Cryptography

Markus Kuhn

Department of Computer Science and Technology
University of Cambridge

https://www.cl.cam.ac.uk/teaching/2425/Crypto/

These notes are merely provided as an aid for following the lectures.
They are no substitute for attending the course.

Lent 2025 – CST Part II

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.

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

Related textbooks

Main reference:

- Jonathan Katz, Yehuda Lindell:
  Introduction to Modern Cryptography, Chapman & Hall/CRC

Further reading:

- Christof Paar, Jan Pelzl: **Understanding Cryptography**, Springer

  `https://link.springer.com/book/10.1007/978-3-642-04101-3`
  `https://www.cryptography-textbook.com/`

- Douglas Stinson: **Cryptography – Theory and Practice**,
  3rd ed., CRC Press, 2005

- Menezes, van Oorschot, Vanstone: **Handbook of Applied Cryptography**, CRC Press, 1996

  `https://cacr.uwaterloo.ca/hac/`

- Hankerson, Menezes, Vanstone: **Guide to Elliptic Curve Cryptography**, Springer, 2004

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:

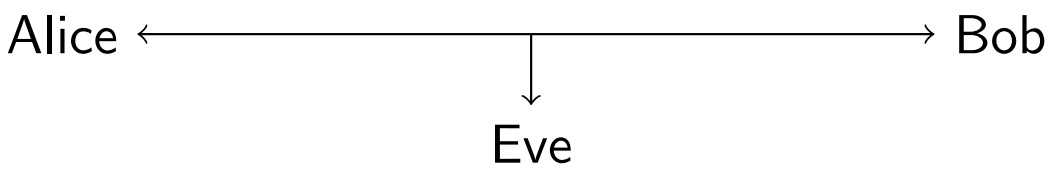
Confidentiality ensuring that information is accessible only to those authorised to have access

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

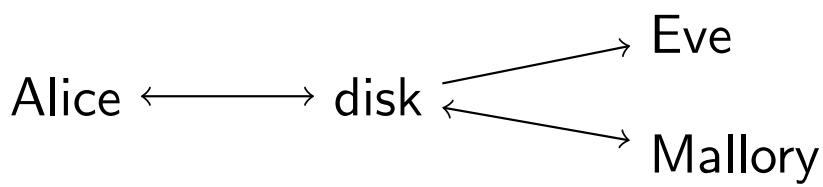
Basic threat scenarios:

Eavesdropper: (passive)



Middle-person attack: (active)

Storage security:



Encryption schemes

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

Private-key (symmetric) encryption scheme

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

Public-key (asymmetric) encryption scheme

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

Message integrity schemes

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

Message authentication code (MAC)

$K \leftarrow \mathsf{Gen}$	private-key generation
$T := \mathsf{Mac}_K(M)$	message tag generation
$M' \neq M \stackrel{?}{\Rightarrow}$ $\mathsf{Mac}_K(M') \neq T$	MAC verification: recalculate and compare tag

Digital signature

$PK, SK \leftarrow \mathsf{Gen}$	public/secret key-pair generation
$S \leftarrow \mathsf{Sign}_{SK}(M)$	signature generation using secret key
$\mathsf{Vrfy}_{PK}(M, S) = 1,$ $M' \neq M \stackrel{?}{\Rightarrow}$ $\mathsf{Vrfy}_{PK}(M', S) = 0$	signature verification using public key

Key exchange

Key-agreement protocol

- $(PK_A, SK_A) \leftarrow \mathsf{Gen}$ public/secret key-pair generation by Alice
- $(PK_B, SK_B) \leftarrow \mathsf{Gen}$ public/secret key-pair generation by Bob
- $K := \mathsf{DH}(SK_A, PK_B)$ key derivation from exchanged public keys
  $= \mathsf{DH}(PK_A, SK_B)$

Diffie–Hellman protocol:

Alice and Bob standardize suitably chosen very large public numbers g, p and q. Alice picks a random number $0 < x < q$ and Bob a secret number $0 < y < q$ as their respective secret keys. They then exchange the corresponding public keys:

$$A \to B: \qquad PK_A = g^x \bmod p$$
$$B \to A: \qquad PK_B = g^y \bmod p$$

Alice and Bob each now can calculate

$$K = (g^y \bmod p)^x \bmod p = (g^x \bmod p)^y \bmod p$$

and use that as a shared private key. With suitably chosen parameters, outside observers will not be able to infer x, y, or K.

Why might one also want to sign or otherwise authenticate PK_A and/or PK_B?

Key types

- **Private** keys = **symmetric** keys
- **Public/secret** key pairs = **asymmetric** keys

  Warning: this "private" vs "secret" key terminology is not universal in the literature

- **Ephemeral** keys / **session** keys are only used briefly and often generated fresh for each communication session.

  They can be used to gain *privacy* (observers cannot identify users from public keys exchanged in clear) and *forward secrecy* (if a communication system gets compromised in future, this will not compromise past communication).

- **Static** keys remain unchanged over a longer period of time (typically months or years) and are usually intended to identify users.

  Static public keys are usually sent as part of a signed "certificate" $\text{Sign}_{SK_C}(A, PK_A)$, where a "trusted third party" or "certification authority" C certifies that PK_A is the public key associated with user A.

- **Master** keys are used to generate other **derived** keys.
- By purpose: encryption, message-integrity, authentication, signing, key-exchange, certification, revocation, attestation, etc. keys

When is a cryptographic scheme "secure"?

For an **encryption scheme**, 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"

For an **integrity scheme**, 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 $C \leftarrow \mathsf{Enc}_K(M)$?
- ability to trick the user of Enc_K into encrypting some plaintext of the adversary's choice and return the result? ("oracle access" to Enc)
- ability to trick the user of Dec_K into decrypting some ciphertext of the adversary's choice and return the result? ("oracle access" to Dec)?
- ability to modify or replace C en route? (not limited to eavesdropping)
- how many applications of Enc_K or 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.

Auguste Kerckhoffs: *La cryptographie militaire*, Journal des sciences militaires, 1883.
`https://petitcolas.net/fabien/kerckhoffs/`

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`
- Data compression can be abused to extract information from an encrypted message if an adversary can control part of that message.
  J. Kelsey: Compression and information leakage of plaintext.
  `https://www.iacr.org/cryptodb/archive/2002/FSE/3091/3091.pdf`
  Also: CVE-2012-4929/CRIME

Demo: leaking plaintext through compressed data length

```
$ cat compression-leak
#!/bin/bash
PLAINTEXT=cafe  ←
KEY="N-32m5qEj/emdVr.69w1fX"
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
```

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

Historic examples of simple ciphers

Shift Cipher: Treat letters $\{\mathsf{A}, \ldots, \mathsf{Z}\}$ 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.

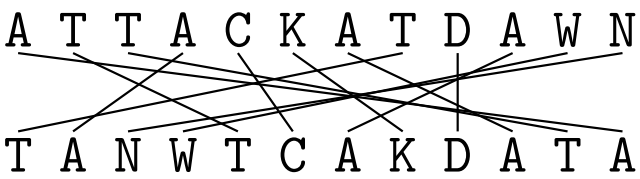
Example with $K = 25 \equiv -1 \pmod{26}$: `IBM`$\rightarrow$`HAL`.

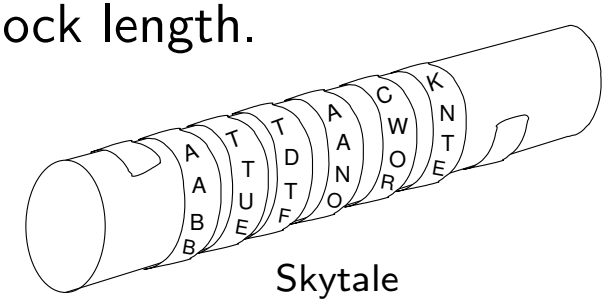
$K = -3$ known as *Caesar Cipher*, $K = 13$ as `rot13`.

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.



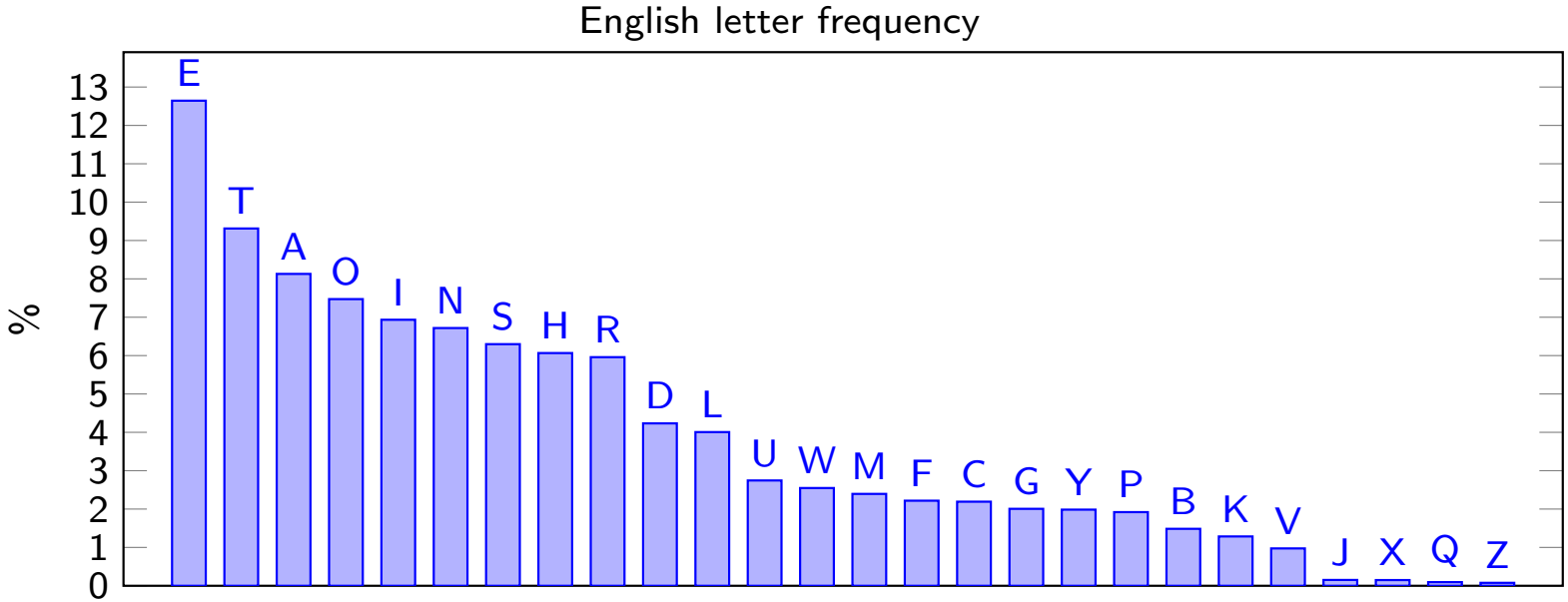


Skytale

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:
E, T, A, O, I, N, S, H, R, D, L, U, ...
The most common digrams 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.

Monoalphabetic 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:

$$c_i = (m_i + k_{[(i-1) \bmod l]+1}) \bmod 26$$

Example: $K = \texttt{SECRET}$

S	*E*	*C*	*R*	*E*	*T*	*S*	*E*	*C*	...
A	*T*	*T*	*A*	*C*	*K*	*A*	*T*	*D*	...
S	*X*	*V*	*R*	*G*	*D*	*S*	*X*	*F*	...

The modular addition can be replaced with XOR:

$$c_i = m_i \oplus k_{[(i-1) \bmod l]+1} \qquad m_i, k_i, c_i \in \{0, 1\}$$

Vigenère is an example of a *polyalphabetic* cipher.

