Enc}_{K(M_b)}(\text{challenger}) | M_0, M_1 | A \ \text{adversary} | 1^{e} | b' |

Some basic discrete mathematics notation

- $|A|$ is the number of elements (size) of the finite set $A$.
- $A_1 \times A_2 \times \cdots \times A_n$ is the set of all $n$-tuples $(a_1, a_2, \ldots, a_n)$ with $a_1 \in A_1$, $a_2 \in A_2$, etc. If all the sets $A_i$ \ (1 \leq i \leq n) \ are finite: $|A_1 \times A_2 \times \cdots \times A_n| = |A_1| \cdot |A_2| \cdots |A_n|$.
- $A^n$ is the set of all $n$-tuples $(a_1, a_2, \ldots, a_n)$ with $a_1, a_2, \ldots, a_n \in A$. If $A$ is finite then $|A^n| = |A|^n$.
- $A^* = \bigcup_{n=0}^\infty A^n$ and $A^+ = \bigcup_{n=0}^\infty A^n$.
- Function $f : A \rightarrow B$ maps each element of $A$ to an element of $B$: $a \mapsto f(a)$ or $b = f(a)$ with $a \in A$ and $b \in B$.
- A function $f : A_1 \times A_2 \times \cdots \times A_n \rightarrow B$ maps each parameter tuple to an element of $B$: $(a_1, a_2, \ldots, a_n) \mapsto f(a_1, a_2, \ldots, a_n)$ or $f(a_1, a_2, \ldots, a_n) = b$.
- A permutation $f : A \leftrightarrow A$ maps $A$ onto itself and is invertible: $x = f^{-1}(f(x))$. There are $|\text{Perm}(A)| = |A|! = 1 \cdot 2 \cdot \cdots \cdot |A|$ permutations over $A$.
- $B^A$ is the set of all functions of the form $f : A \rightarrow B$. If $A$ and $B$ are finite, there will be $|B^A| = |B|^{|A|}$ such functions.

Integrity games at a glance

| PRIVK^{conf} | Privacy confidentiality variability | 1^{e} | b \in_R \{0,1\} | \K \leftarrow \text{Gen}(1^{e}) | C \leftarrow \text{Enc}_{K(M_b)}(\text{challenger}) | M_0, M_1, C \leftarrow \text{Dec}_{K}(C') | M_1^1, M_2^1, C_1^1 | A \ \text{adversary} | 1^{e} | b' |
| PRIVK^{conf} | Privacy confidentiality variability | 1^{e} | b := \text{Vrfy}_K(M,T) | A \ \text{adversary} | 1^{e} | b' |
| PRIVK^{conf} | Privacy confidentiality variability | 1^{e} | b := \text{Vrfy}_K(M,T) | A \ \text{adversary} | 1^{e} | b' |