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
BCDEFGHIJKLMNOPQRSTUVWXYZA
CDEFGHIJKLMNOPQRSTUVWXYZAB
DEFGHIJKLMNOPQRSTUVWXYZABC
EFGHIJKLMNOPQRSTUVWXYZABCD
FGHIJKLMNOPQRSTUVWXYZABCDE
GHIJKLMNOPQRSTUVWXYZABCDEF
HIJKLMNOPQRSTUVWXYZABCDEFG
IJKLMNOPQRSTUVWXYZABCDEFGH
JKLMNOPQRSTUVWXYZABCDEFGHI
KLMNOPQRSTUVWXYZABCDEFGHIJ
LMNOPQRSTUVWXYZABCDEFGHIJK
MNOPQRSTUVWXYZABCDEFGHIJKL
NOPQRSTUVWXYZABCDEFGHIJKLM
OPQRSTUVWXYZABCDEFGHIJKLMN
PQRSTUVWXYZABCDEFGHIJKLMNO
QRSTUVWXYZABCDEFGHIJKLMNOP
RSTUVWXYZABCDEFGHIJKLMNOPQ
STUVWXYZABCDEFGHIJKLMNOPQR
TUVWXYZABCDEFGHIJKLMNOPQRS
UVWXYZABCDEFGHIJKLMNOPQRST
VWXYZABCDEFGHIJKLMNOPQRSTU
WXYZABCDEFGHIJKLMNOPQRSTUV
XYZABCDEFGHIJKLMNOPQRSTUVW
YZABCDEFGHIJKLMNOPQRSTUVWX
ZABCDEFGHIJKLMNOPQRSTUVWXY
```

Attacking the Vigenère cipher

First determine the key length l. For each candidate keylength l:

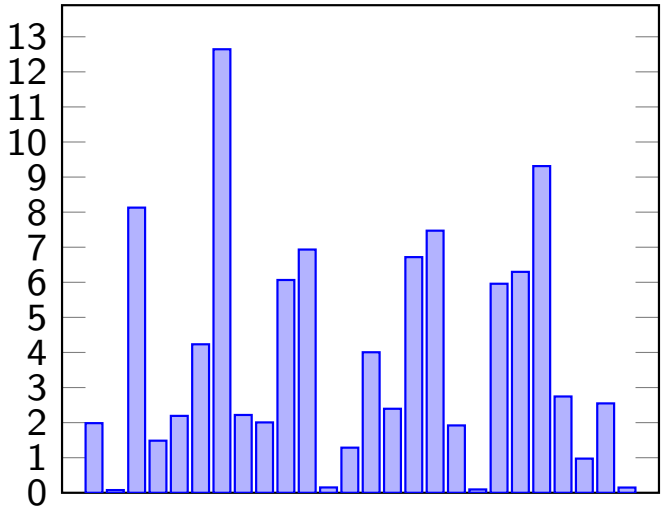
- Treat each l-th ciphertext character as part of a separate message $M_1, M_2, \ldots, M_l$ encrypted with just a (monoalphabetic) shift cipher, resulting in separate ciphertexts $C_1, C_2, \ldots, C_l$.
- Consider the l letter-frequency histograms for these C_i ($1 \leq i \leq l$).
- If choice of l is incorrect, the letter-frequency histograms of each of $C_1, C_2, \ldots, C_l$ will be more even/flatter (as they are the average of several rotated histograms) than if l was correct.
- If $p_{a,i}$ is the relative frequency of letter a in C_i (for all a in alphabet A), then the **index of coincidence**

  $$IC(C_i) = \sum_{a \in A} p_{a,i}^2$$

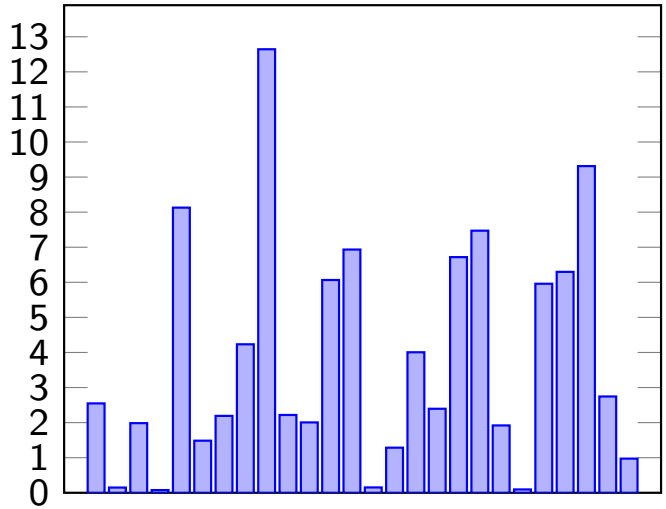
  is the probability that two randomly chosen letters from C_i are identical. IC is a measure of the unevenness of a histogram (minimal if $\forall a \in A : p_{a,i} = |A|^{-1}$).
- Pick the key length l that leads to the highest $l^{-1} \sum_{i=1}^{l} IC(C_i)$. In other words, maximise the probability of two letters being identical when looking only at letters that are a multiple of l characters apart in C.

Once the correct key length l is known, compare the histograms of $C_1, C_2, \ldots, C_l$. They will just be shifted versions of each other ($p_{a,2} = p_{(a-k_2+k_1) \bmod 26,1}$, etc.), and the shift offsets reveal the differences between the corresponding key characters. Finally, try decryption with all possible first key characters k_1.

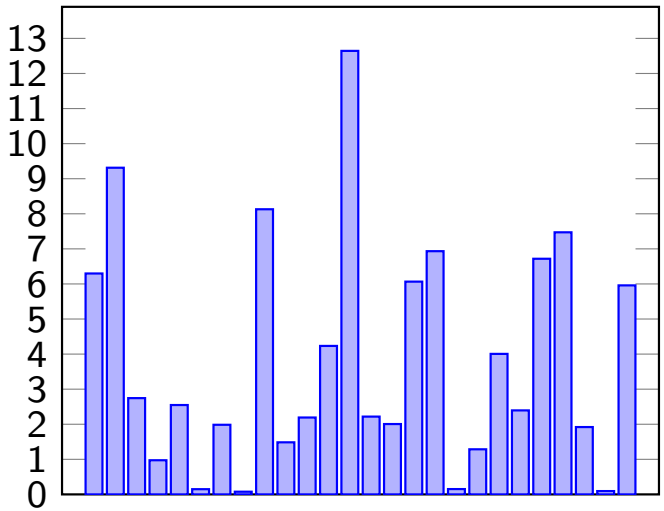
shifted letter freq. (IC=0.065)



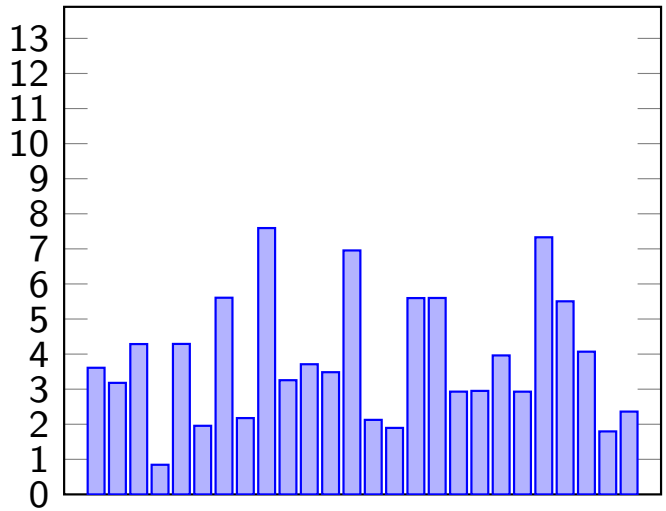
shifted letter freq. (IC=0.065)



shifted letter freq. (IC=0.065)



averaged letter freq. (IC=0.046)



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

$$\mathsf{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}, \quad \mathsf{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$$

with $\mathsf{Dec}_K(\mathsf{Enc}_K(M)) = M$ for all $K \in \mathcal{K}, M \in \mathcal{M}$, where $\mathcal{M}, \mathcal{C}, \mathcal{K}$ are the sets of possible plaintexts, ciphertexts and keys, respectively.

Let also $M \in \mathcal{M}$, $C \in \mathcal{C}$ and $K \in \mathcal{K}$ be values of plaintext, ciphertext and key. Let $\mathbb{P}(M)$ and $\mathbb{P}(K)$ 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

$$\mathbb{P}(C) = \sum_{K \in \mathcal{K}} \mathbb{P}(K) \cdot \mathbb{P}(\mathsf{Dec}_K(C)).$$

and can also determine the conditional probability

$$\mathbb{P}(C|M) = \sum_{\{K \in \mathcal{K} | M = \mathsf{Dec}_K(C)\}} \mathbb{P}(K)$$

Perfect secrecy III

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

$$\mathbb{P}(M|C) = \frac{\mathbb{P}(M) \cdot \mathbb{P}(C|M)}{\mathbb{P}(C)} = \frac{\mathbb{P}(M) \cdot \sum_{\{K|M=\mathsf{Dec}_K(C)\}} \mathbb{P}(K)}{\sum_K \mathbb{P}(K) \cdot \mathbb{P}(\mathsf{Dec}_K(C))}.$$

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 $\mathbb{P}(C) > 0$ we have

$$\mathbb{P}(M|C) = \mathbb{P}(M).$$

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

C.E. Shannon: Communication theory of secrecy systems. Bell System Technical Journal, Vol 28, Oct 1949, pp 656–715. `https://archive.org/details/bstj28-4-656/page/n55/mode/2up`

Vernam cipher / one-time pad I

Shannon's theorem:

Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ 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 $C = \mathsf{Enc}_K M$.

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

One-time pad

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

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

Example:
`0xbd4b083f6aae` $\oplus$ "`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 ($\mathbb{P}(y = 0) = \mathbb{P}(y = 1) = \frac{1}{2}$), 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 $M = \mathsf{Dec}_K(C)$. 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 duplicate one-time pads, leading to the success of the US decryption project VENONA.
`https://www.nsa.gov/portals/75/documents/news-features/declassified-documents/crypto-almanac-50th/VENONA_An_Overview.pdf`

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

A non-secure pseudo-random number generator

Example (insecure)

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

$$R_{i+1} = (aR_i + b) \bmod m$$

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 $\mathbb{Z}_m$:

$$\begin{aligned} R_2 &\equiv aR_1 + b \pmod m \\ R_3 &\equiv aR_2 + b \pmod m \end{aligned}$$

Solution:

$$\begin{aligned} a &\equiv (R_2 - R_3)/(R_1 - R_2) \pmod m \\ b &\equiv R_2 - R_1(R_2 - R_3)/(R_1 - R_2) \pmod m \end{aligned}$$

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

Private-key (symmetric) encryption

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

- the **key generation algorithm** Gen receives a security parameter ℓ and outputs a key $K \leftarrow \mathsf{Gen}(1^\ell)$, with $K \in \mathcal{K}$, key length $|K| \geq \ell$;
- the **encryption algorithm** Enc maps a key K and a plaintext message $M \in \mathcal{M} = \{0,1\}^m$ to a ciphertext message $C \leftarrow \mathsf{Enc}_K(M)$;
- the **decryption algorithm** Dec maps a key K and a ciphertext $C \in \mathcal{C} = \{0,1\}^n$ $(n \geq m)$ to a plaintext message $M := \mathsf{Dec}_K(C)$;
- for all ℓ, $K \leftarrow \mathsf{Gen}(1^\ell)$, and $M \in \{0,1\}^m$: $\mathsf{Dec}_K(\mathsf{Enc}_K(M)) = M$.

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 ℓ bits long. (think Turing machine)

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

Negligible functions

We call a function $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$ "negligible" if, as its input n increases, its output converges to zero faster than the inverse of any polynomial.

Negligible functions appear in formal security definitions for encryption schemes to limit the allowed success probability of an adversary as a function of some security parameter, e.g. a key length.

Definition: negligible function

A function f from the natural numbers to the non-negative real numbers is "negligible" if for every polynomial p there is an integer N such that for all $n > N$ it holds that $f(n) < \frac{1}{p(n)}$.

Equivalently: for all positive integers c there exists an N_c such that for all $n > N_c$ it holds that $f(n) < n^{-c}$.

Examples of negligible functions: $n \mapsto 2^{-n}$, $n \mapsto 2^{-\sqrt{n}}$, $n \mapsto n^{-\log n}$.

For example: if $f(n) = 2^{-\sqrt{n}}$ and $p(n) = n^5$ then solving $f(n) < 1/p(n)$ we get $n > 25 \log_2^2 n$, which holds for all $n > 3453$, i.e. $N_5 = 3453$.

We normally will write a negligible function as "negl".

Closure properties: If negl and negl2 are negligible, then so are $n \mapsto \mathsf{negl}(n) + \mathsf{negl2}(n)$ and $n \mapsto \mathsf{negl}(n) \cdot p(n)$ for any polynomial $p(n)$.

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 $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$
- an adversary $\mathcal{A}$, who tries to demonstrate a weakness in Π.

Most of these games follow a simple pattern:

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

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

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

Probabilistic security game X: $$X_{\mathcal{A},\Pi}(\ell) = \begin{cases} 1, & b = b' \\ 0, & b \neq b' \end{cases}$$

Security definition

An encryption scheme Π is considered "X secure" if for all probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$ there exists a "negligible" function negl such that

$$\mathbb{P}(X_{\mathcal{A},\Pi}(\ell) = 1) < \frac{1}{2} + \mathsf{negl}(\ell).$$

In practice: We want $\mathsf{negl}(\ell)$ to drop below a small number (e.g., 2^{-80} or 2^{-100}) for modest key lengths ℓ (e.g., $\log_{10} \ell \approx 2 \ldots 3$). 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.

Negligible advantage: Some authors prefer an alternative definition where $\mathcal{A}$'s ability to guess b is quantified instead as

$$\mathsf{Adv}_{X_{\mathcal{A},\Pi}(\ell)} = \left|\mathbb{P}(b = 1 \text{ and } b' = 1) - \mathbb{P}(b = 0 \text{ and } b' = 1)\right| < \mathsf{negl}(\ell).$$

“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 feasiblity: 2^{80} iterations
(i.e., $< 2^{60}$ feasible with effort, but $> 2^{100}$ certainly not)

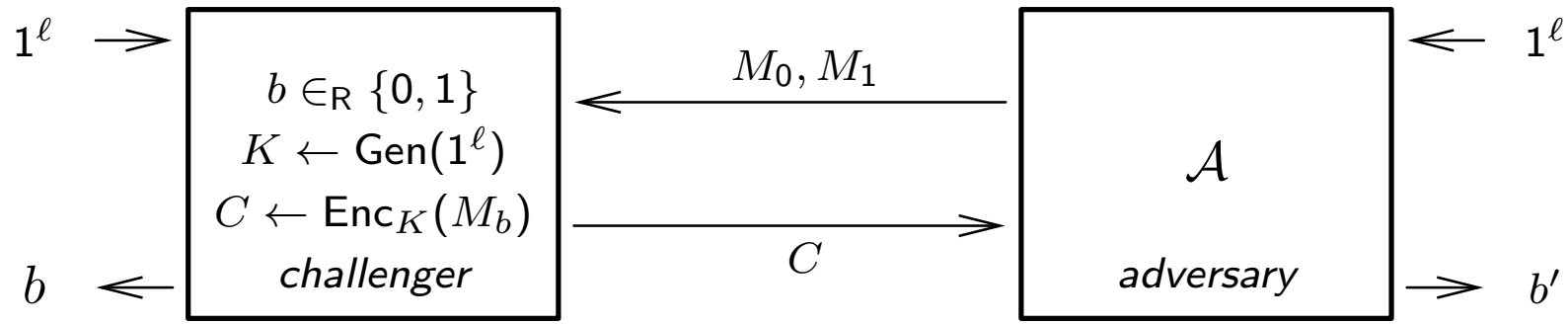
For comparison:

- The fastest key search effort using thousands of Internet PCs (RC5-64, 2002) achieved in the order of 2^{37} keys per second.
  `https://www.cl.cam.ac.uk/~rnc1/brute.html`
  `https://www.distributed.net/`
- Since January 2018, the Bitcoin network has been searching through about $10^{19} \approx 2^{63}$ cryptographic hash values per second, mostly using ASICs.
  `https://bitcoin.sipa.be/`

Indistinguishability in the presence of an eavesdropper

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

Experiment/game $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(\ell)$:



Setup:

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

Rules for the interaction:

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

Finally, $\mathcal{A}$ outputs b'. If $b' = b$ then $\mathcal{A}$ has succeeded $\Rightarrow \mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(\ell) = 1$

Indistinguishability in the presence of an eavesdropper

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

$$\mathbb{P}(\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(\ell) = 1) \leq \frac{1}{2} + \mathsf{negl}(\ell)$$

In other words: as we increase the security parameter ℓ, 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 negl such that

$$|\mathbb{P}(D(r) = 1) - \mathbb{P}(D(G(s)) = 1)| \leq \mathsf{negl}(n)$$

where both $r \in_R \{0,1\}^{e(n)}$ and the seed $s \in_R \{0,1\}^n$ 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

$$\mathbb{P}(D(G(s)) = 1) = 1$$
$$\mathbb{P}(D(r) = 1) = \frac{2^n}{2^{e(n)}}$$

the difference of which converges to 1, which is not negligible.

But a brute-force distinguisher has a exponential run-time $O(2^n)$, 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:

$\Pi_{\mathsf{PRG}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$:

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

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

Such constructions are known as "stream ciphers".

We can prove that Π_{PRG} has "indistinguishable encryption in the presence of an eavesdropper" assuming that G is a pseudo-random generator: if we had a polynomial-time adversary $\mathcal{A}$ that can succeed with non-negligible advantage against Π_{PRG}, we can turn that using a polynomial-time algorithm into a polynomial-time distinguisher for G, which would violate the assumption.

Security proof for a stream cipher

Claim: Π_{PRG} has indistinguishability in the presence of an eavesdropper if G is a pseudo-random generator.

Proof: (outline) If Π_{PRG} did not have indistinguishability in the presence of an eavesdropper, there would be an adversary $\mathcal{A}$ for which

$$\epsilon(\ell) := \mathbb{P}(\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi_{\mathsf{PRG}}}(\ell) = 1) - \frac{1}{2}$$

is not negligible.

Use that $\mathcal{A}$ to construct a distinguisher D for G:

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

Now, what is $|\mathbb{P}(D(r) = 1) - \mathbb{P}(D(G(K)) = 1)|$?

Security proof for a stream cipher (cont'd)

What is $|\mathbb{P}(D(r) = 1) - \mathbb{P}(D(G(K)) = 1)|$?

- What is $\mathbb{P}(D(r) = 1)$?
  Let $\tilde{\Pi}$ be an instance of the one-time pad, with key and message length $e(\ell)$, i.e. compatible to Π_{PRG}. In the $D(r)$ case, where we feed it a random string $r \in_{\mathsf{R}} \{0,1\}^{e(n)}$, then from the point of view of $\mathcal{A}$ being called as a subroutine of $D(r)$, it is confronted with a one-time pad $\tilde{\Pi}$. The perfect secrecy of $\tilde{\Pi}$ implies $\mathbb{P}(D(r) = 1) = \frac{1}{2}$.
- What is $\mathbb{P}(D(G(K)) = 1)$?
  In this case, $\mathcal{A}$ participates in the game $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi_{\mathsf{PRG}}}(\ell)$. Thus we have $\mathbb{P}(D(G(K)) = 1) = \mathbb{P}(\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi_{\mathsf{PRG}}}(\ell) = 1) = \frac{1}{2} + \epsilon(\ell)$.

Therefore

$$|\mathbb{P}(D(r) = 1) - \mathbb{P}(D(G(K)) = 1)| = \epsilon(\ell)$$

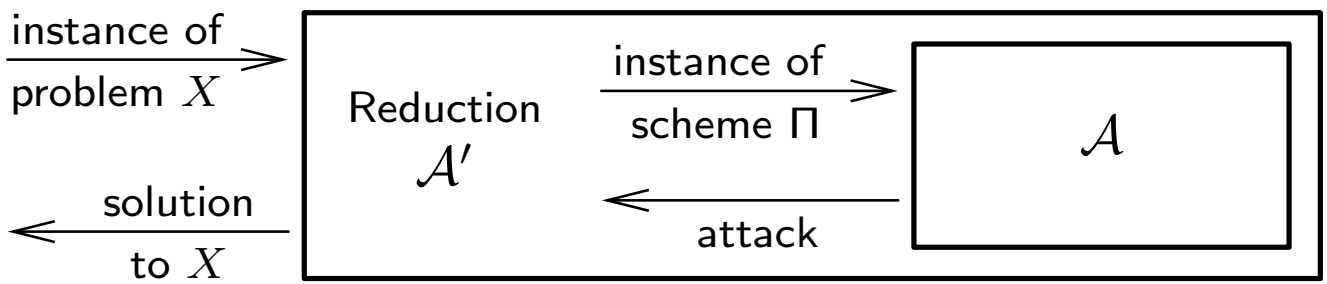
which we have assumed not to be negligible, which implies that G is not a pseudo-random generator, contradicting the assumption. □

Katz/Lindell (1st ed.), pp 73-75

Security proofs through reduction

Some key points about this style of "security proof":

- We have *not* shown that the encryption scheme Π_{PRG} is "secure". (We don't know how to do this!)
- We have shown that Π_{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 Π_{PRG} to another problem X:



Here: X = distinguishing output of G from random string

- We have shown how to turn any successful attack on Π_{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

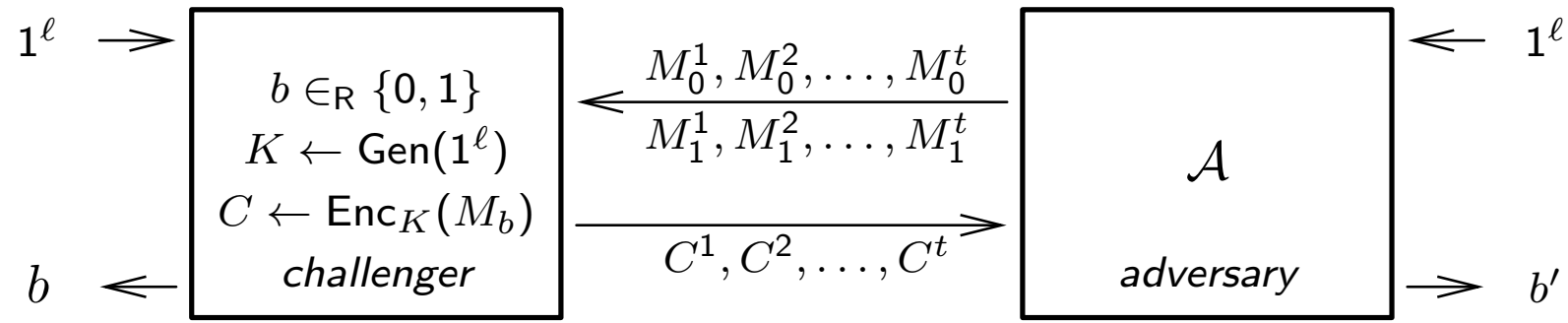
In the end, the provable security of some cryptographic construct (e.g., Π_{PRG}, some mode of operation, some security protocol) boils down to these questions:

- 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 $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, $\mathcal{M} = \{0,1\}^m$, security parameter ℓ.

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



Setup:

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

Rules for the interaction:

1. The adversary $\mathcal{A}$ 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 $M_j^i \in \{0,1\}^m$.
2. The challenger computes $C^i \leftarrow \mathsf{Enc}_K(M_b^i)$ and returns $C^1, C^2, \ldots, C^t$ to $\mathcal{A}$

Finally, $\mathcal{A}$ outputs b'. If $b' = b$ then $\mathcal{A}$ has succeeded $\Rightarrow \mathsf{PrivK}^{\mathsf{mult}}_{\mathcal{A},\Pi}(\ell) = 1$

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 $\mathcal{A}$ there exists a negligible function negl, such that

$$\mathbb{P}(\mathsf{PrivK}^{\mathsf{mult}}_{\mathcal{A},\Pi}(\ell) = 1) \leq \frac{1}{2} + \mathsf{negl}(\ell)$$

Same definition as for *indistinguishable encryptions in the presence of an eavesdropper*, except for referring to the multi-message eavesdropping experiment $\mathsf{PrivK}^{\mathsf{mult}}_{\mathcal{A},\Pi}(\ell)$.

Example: Does our stream cipher Π_{PRG} offer indistinguishable *multiple* encryptions in the presence of an eavesdropper?

Adversary $\mathcal{A}_4$ outputs four messages , and returns $b' = 1$ iff . $\mathbb{P}(\mathsf{PrivK}^{\mathsf{mult}}_{\mathcal{A}_4,\Pi_{\mathsf{PRG}}}(\ell) = 1) =$

Actually: Any encryption scheme is going to fail here!

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:

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

$\|$ 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

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.

$$\text{for each } i: \qquad IV_i \in_{\mathsf{R}} \{0,1\}^n, \quad C_i := (IV_i, G(K, IV_i) \oplus M_i)$$

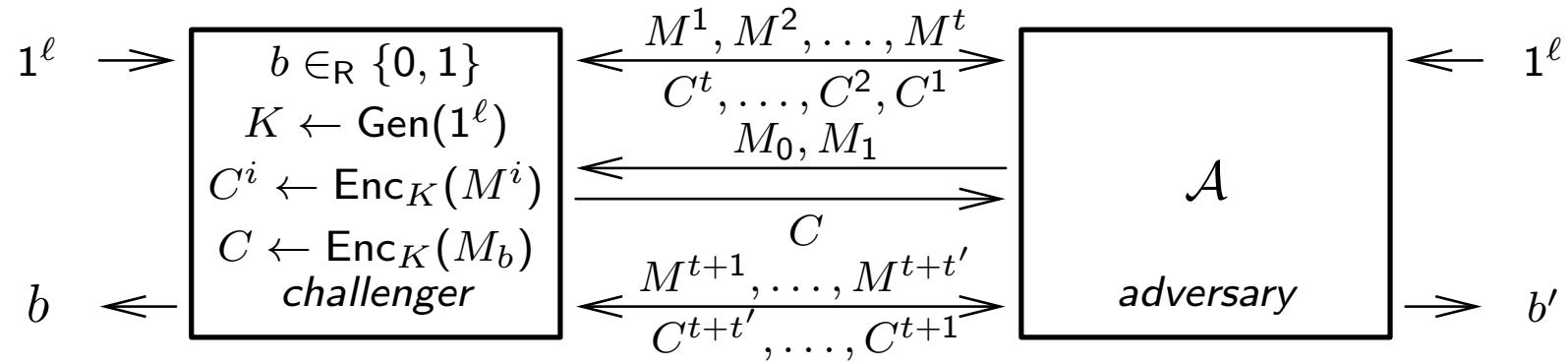
But: what exact security properties do we expect of a G with IV input?

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

Security against chosen-plaintext attacks (CPA)

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

Experiment/game $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(\ell)$:



Setup: (as before)

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

Rules for the interaction:

1. The adversary $\mathcal{A}$ is given oracle access to Enc_K:
   $\mathcal{A}$ outputs M^1, gets $\mathsf{Enc}_K(M^1)$, outputs M^2, gets $\mathsf{Enc}_K(M^2)$, ...
2. The adversary $\mathcal{A}$ outputs a pair of messages: $M_0, M_1 \in \{0,1\}^m$.
3. The challenger computes $C \leftarrow \mathsf{Enc}_K(M_b)$ and returns C to $\mathcal{A}$
4. The adversary $\mathcal{A}$ continues to have oracle access to Enc_K.

Finally, $\mathcal{A}$ outputs b'. If $b' = b$ then $\mathcal{A}$ has succeeded $\Rightarrow \mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(\ell) = 1$

Security against chosen-plaintext attacks (cont'd)

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

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

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

Random function

Consider all possible functions of the form

$$f : \{0,1\}^m \to \{0,1\}^n$$

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 *random function* f is one that we have picked uniformly at random from all these possible functions.

Random permutation

Consider all possible permutations of the form

$$g : \{0,1\}^n \leftrightarrow \{0,1\}^n$$

How many different such g are there?

An n-bit to n-bit *random permutation* g is one that we have picked uniformly at random from all these possible permutations.

Pseudo-random functions and permutations

Basic idea:

A *pseudo-random function (PRF)* is a fixed, efficiently computable function

$$F : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^n$$

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

$$F_K : \{0,1\}^m \to \{0,1\}^n$$

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 *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)

$F : \underset{\text{key}}{\{0,1\}^n} \times \underset{\text{input}}{\{0,1\}^n} \to \underset{\text{output}}{\{0,1\}^n}$ efficient, keyed, length preserving $_{|\text{input}|=|\text{output}|}$

Definition

F is a *pseudo-random function* if for all probabilistic, polynomial-time distinguishers D there exists a negligible function negl such that

$$\left|\mathbb{P}(D^{F_K(\cdot)}(1^n) = 1) - \mathbb{P}(D^{f(\cdot)}(1^n) = 1)\right| \leq \text{negl(n)}$$

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

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

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:

$\Pi_{\mathsf{PRF}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$:

Let F be a pseudo-random function.

- Gen: on input 1^ℓ choose $K \in_{\mathsf{R}} \{0,1\}^\ell$ randomly
- Enc: read $K \in \{0,1\}^\ell$ and $M \in \{0,1\}^\ell$, choose $R \in_{\mathsf{R}} \{0,1\}^\ell$ randomly, then output

$$C := (R, F_K(R) \oplus M)$$

- Dec: read $K \in \{0,1\}^\ell$, $C = (R, S) \in \{0,1\}^{2\ell}$, then output

$$M := F_K(R) \oplus S$$

Strategy for proving Π_{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 Π_{PRF}

First consider $\tilde{\Pi}$, a variant of Π_{PRF} in which the pseudo-random function F_K was replaced with a random function f. Claim:

$$\mathbb{P}(\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(\ell) = 1) \leq \frac{1}{2} + \frac{q(\ell)}{2^\ell} \qquad \text{with } q(\ell) \text{ oracle queries}$$

Recall: when the challenge ciphertext C in $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(\ell)$ is computed, the challenger picks $R_C \in_{\mathsf{R}} \{0,1\}^\ell$ and returns $C := (R_C, f(R_C) \oplus M_b)$.

Case 1: R_C is also used in one of the oracle queries. In which case $\mathcal{A}$ can easily find out $f(R_C)$ and decrypt M_b. $\mathcal{A}$ makes at most $q(\ell)$ oracle queries and there are 2^ℓ possible values of R_C, this case happens with a probability of at most $q(\ell)/2^\ell$.

Case 2: R_C is not used in any of the oracle queries. For $\mathcal{A}$ the value R_C remains completely random, $f(R_C)$ remains completely random, m_b is returned one-time pad encrypted, and $\mathcal{A}$ can only make a random guess, so in this case $\mathbb{P}(b' = b) = \frac{1}{2}$.

$$\begin{aligned}
&\mathbb{P}(\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(\ell) = 1) \\
&\quad = \mathbb{P}(\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(\ell) = 1 \wedge \text{Case 1}) + \mathbb{P}(\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(\ell) = 1 \wedge \text{Case 2}) \\
&\quad \leq \mathbb{P}(\text{Case 1}) + \mathbb{P}(\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(\ell) = 1 | \text{Case 2}) \leq \frac{q(\ell)}{2^\ell} + \frac{1}{2}.
\end{aligned}$$

Security proof for encryption scheme Π_{PRF} (cont'd)

Assume we have an attacker $\mathcal{A}$ against Π_{PRF} with non-negligible

$$\epsilon(\ell) = \mathbb{P}(\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi_{\mathsf{PRF}}}(\ell) = 1) - \frac{1}{2}$$

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

$$\mathbb{P}(\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(\ell) = 1) \leq \frac{1}{2} + \frac{q(\ell)}{2^\ell}$$

Combining those two equations we get

$$\mathbb{P}(\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi_{\mathsf{PRF}}}(\ell) = 1) - \mathbb{P}(\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(\ell) = 1) \geq \epsilon(\ell) - \frac{q(\ell)}{2^\ell}$$

which is not negligible either, allowing us to distinguish f from F_K:
Build distinguisher $D^{\mathcal{O}}$ using oracle $\mathcal{O}$ to play $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(\ell)$ with $\mathcal{A}$:

1. Run $\mathcal{A}(1^\ell)$ and for each of its oracle queries M^i pick $R^i \in_{\mathsf{R}} \{0,1\}^\ell$, then return $C^i := (R^i, \mathcal{O}(R^i) \oplus M^i)$ to $\mathcal{A}$.
2. When $\mathcal{A}$ outputs M_0, M_1, pick $b \in_{\mathsf{R}} \{0,1\}$ and $R_C \in_{\mathsf{R}} \{0,1\}^\ell$, then return $C := (R_C, \mathcal{O}(R_C) \oplus M_b)$ to $\mathcal{A}$.
3. Continue answering $\mathcal{A}$'s encryption oracle queries. When $\mathcal{A}$ outputs b', output 1 if $b' = b$, otherwise 0.

Security proof for encryption scheme Π_{PRF} (cont'd)

How effective is this D?

1. **If D's oracle is F_K:** $\mathcal{A}$ effectively plays $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi_{\mathsf{PRF}}}(\ell)$ because if K was chosen randomly, D^{F_K} behaves towards $\mathcal{A}$ just like Π_{PRF}, and therefore

$$\mathbb{P}(D^{F_K(\cdot)}(1^\ell) = 1) = \mathbb{P}(\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi_{\mathsf{PRF}}}(\ell) = 1)$$

2. **If D's oracle is f:** likewise, $\mathcal{A}$ effectively plays $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(\ell)$ and therefore

$$\mathbb{P}(D^{f(\cdot)}(1^\ell) = 1) = \mathbb{P}(\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(\ell) = 1)$$

   if $f \in_{\mathsf{R}} (\{0,1\}^\ell)^{\{0,1\}^\ell}$ is chosen uniformly at random.

All combined the difference

$$\mathbb{P}(D^{F_K(\cdot)}(1^\ell) = 1) - \mathbb{P}(D^{f(\cdot)}(1^\ell) = 1) \geq \epsilon(\ell) - \frac{q(\ell)}{2^\ell}$$

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

Katz/Lindell (1st ed.), pp 90–93

Pseudo-random permutation

$F : \underset{\text{key}}{\{0,1\}^n} \times \underset{\text{input}}{\{0,1\}^n} \to \underset{\text{output}}{\{0,1\}^n}$ efficient, keyed, length preserving $_{|\text{input}|=|\text{output}|}$

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 negl such that

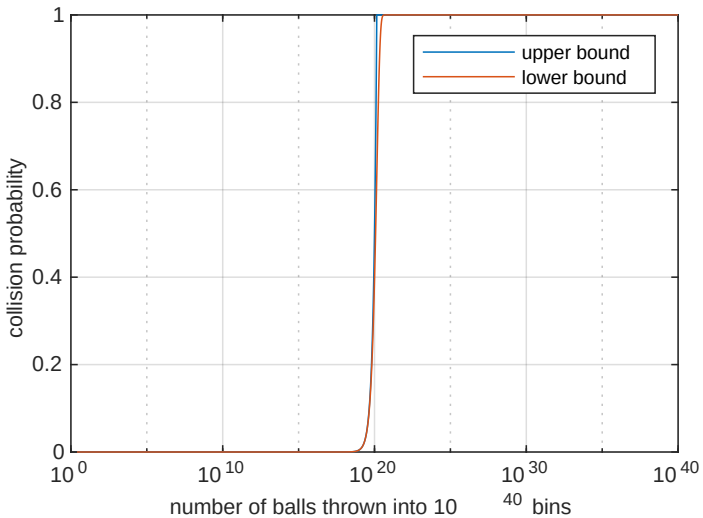
$$\left|\mathbb{P}(D^{F_K(\cdot),F_K^{-1}(\cdot)}(1^n) = 1) - \mathbb{P}(D^{f(\cdot),f^{-1}(\cdot)}(1^n) = 1)\right| \leq \text{negl(n)}$$

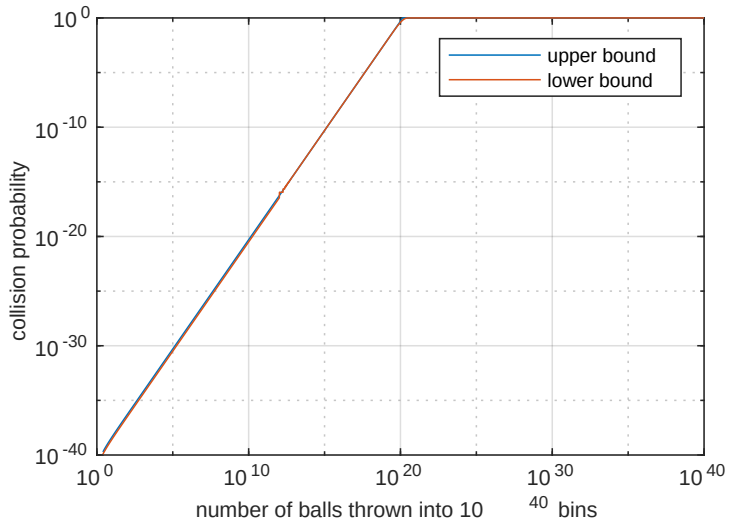
where $K \in_R \{0,1\}^n$ is chosen uniformly at random, and f is chosen uniformly at random from the set of permutations on n-bit strings.

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?





Remember: for large n the collision probability

- is near 1 for $b \gg \sqrt{n}$
- is near 0 for $b \ll \sqrt{n}$, growing roughly proportional to $\frac{b^2}{n}$

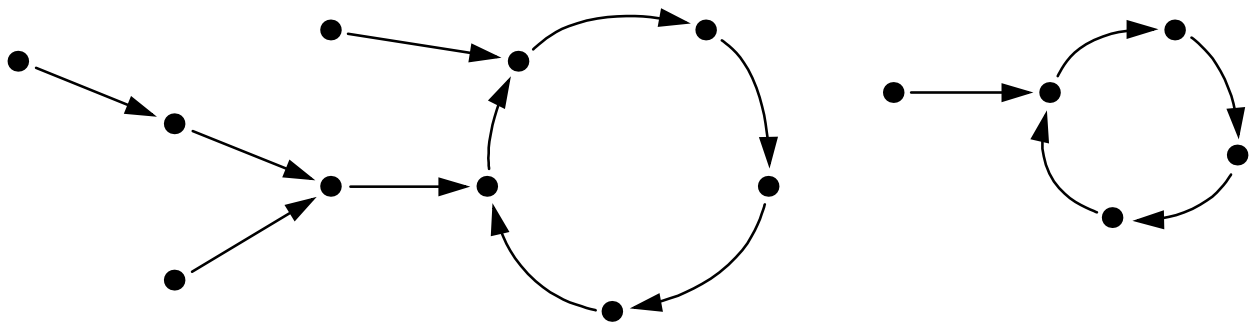
Expected number of balls thrown before first collision: $\sqrt{\frac{\pi}{2}n}$ (for $n \to \infty$)

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

Iterating a random function

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

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



Several components, each a directed cycle and trees attached to it.

Some expected values for $n \to \infty$, random $u \in_R \{1, \ldots, n\}$:

- tail length $\mathbb{E}(t(u)) = \sqrt{\pi n/8}$ $f^{t(u)}(u) = f^{t(u)+c(u)\cdot i}(u)$, $\forall i \in \mathbb{N}$,
- cycle length $\mathbb{E}(c(u)) = \sqrt{\pi n/8}$ where $t(u), c(u)$ minimal
- rho-length $\mathbb{E}(t(u) + c(u)) = \sqrt{\pi n/2}$
- predecessors $\mathbb{E}(|\{v | f^i(v) = u \wedge i > 0\}|) = \sqrt{\pi n/8}$
- edges of component containing u: $2n/3$

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

Menezes/van Oorschot/Vanstone, §2.1.6. Knuth: TAOCP, §1.3.3, exercise 17.
Flajolet/Odlyzko: Random mapping statistics, EUROCRYPT'89, LNCS 434.

Block ciphers

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

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

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 $f_{K,i} : \{0,1\}^{\frac{n}{2}} \to \{0,1\}^{\frac{n}{2}}$ (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:

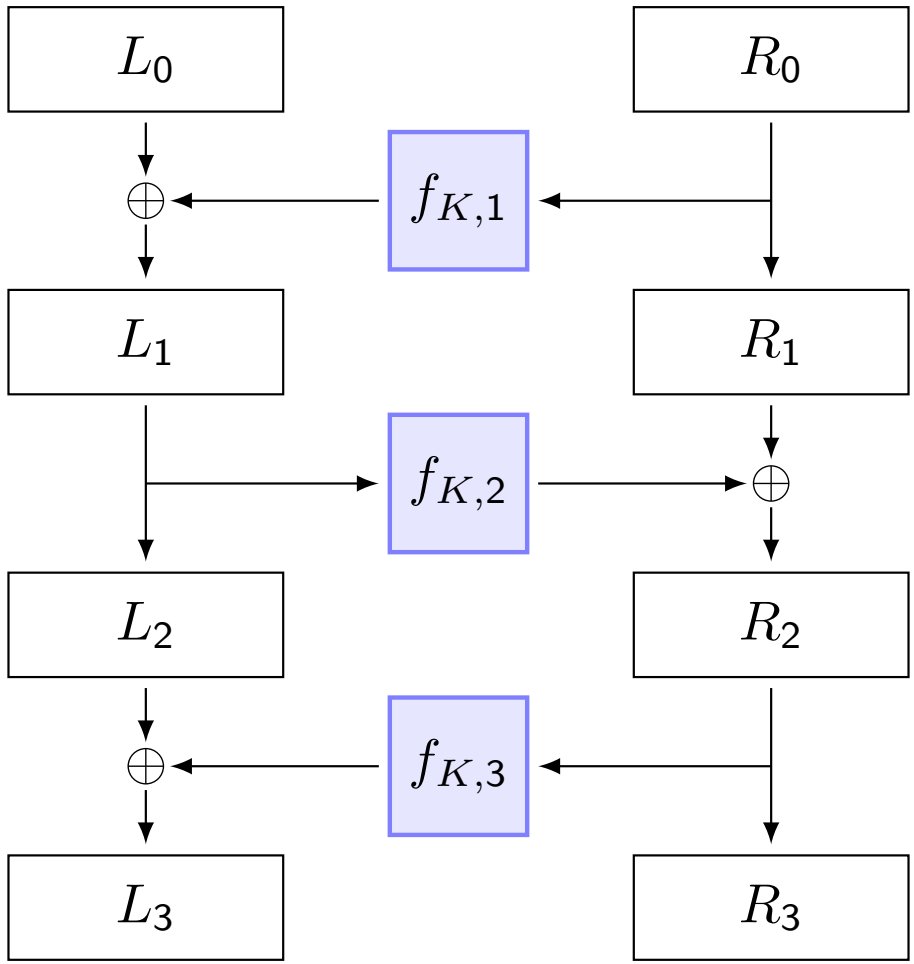
$$\begin{aligned} L_i &= L_{i-1} \oplus f_{K,i}(R_{i-1}) \quad \text{and} \quad R_i = R_{i-1} \qquad \text{for odd } i \\ R_i &= R_{i-1} \oplus f_{K,i}(L_{i-1}) \quad \text{and} \quad L_i = L_{i-1} \qquad \text{for even } i \end{aligned}$$

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

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

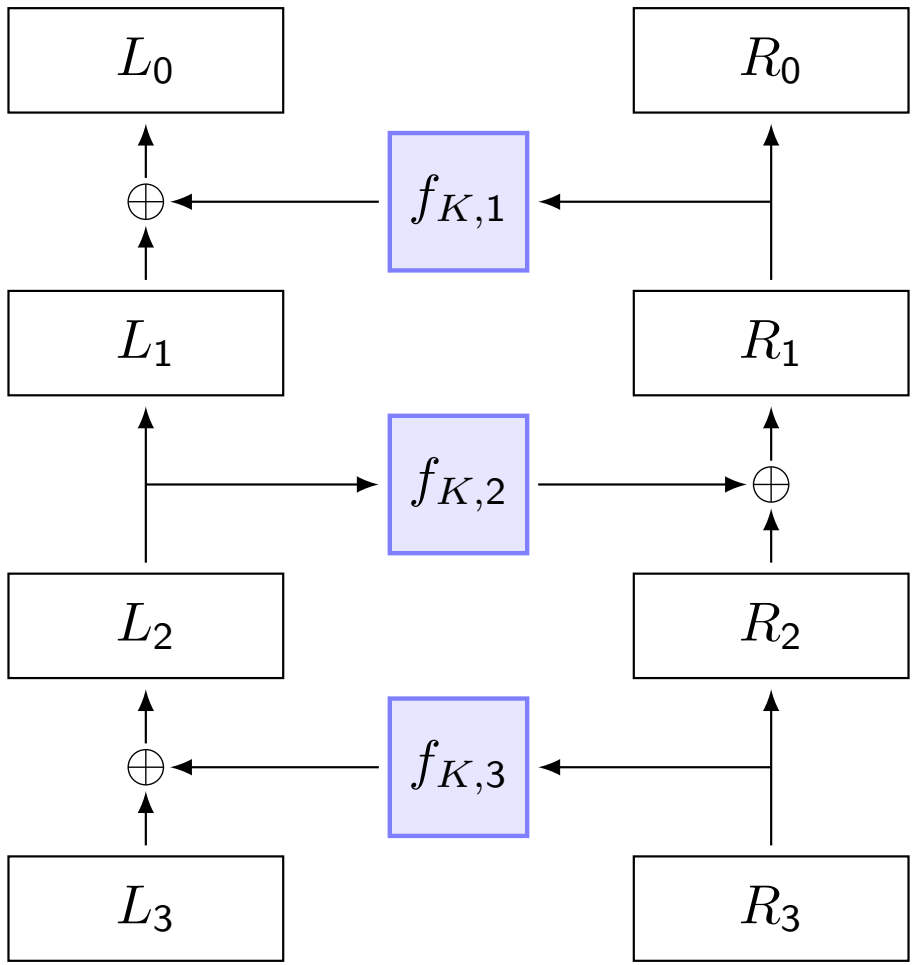
Feistel structure II

$r = 3$ rounds:



Feistel structure III

Decryption:



Feistel structure IV

Decryption works backwards ($i = r, \ldots, 1$), undoing round after round, starting from the ciphertext:

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

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, `https://link.springer.com/chapter/10.1007/3-540-39799-X_34`

Data Encryption Standard (DES)

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

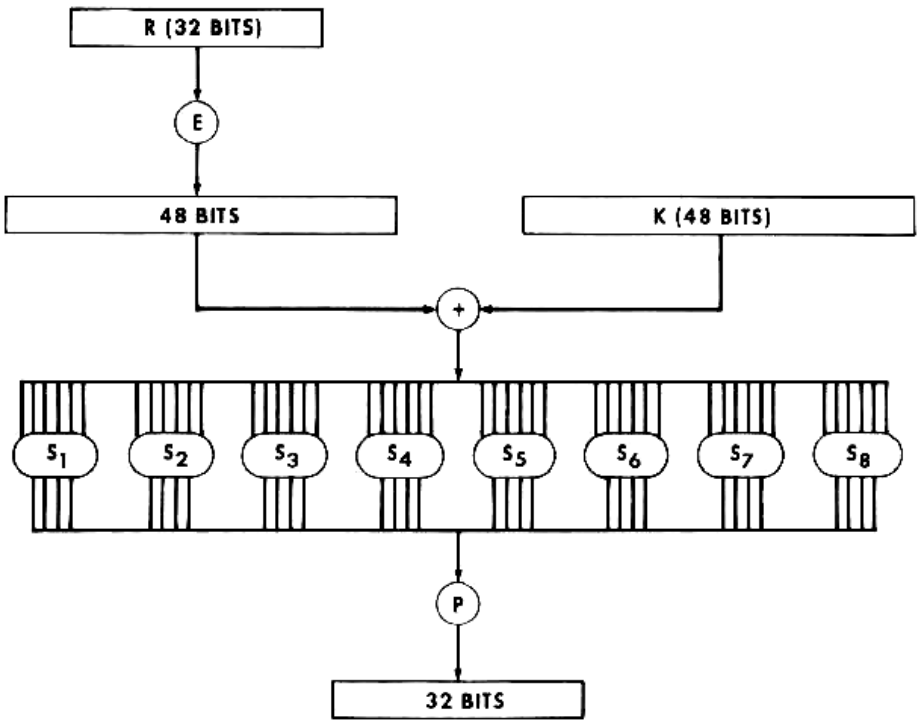
DES has a block size of 64 bits and a key size of 56 bits. The relatively short key size and its limited protection against brute-force key searches immediately triggered criticism, but this did not prevent DES from becoming the most commonly used cipher for banking networks and numerous other applications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

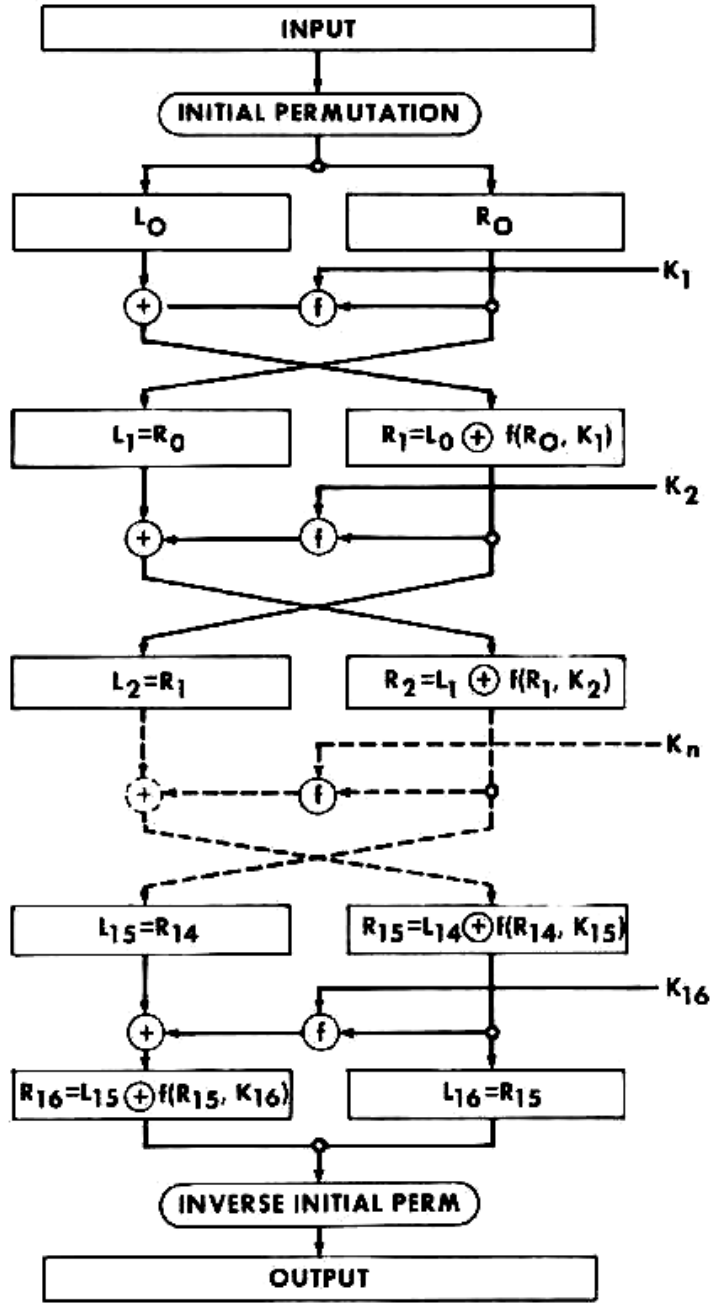
DES uses a 16-round Feistel structure. Its round function f is much simpler than a good pseudo-random function, but the number of iterations increases the complexity of the resulting permutation sufficiently.

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.

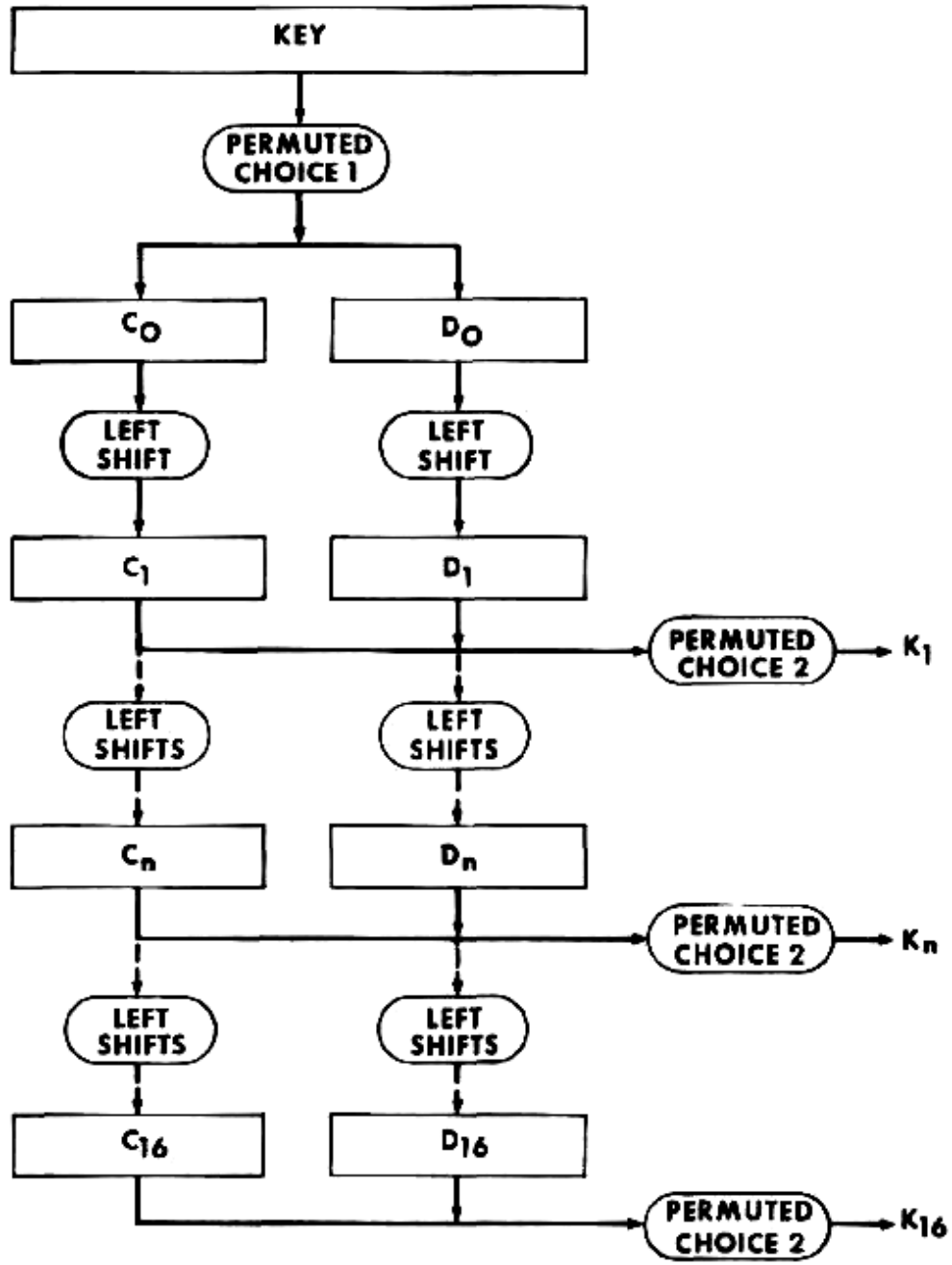
`https://csrc.nist.gov/files/pubs/fips/46-3/final/docs/fips46-3.pdf`

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.



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:

$$\begin{aligned} \text{TDES}_K(M) &= \text{DES}_{K_3}(\text{DES}^{-1}_{K_2}(\text{DES}_{K_1}(M))) \\ \text{TDES}^{-1}_K(C) &= \text{DES}^{-1}_{K_1}(\text{DES}_{K_2}(\text{DES}^{-1}_{K_3}(C))) \end{aligned}$$

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 4×4 matrix in $\mathbb{F}_{2^8}$
- 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.

`https://csrc.nist.gov/pubs/fips/197/final`

Recent CPUs with AES hardware support: Intel/AMD x86 AES-NI instructions, VIA PadLock.

AES round

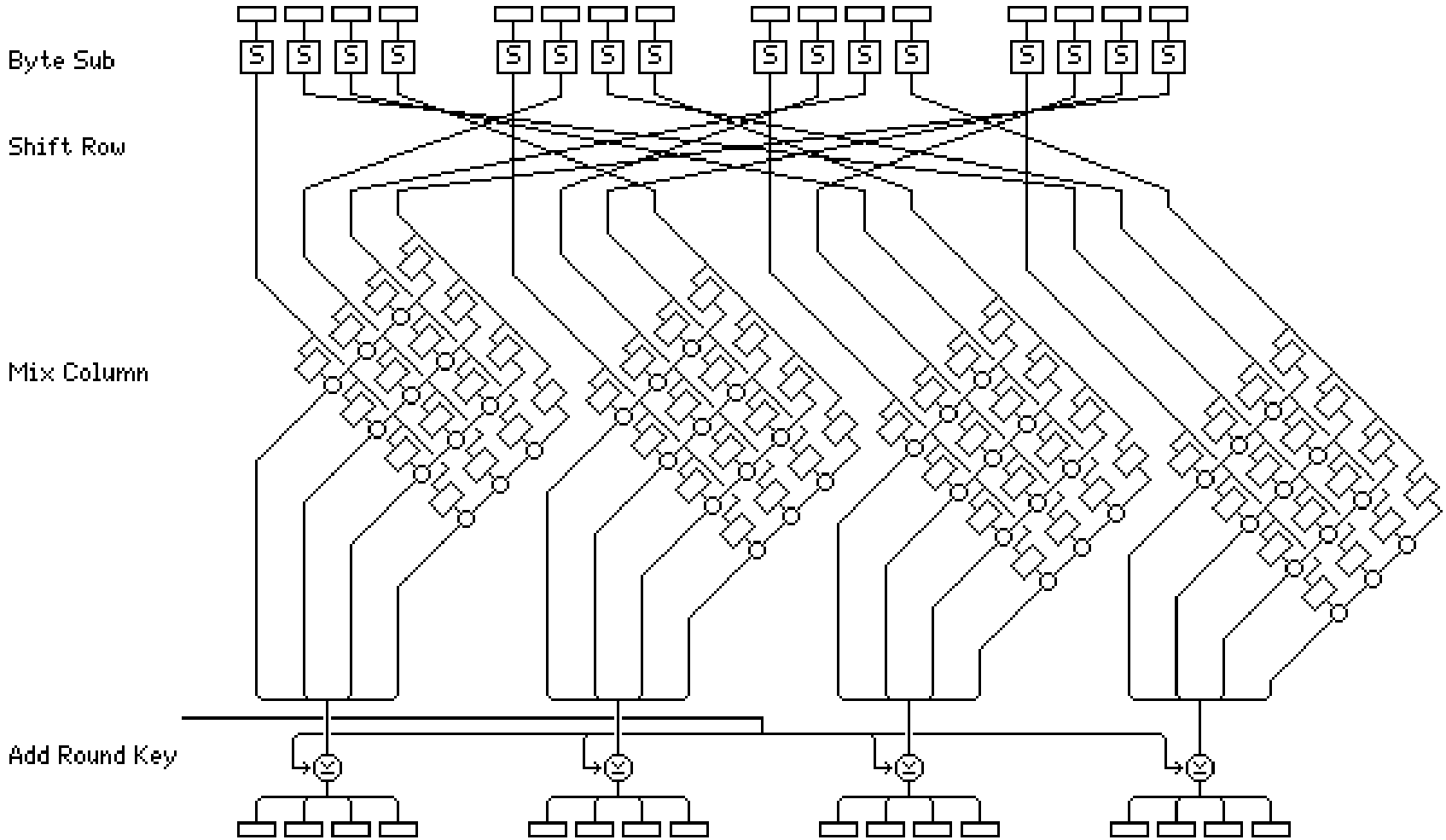


Illustration by John Savard, `http://www.quadibloc.com/crypto/co040401.htm`

Electronic Code Book (ECB) I

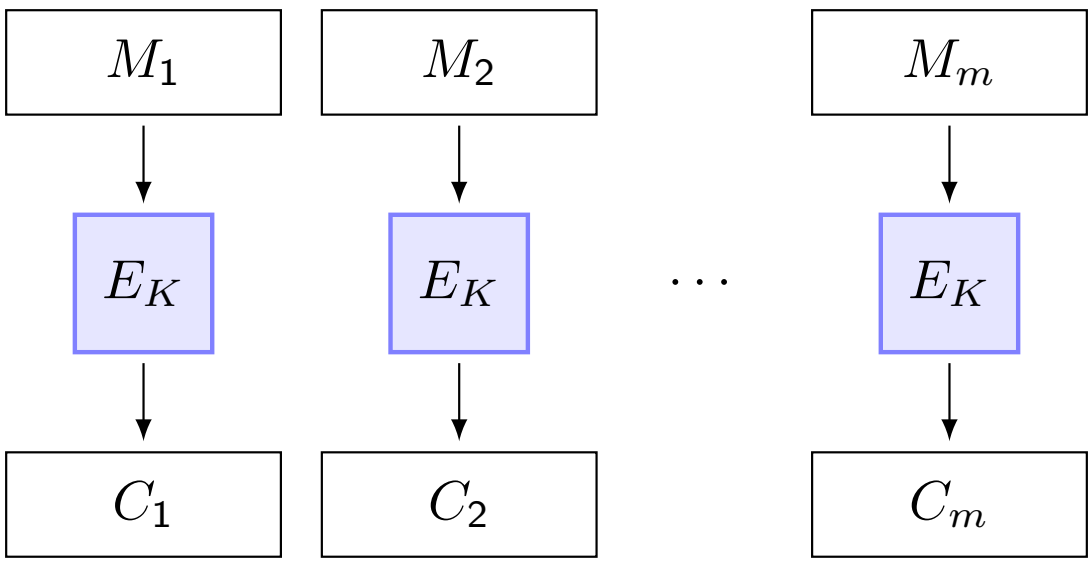
ECB is the simplest **mode of operation** for block ciphers (DES, AES).

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:

$$C_i = E_K(M_i) \qquad i = 1, \ldots, m$$
$$C = C_1 \| C_2 \| \ldots \| C_m$$



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:

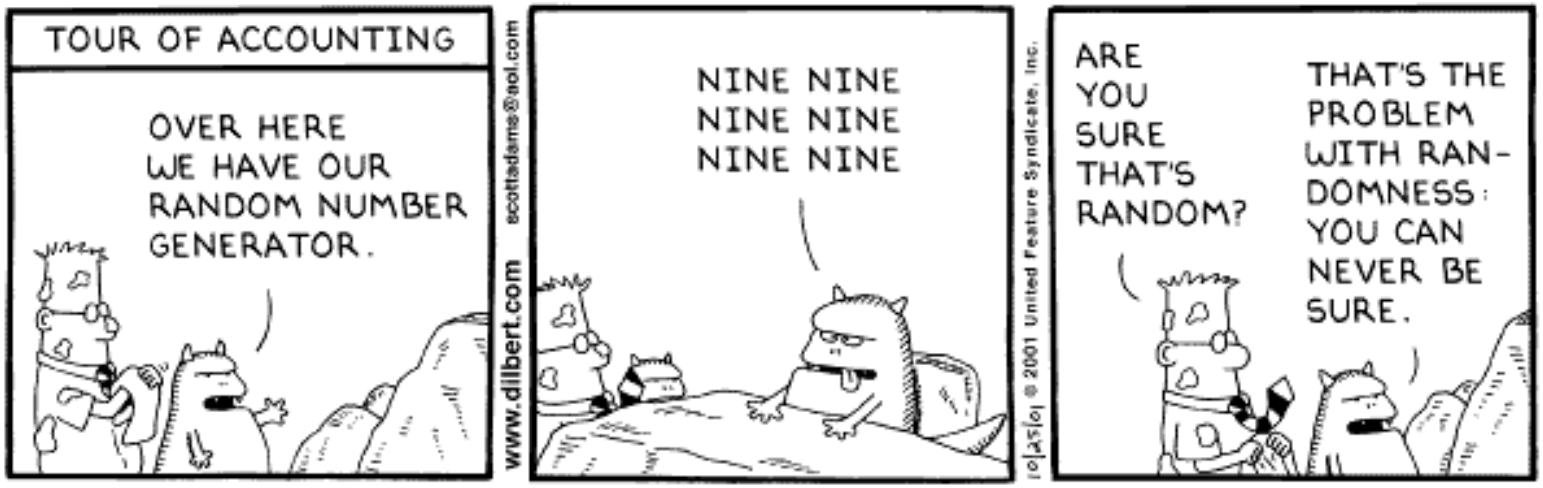
Plaintext block values are often not uniformly distributed, for example in ASCII encoded English text, some bits have almost fixed values.

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.

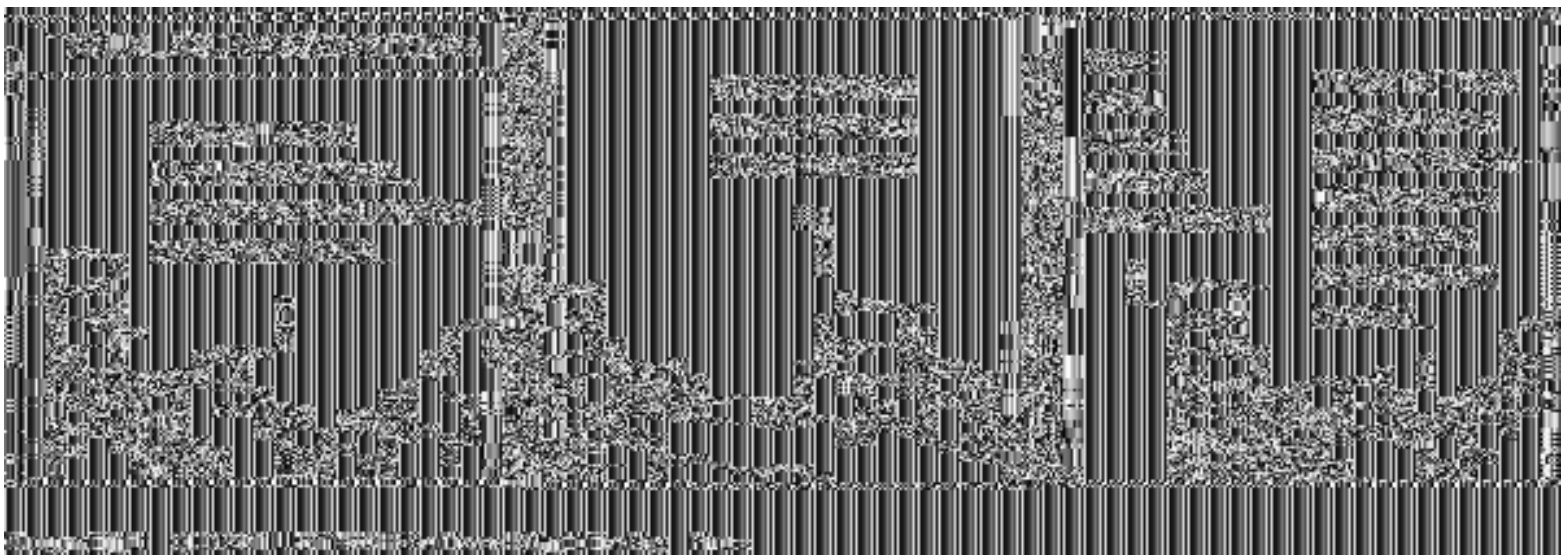
`https://csrc.nist.gov/pubs/sp/800/38/a/final`

Electronic Code Book (ECB) III

Plain-text bitmap:



DES-ECB encrypted:



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:

$$\begin{aligned} \text{Enc} &: \{0,1\}^k \times \{0,1\}^r \times \{0,1\}^l \to \{0,1\}^m \\ \text{Dec} &: \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^l \end{aligned}$$

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 $M \| \text{pad}(M) = M_1 \| M_2 \| \ldots \| M_n$ by applying Π_{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 blockciphers 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:

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

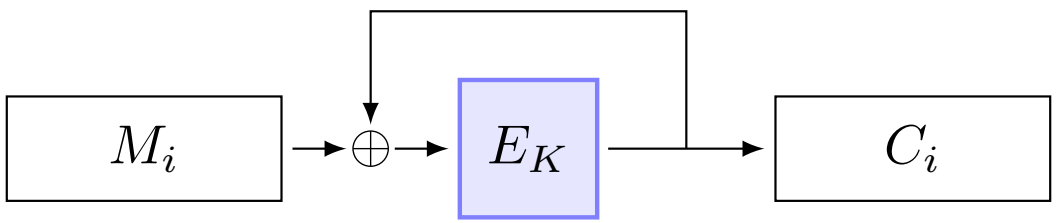
2. Generate a random, unpredictable n-bit *initial vector* (IV) C_0.
3. Starting with C_0, XOR the previous ciphertext block into the plaintext block before applying the block cipher:

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

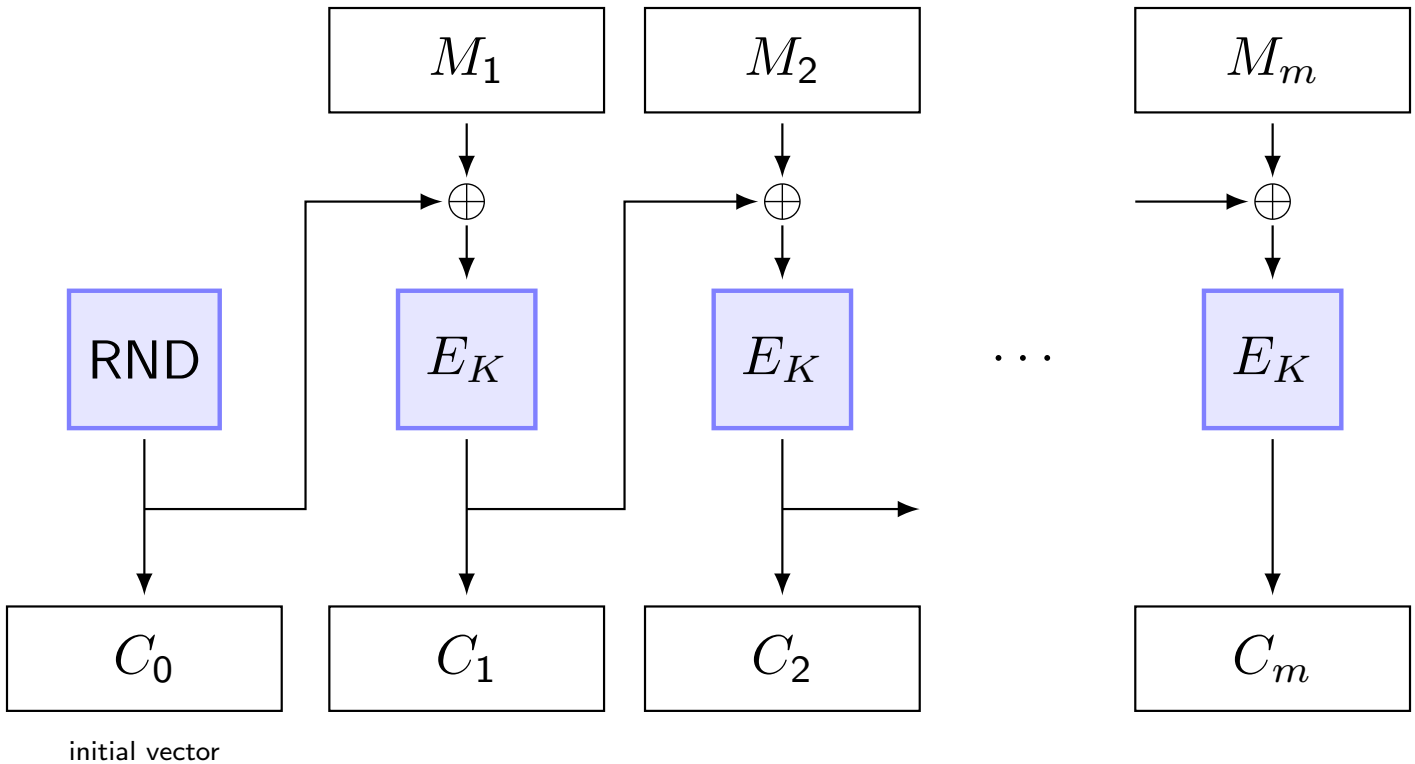
4. Output the $(m+1) \times n$-bit cipher text

$$C = C_0 \| C_1 \| \ldots \| C_m$$

(which starts with the random initial vector)



Cipher Block Chaining (CBC) II

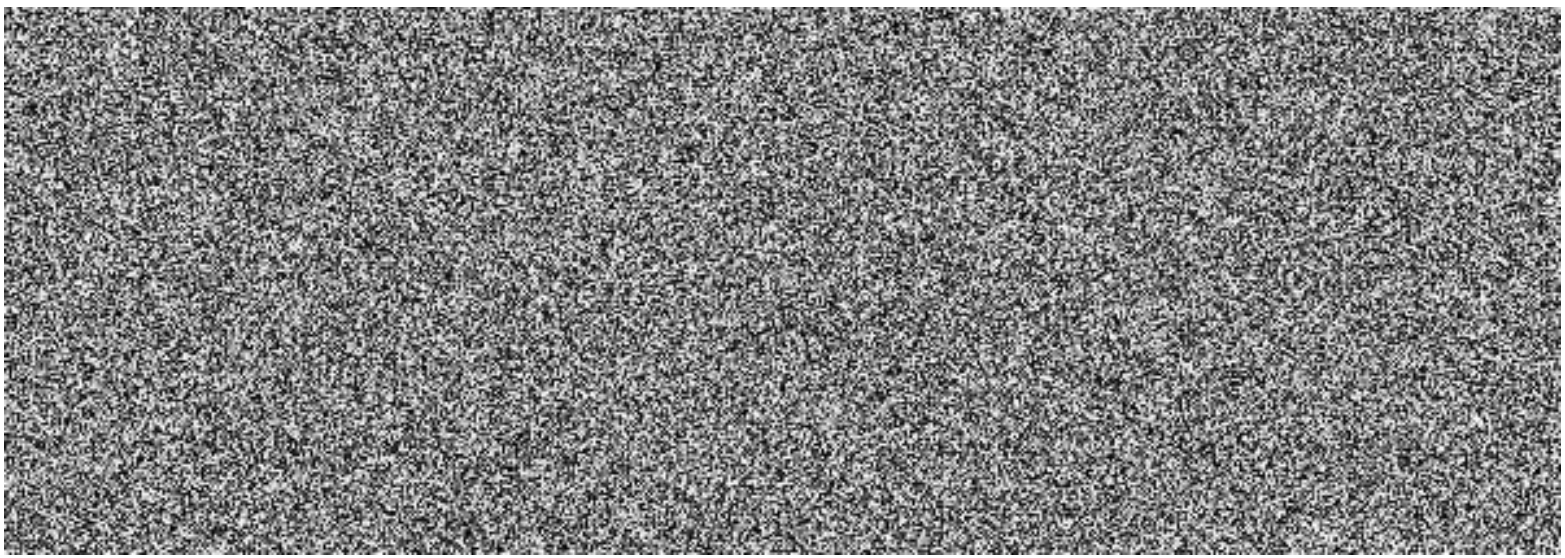


The input of the block cipher E_K is now uniformly distributed.

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

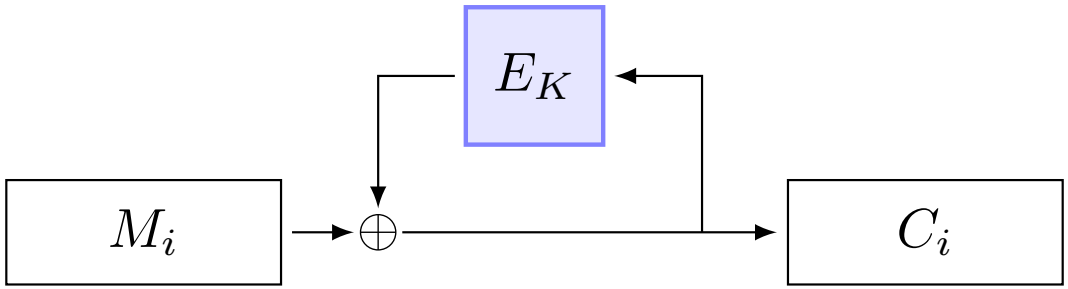
Plain-text bitmap:

DES-CBC encrypted:



Cipher Feedback Mode (CFB)

$$C_i = M_i \oplus E_K(C_{i-1})$$



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)

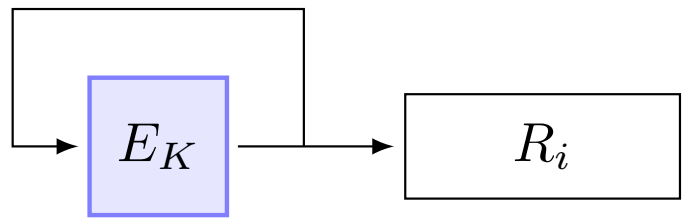
Output Feedback Mode is a stream cipher seeded by the initial vector:

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

$$R_i = E_K(R_{i-1})$$
$$C_i = M_i \oplus R_i$$



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 $2^{\frac{n}{2}}$ 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:

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

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

Here $\langle i \rangle_n$ shall mean "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

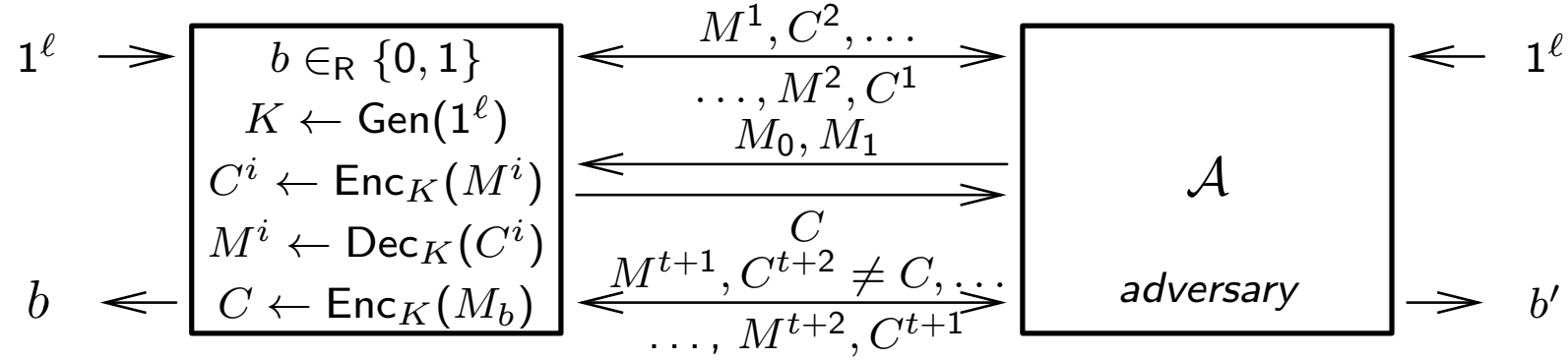
Today, Counter Mode is generally preferred over CBC, CFB, and OFB.

Alternatively, the T_i can also be generated by a maximum-length linear-feedback shift register (replacing the operation $O + i$ in $\mathbb{Z}_{2^n}$ with $O(x) \cdot x^i$ in $\mathbb{F}_{2^n}$ to avoid slow carry bits).

Security against chosen-ciphertext attacks (CCA)

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

Experiment/game $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(\ell)$:



Setup:

- handling of ℓ, b, K as before

Rules for the interaction:

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

Finally, $\mathcal{A}$ outputs b'. If $b' = b$ then $\mathcal{A}$ has succeeded $\Rightarrow \mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(\ell) = 1$

Malleability

We call an encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ **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:

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

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 electonic 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.

Message authentication code (MAC)

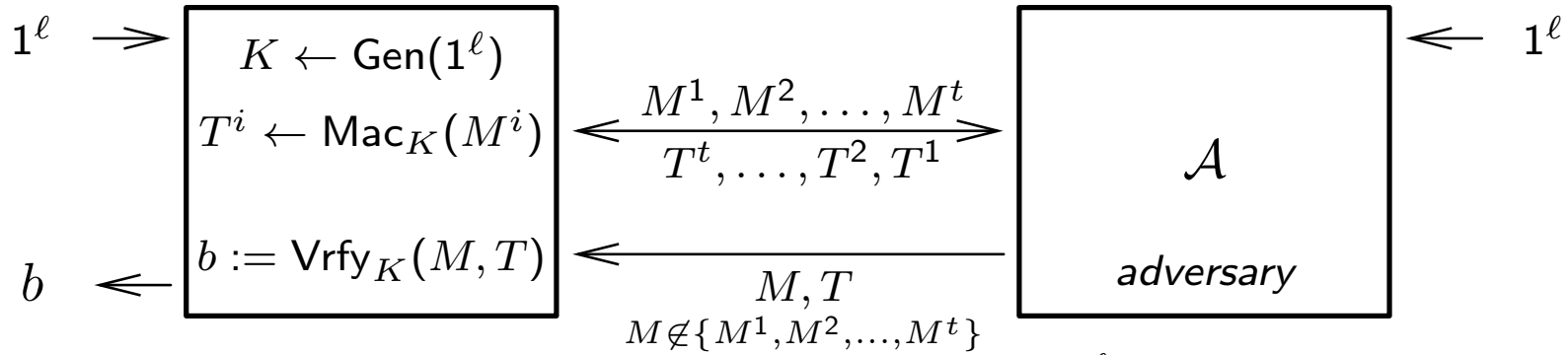
A **message authentication code** is a tuple of probabilistic polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ and sets $\mathcal{K}, \mathcal{M}$ such that

- the **key generation algorithm** Gen receives a security parameter ℓ and outputs a key $K \leftarrow \mathsf{Gen}(1^\ell)$, with $K \in \mathcal{K}$, key length $|K| \geq \ell$;
- the **tag-generation algorithm** Mac maps a key K and a message $M \in \mathcal{M} = \{0,1\}^*$ to a tag $T \leftarrow \mathsf{Mac}_K(M)$;
- the **verification algorithm** Vrfy maps a key K, a message M and a tag T to an output bit $b := \mathsf{Vrfy}_K(M, T) \in \{0,1\}$, with $b = 1$ meaning the tag is "valid" and $b = 0$ meaning it is "invalid".
- for all ℓ, $K \leftarrow \mathsf{Gen}(1^\ell)$, and $M \in \{0,1\}^m$: $\mathsf{Vrfy}_K(M, \mathsf{Mac}_K(M)) = 1$.

MAC security definition: existential unforgeability

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

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



1. challenger generates random key $K \leftarrow \mathsf{Gen}(1^\ell)$
2. adversary $\mathcal{A}$ is given oracle access to $\mathsf{Mac}_K(\cdot)$; let $\mathcal{Q} = \{M^1, \ldots, M^t\}$ denote the set of queries that $\mathcal{A}$ asks the oracle
3. adversary outputs (M, T)
4. the experiment outputs 1 if $\mathsf{Vrfy}_K(M,T) = 1$ and $M \notin \mathcal{Q}$

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

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

MACs versus security protocols

MACs prevent adversaries forging new messages. But adversaries can still

1. replay messages seen previously ("pay £1000", 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

$$
\begin{array}{lll}
A \to B: & (M_1, T_1) & \text{with } T_1 = \mathsf{Mac}_K(M_1, 0) \\
B \to A: & (M_2, T_2) & \text{with } T_2 = \mathsf{Mac}_K(M_2, T_1) \\
A \to B: & (M_3, T_3) & \text{with } T_3 = \mathsf{Mac}_K(M_3, T_2) \\
 & \vdots & \\
B \to A: & (M_{2i}, T_{2i}) & \text{with } T_{2i} = \mathsf{Mac}_K(M_{2i}, T_{2i-1}) \\
A \to B: & (M_{2i+1}, T_{2i+1}) & \text{with } T_{2i+1} = \mathsf{Mac}_K(M_{2i+1}, T_{2i}) \\
 & \vdots &
\end{array}
$$

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

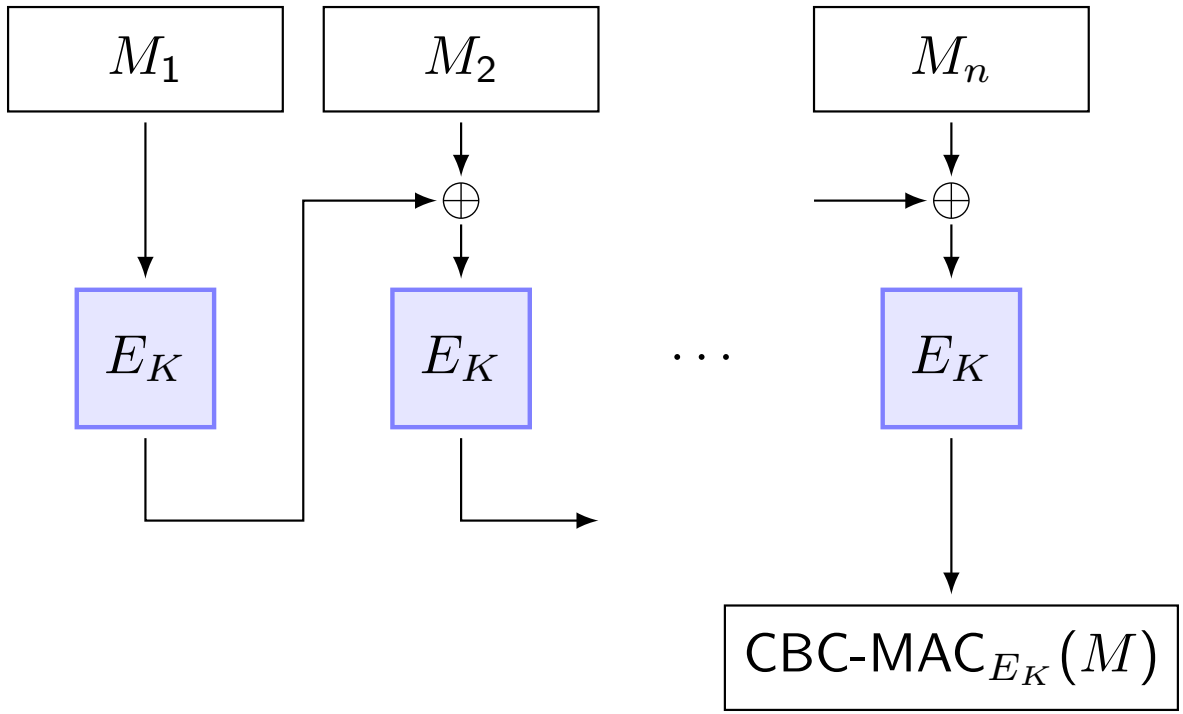
Let F be a pseudo-random function.

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

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

MAC using a block cipher: CBC-MAC

Blockcipher $E : \{0,1\}^\ell \times \{0,1\}^m \to \{0,1\}^m$



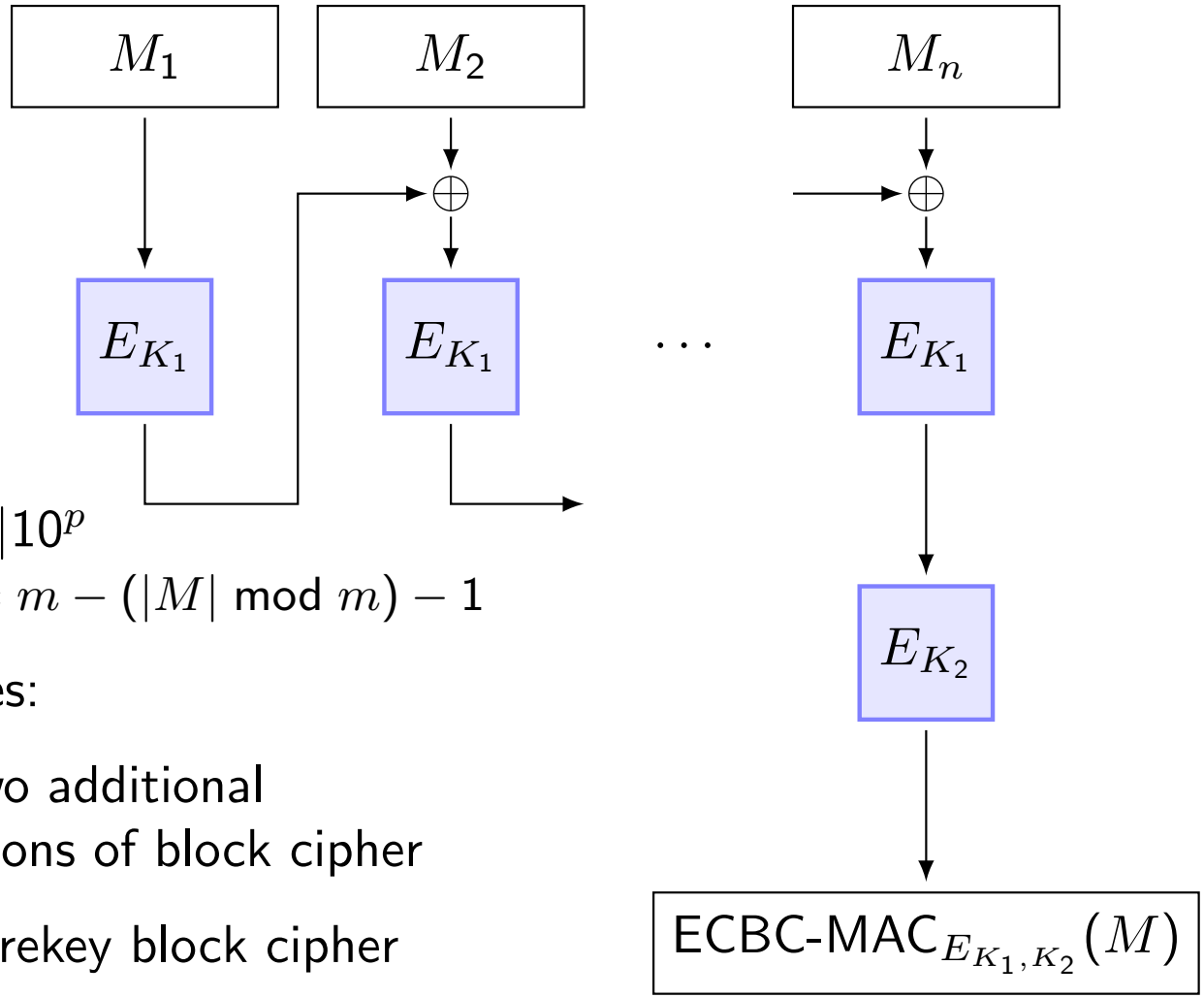
Similar to CBC: $IV = 0^m$, last ciphertext block serves as tag.

Provides existential unforgeability, but only for **fixed** message length n:

Adversary asks oracle for $T^1 :=$ CBC-MAC$_{E_K}(M^1) = E_K(M^1)$ and then presents $M = M^1 \| (T^1 \oplus M^1)$ and $T :=$ 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 $E : \{0,1\}^\ell \times \{0,1\}^m \to \{0,1\}^m$



Padding: $M \| 10^p$

$$p = m - (|M| \bmod m) - 1$$

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

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

CMAC algorithm:

$M_1 \| M_2 \| \ldots \| M_n := M$
$r := |M_n|$
if $r = m$ **then** $M_n := K_1 \oplus M_n$
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

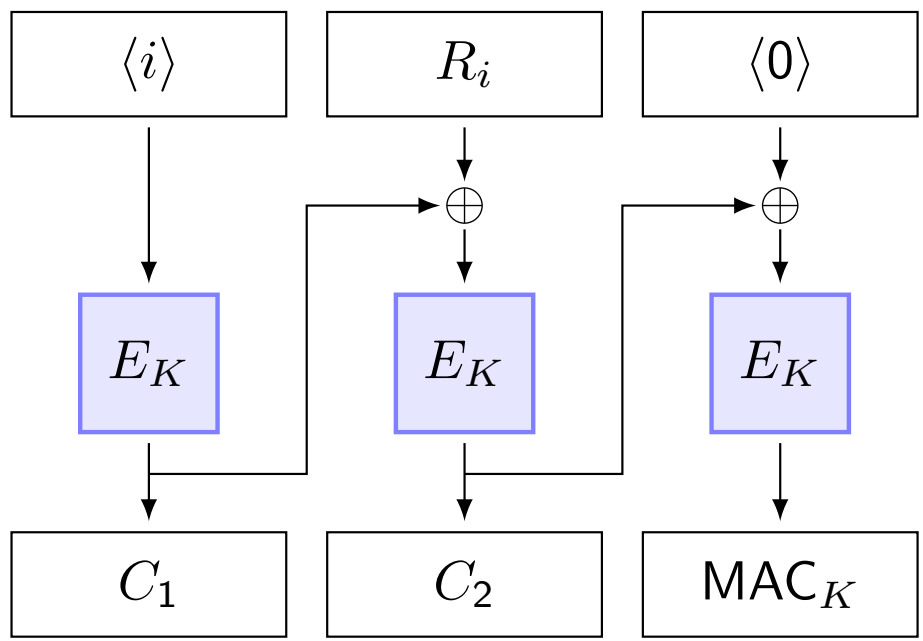
Birthday attack against CBC-MAC, ECBC-MAC, CMAC

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

Birthday/collision attack:

- Make $t \approx \sqrt{2^m}$ oracle queries for $T^i := \mathsf{MAC}_K(\langle i \rangle \| R_i \| \langle 0 \rangle)$ with $R_i \in_R \{0,1\}^m$, $1 \leq i \leq t$.
  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 $T' := \mathsf{MAC}_K(\langle i \rangle \| R_i \| \langle 1 \rangle)$
- Present $M := \langle j \rangle \| R_j \| \langle 1 \rangle$ and $T := T' = \mathsf{MAC}_K(M)$

The same intermediate value C_2 occurs while calculating the MAC of
$\langle i \rangle \| R_i \| \langle 0 \rangle$, $\langle j \rangle \| R_j \| \langle 0 \rangle$,
$\langle i \rangle \| R_i \| \langle 1 \rangle$, $\langle j \rangle \| R_j \| \langle 1 \rangle$.



Possible workaround:
Truncate MAC result to less than m bits, such that adversary cannot easily spot collisions in C_2 from C_3.

Solution: big enough m.

A one-time MAC (Carter–Wegman)

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

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

- Pick a random key pair $K = (K_1, K_2) \in \mathbb{F}^2$
- Split padded message M into blocks $M_1, \ldots, M_n \in \mathbb{F}$
- Evaluate the following polynomial over $\mathbb{F}$ to obtain the MAC:

$$\text{OT-MAC}_{K_1,K_2}(M) = K_1^{n+1} + M_n K_1^n + \cdots + M_2 K_1^2 + M_1 K_1 + K_2$$

Converted into a computationally secure many-time MAC:

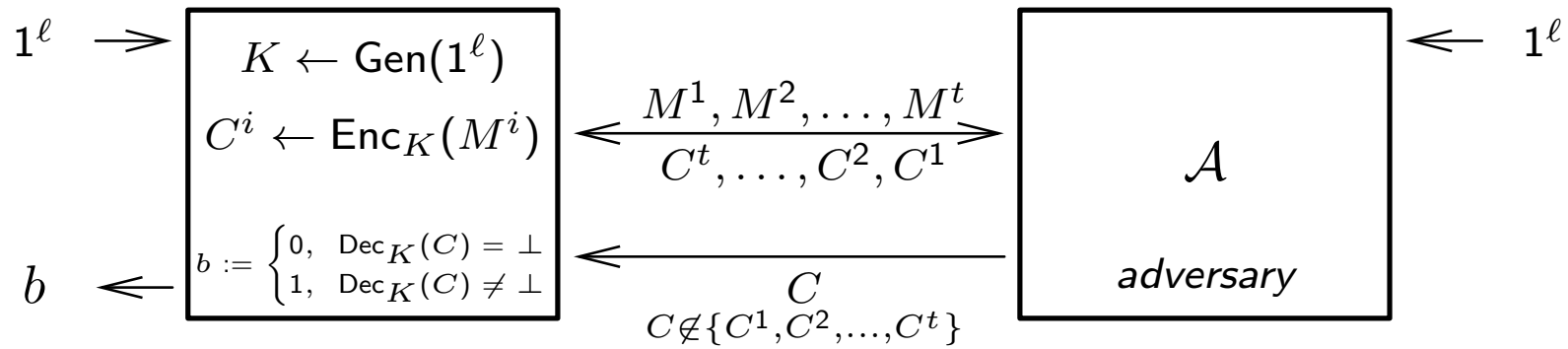
- Pseudo-random function/permutation $E_K : \mathbb{F} \to \mathbb{F}$
- Pick per-message random value $R \in \mathbb{F}$
- $\text{CW-MAC}_{K_1,K_2}(M) =$
  $(R, K_1^{n+1} + M_n K_1^n + \cdots + M_2 K_1^2 + M_1 K_1 + E_{K_2}(R))$

M. Wegman and L. Carter. New hash functions and their use in authentication and set equality. Journal of Computer and System Sciences, 22:265279, 1981.

Ciphertext integrity

Private-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, Dec can output error: $\perp$

Experiment/game $\mathsf{CI}_{\mathcal{A},\Pi}(\ell)$:



1. challenger generates random key $K \leftarrow \mathsf{Gen}(1^\ell)$
2. adversary $\mathcal{A}$ is given oracle access to $\mathsf{Enc}_K(\cdot)$; let $\mathcal{Q} = \{C^1, \ldots, C^t\}$ denote the set of query answers that $\mathcal{A}$ got from the oracle
3. adversary outputs C
4. the experiment outputs 1 if $\mathsf{Dec}_K(C) \neq \perp$ and $C \notin \mathcal{Q}$

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

$$\mathbb{P}(\mathsf{CI}_{\mathcal{A},\Pi}(\ell) = 1) \leq \mathsf{negl}(\ell)$$

Authenticated encryption

Definition: An encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ 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 $\Pi_{\mathsf{E}} = (\mathsf{Gen}_{\mathsf{E}}, \mathsf{Enc}, \mathsf{Dec})$
Message authentication code $\Pi_{\mathsf{M}} = (\mathsf{Gen}_{\mathsf{M}}, \mathsf{Mac}, \mathsf{Vrfy})$

Encryption scheme $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$:

1. $\mathsf{Gen}'(1^\ell) := (K_{\mathsf{E}}, K_{\mathsf{M}})$ with $K_{\mathsf{E}} \leftarrow \mathsf{Gen}_{\mathsf{E}}(1^\ell)$ and $K_{\mathsf{M}} \leftarrow \mathsf{Gen}_{\mathsf{M}}(1^\ell)$
2. $\mathsf{Enc}'_{(K_{\mathsf{E}}, K_{\mathsf{M}})}(M) := (C, T)$ with $C \leftarrow \mathsf{Enc}_{K_{\mathsf{E}}}(M)$ and $T \leftarrow \mathsf{Mac}_{K_{\mathsf{M}}}(C)$
3. Dec' on input of $(K_{\mathsf{E}}, K_{\mathsf{M}})$ and (C, T) first check if $\mathsf{Vrfy}_{K_{\mathsf{M}}}(C, T) = 1$. If yes, output $\mathsf{Dec}_{K_{\mathsf{E}}}(C)$, if no output $\perp$.

If Π_{E} is a CPA-secure private-key encryption scheme and Π_{M} is a secure message authentication code with unique tags, then Π' 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 $\mathsf{Vrfy}_K(M, T) = 1$.

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: $(\mathsf{Enc}_{K_\mathsf{E}}(M), \mathsf{Mac}_{K_\mathsf{M}}(M))$
Unlikely to be CPA secure: MAC may leak information about M.

Authenticate-then-encrypt: $\mathsf{Enc}_{K_\mathsf{E}}(M \| \mathsf{Mac}_{K_\mathsf{M}}(M))$
May not be CPA secure: the recipient first decrypts the received message with $\mathsf{Dec}_{K_\mathsf{E}}$, then parses the result into M and $\mathsf{Mac}_{K_\mathsf{M}}(M)$ 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: $(\mathsf{Enc}_{K_\mathsf{E}}(M), \mathsf{Mac}_{K_\mathsf{M}}(\mathsf{Enc}_{K_\mathsf{E}}(M)))$
Secure: provides both CCA security and existential unforgeability.

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.

Note: CCA security alone does not imply existential unforgeability.

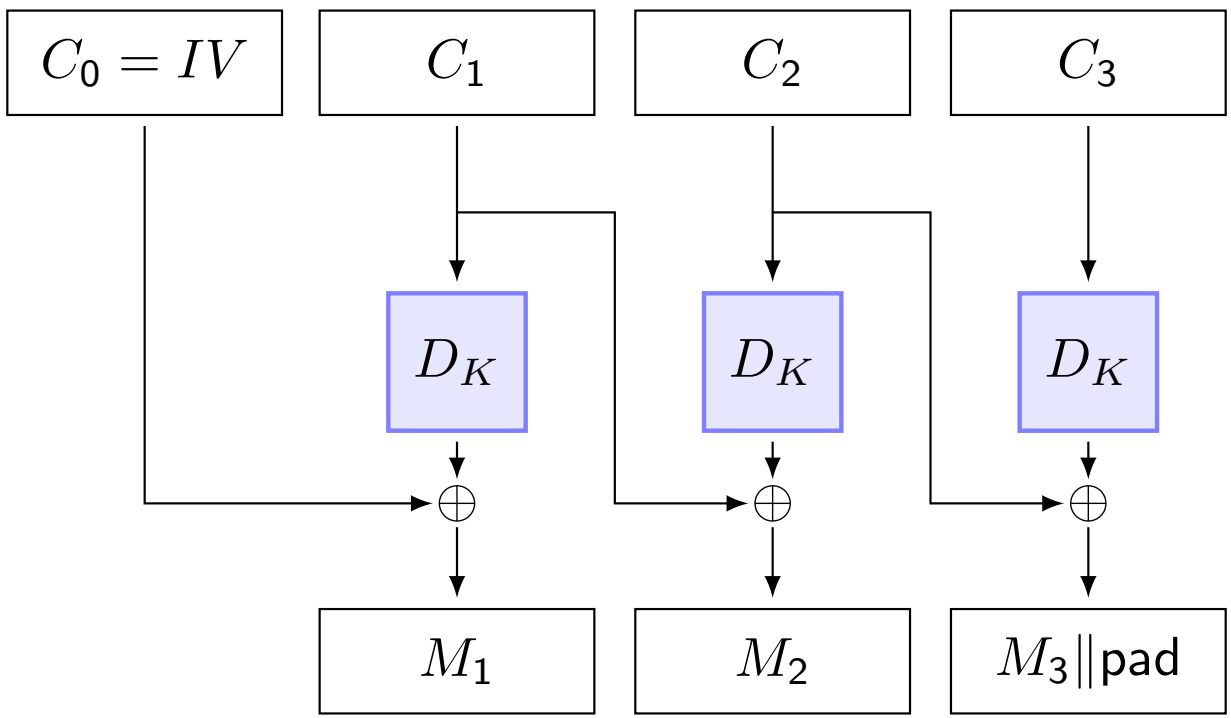
Padding oracle

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.



Padding oracle (cont'd)

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

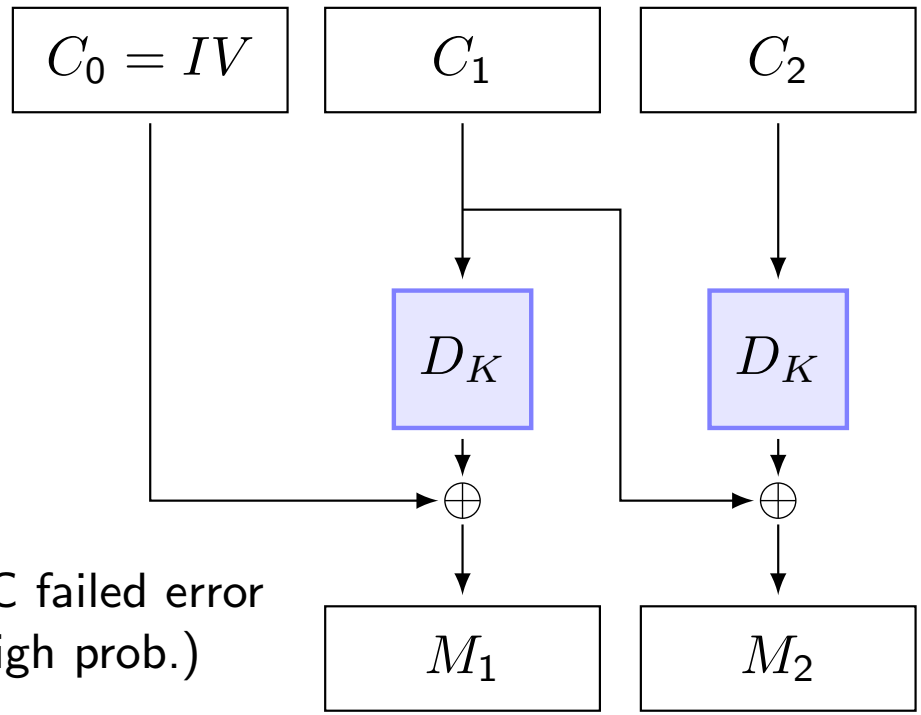
- truncate ciphertext after C_2
- $a =$ actual last byte of M_2,
  $g =$ attacker's guess of a
  (try all $g \in \{0, \ldots, 255\}$)
- XOR the last byte of C_1 with

  $$g \oplus \texttt{0x01}$$

- last byte of M_2 is now

  $$a \oplus g \oplus \texttt{0x01}$$

- $g = a$: padding correct $\Rightarrow$ 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:

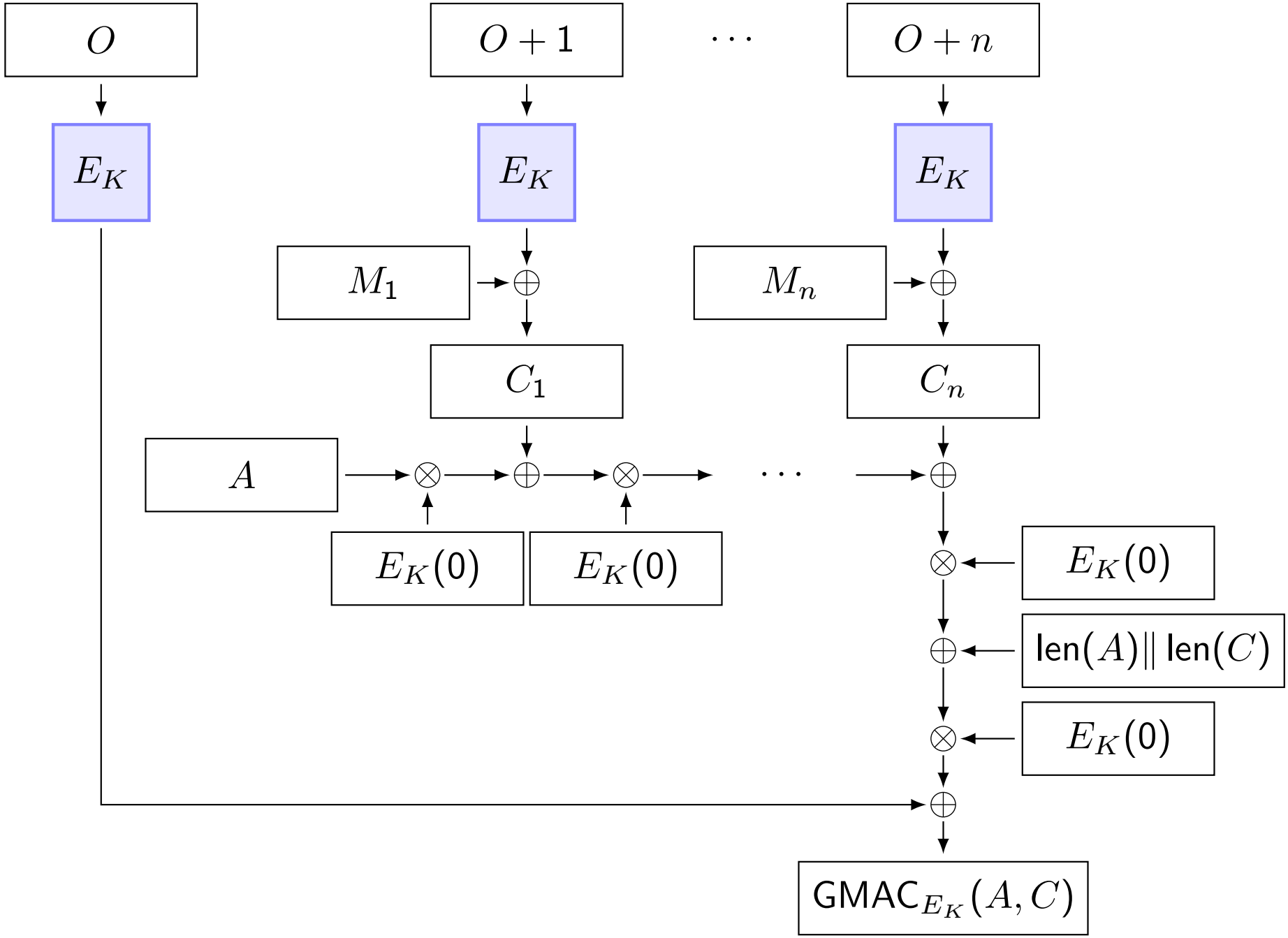
$$\begin{aligned} C_i &= M_i \oplus E_K(O+i) \\ G_i &= (G_{i-1} \oplus C_i) \otimes H, \quad G_0 = A \otimes H, \quad H = E_K(0) \\ \mathsf{GMAC}_{E_K}(A, C) &= \big((G_n \oplus (\mathsf{len}(A) \| \, \mathsf{len}(C))) \otimes H\big) \oplus E_K(O) \end{aligned}$$

A is *associated data*: authenticated, but not encrypted (e.g., header).

The multiplication $\otimes$ is over the Galois field $\mathbb{F}_{2^{128}}$: 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.

`https://csrc.nist.gov/pubs/sp/800/38/d/final`

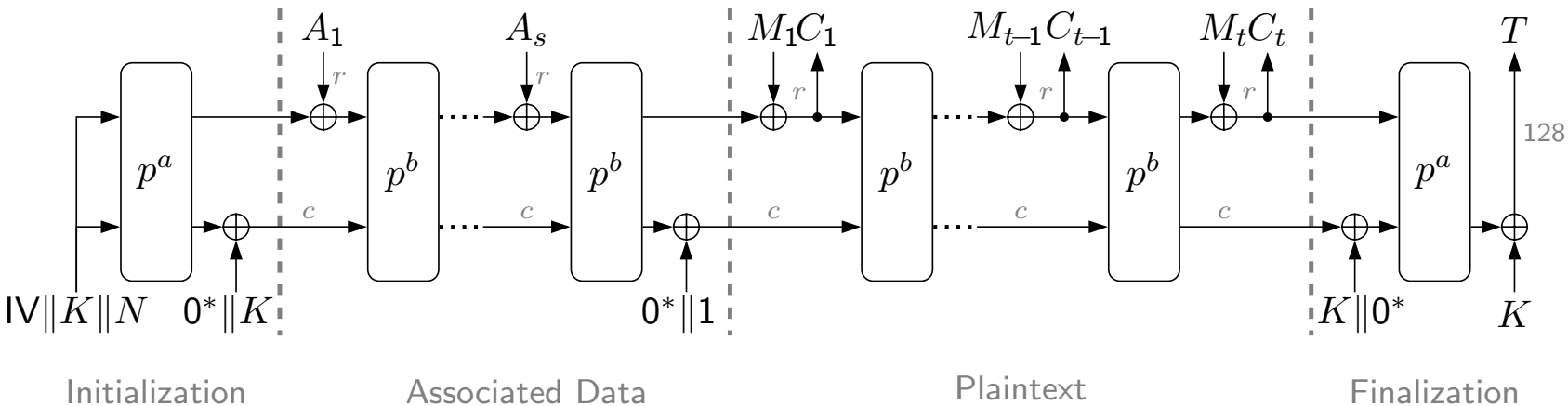

O
$O+1$
$\cdots$
$O+n$
E_K
E_K
E_K
M_1
M_n
C_1
C_n
A
$\cdots$
$E_K(0)$
$E_K(0)$
$E_K(0)$
$\mathsf{len}(A)\|\,\mathsf{len}(C)$
$E_K(0)$
$\mathsf{GMAC}_{E_K}(A, C)$

Duplex mode

Another way of implementing *authenticated encryption with associated data (AEAD)* uses a fixed $(r + c)$-bit permutation p in *duplex mode*.

Plain-text blocks are XOR-ed into the top r bits ("rate") of the $(r + c)$-bit internal state of the construct, the result of which forms the next ciphertext block. The remaining c bits ("capacity") of the state remain inaccessible to the adversary. After each block is processed, a fixed permutation p is applied to mix the entire internal state.

ASCON is an AEAD cipher, selected in 2023 by the NIST Lightweight Cryptography competition, based on a $5 \times 64 = 320$-bit permutation p used in duplex mode. Encryption:



https://ascon.isec.tugraz.at/

Hash functions

A *hash function* $h : \{0,1\}^* \to \{0,1\}^\ell$ efficiently maps arbitrary-length input strings onto fixed-length "hash values" such that the output is uniformly distributed in practice.

Typical applications of hash functions:

- hash table: data structure for fast $t = O(1)$ table lookup; storage address of a record containing value x is determined by $h(x)$.
- Bloom filter: data structure for fast probabilistic set membership test
- fast probabilistic string comparison (record deduplication, diff, rsync)
- Rabin–Karp algorithm: substring search with rolling hash

Closely related: checksums (CRC, Fletcher, Adler-32, etc.)

A good hash function h is one that minimizes the chances of a *collision* of the form $h(x) = h(y)$ with $x \neq y$.

But constructing collisions is not difficult for normal hash functions and checksums, e.g. to modify a file without affecting its checksum.

Algorithmic complexity attack: craft program input to deliberately trigger worst-case runtime (denial of service). Example: deliberately fill a server's hash table with colliding entries.

Secure hash functions

A *secure, collision-resistant* ℓ*-bit hash function* $h : \{0,1\}^* \to \{0,1\}^\ell$ is designed to make it infeasible for an adversary who knows the implementation of h to find *any* collision

$$h(x) = h(y) \quad \text{with} \quad x \neq y$$

Examples for applications of secure hash functions:

- message digest for efficient calculation of digital signatures
- fast message-authentication codes (HMAC)
- tamper-resistant checksum of files

```
$ sha1sum security?-slides.tex
2c1331909a8b457df5c65216d6ee1efb2893903f security1-slides.tex
50878bcf67115e5b6dcc866aa0282c570786ba5b security2-slides.tex
```

- git commit identifiers
- P2P file sharing identifiers
- key derivation functions
- password verification
- hash chains (e.g., Bitcoin, timestamping services)
- commitment protocols

Secure hash functions: standards

- MD5: $\ell = 128$ (Rivest, 1991)
  insecure, collisions were found in 1996/2004, collisions used in real-world attacks (Flame, 2012) $\rightarrow$ avoid (still ok for HMAC)

  `https://www.ietf.org/rfc/rfc1321.txt`

- SHA-1: $\ell = 160$ (NSA, 1995)
  widely used today (e.g., git), but 2^{69}-step algorithm to find collisions found in 2005 $\rightarrow$ being phased out (still ok for HMAC)

- SHA-2: $\ell = 224$, 256, 384, or 512
  close relative of SHA-1, therefore long-term collision-resistance questionable, very widely used standard

  FIPS 180-4 US government secure hash standard,
  `https://csrc.nist.gov/publications/fips/`

- SHA-3: KECCAK wins 5-year NIST contest in October 2012
  no length-extension attack, arbitrary-length output,
  can also operate as PRNG, very different from SHA-1/2.
  (other finalists: BLAKE, Grøstl, JH, Skein)

  `https://csrc.nist.gov/projects/hash-functions/sha-3-project`
  `https://keccak.team/keccak.html`

Collision resistance – a formal definition

Hash function

A hash function is a pair of probabilistic polynomial-time (PPT) algorithms (Gen, H) where

- Gen reads a security parameter 1^n and outputs a key s.
- H reads key s and input string $x \in \{0,1\}^*$ 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 $s = \text{Gen}(1^n)$
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 (Gen, H) is *collision resistant* if for all PPT adversaries $\mathcal{A}$ there is a negligible function negl such that

$$\mathbb{P}(\mathcal{A} \text{ found a collision}) \leq \text{negl}(n)$$

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 $h : \{0,1\}^* \to \{0,1\}^\ell$.

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 $\mathcal{A}$ 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.

Weaker properties implied by collision resistance

Second-preimage resistance

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

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

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 $\mathcal{A}$ that can break the pre-image resistance of H_s, then $\mathcal{A}$ 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 ($\rightarrow$ CPA).

Merkle–Damgård construction

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

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

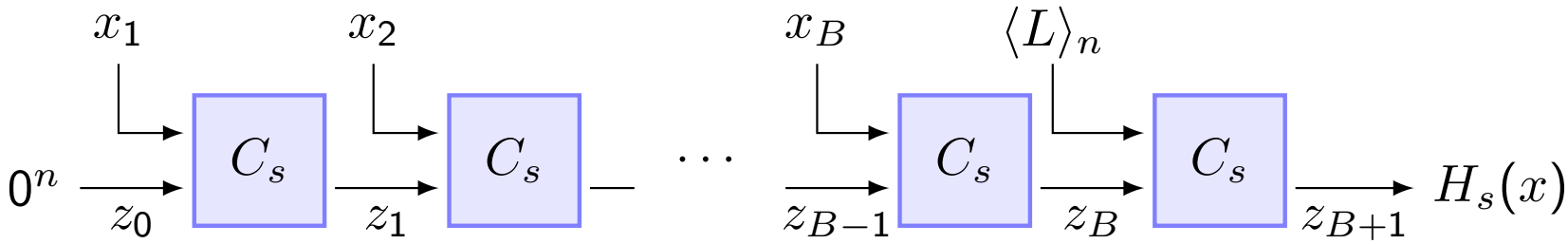
Input of H: key s, string $x \in \{0,1\}^L$ with length $L < 2^n$

1. 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:

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

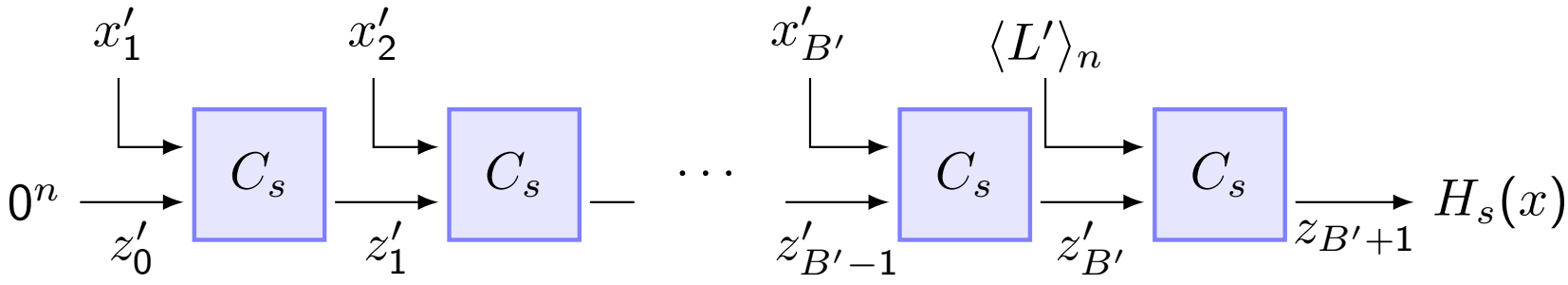
2. Append a final block $x_{B+1} = \langle L \rangle_n$, which contains the n-bit binary representation of input length $L = |x|$.
3. Set $z_0 := 0^n$ (initial vector, IV)
4. compute $z_i := C_s(z_{i-1} \| x_i)$ for $i = 1, \ldots, B+1$
5. Output $H_s(x) := z_{B+1}$

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



$$x \neq x'$$

$$x' \| 0^{n\lceil \frac{L'}{n} \rceil - L'} = x'_1 \| x'_2 \| x'_3 \| \dots \| x'_{B'-1} \| x'_{B'}$$



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_s is collision resistant, but H is not, because some PPT adversary $\mathcal{A}$ 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 $x'_1, \ldots, x'_{B'}$ those of L'-bit input x', and $x_{B+1} = \langle L \rangle_n$, $x'_{B'+1} = \langle L' \rangle_n$.

Case $L \neq L'$: Then $x_{B+1} \neq x'_{B'+1}$ but $H_s(x) = z_{B+1} = C_s(z_B \| x_{B+1}) = C_s(z'_{B'} \| x'_{B'+1}) = z'_{B'+1} = H_s(x')$, 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.

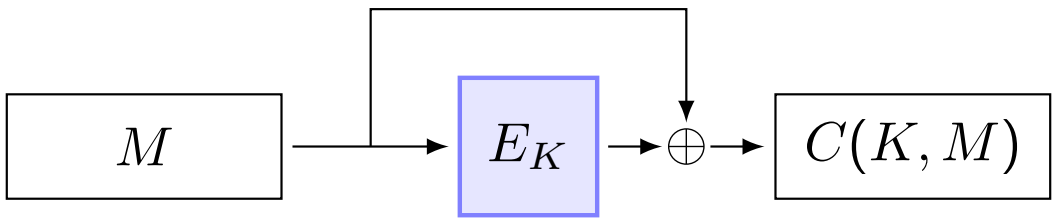
So C_s was not collision resistant, invalidating the assumption.

Compression function from block ciphers

Davies–Meyer construction

One possible technique for obtaining a collision-resistant compression function C is to use a block cipher $E : \{0,1\}^\ell \times \{0,1\}^n \to \{0,1\}^n$ in the following way:

$$C(K, M) = E_K(M) \oplus M$$



or in the notation of slide 113 (with $K = x_i$ and $M = z_{i-1}$):

$$C(z_{i-1} \| x_i) = E_{x_i}(z_{i-1}) \oplus z_{i-1}$$

However, the security proof for this construction requires E to be an *ideal cipher*, a keyed random permutation. It is not sufficient for E to merely be a strong pseudo-random permutation.

Warning: use only block ciphers that have specifically been designed to be used this way. Other block ciphers (e.g., DES) may have properties that can make them unsuitable here (e.g., related key attacks, block size too small).

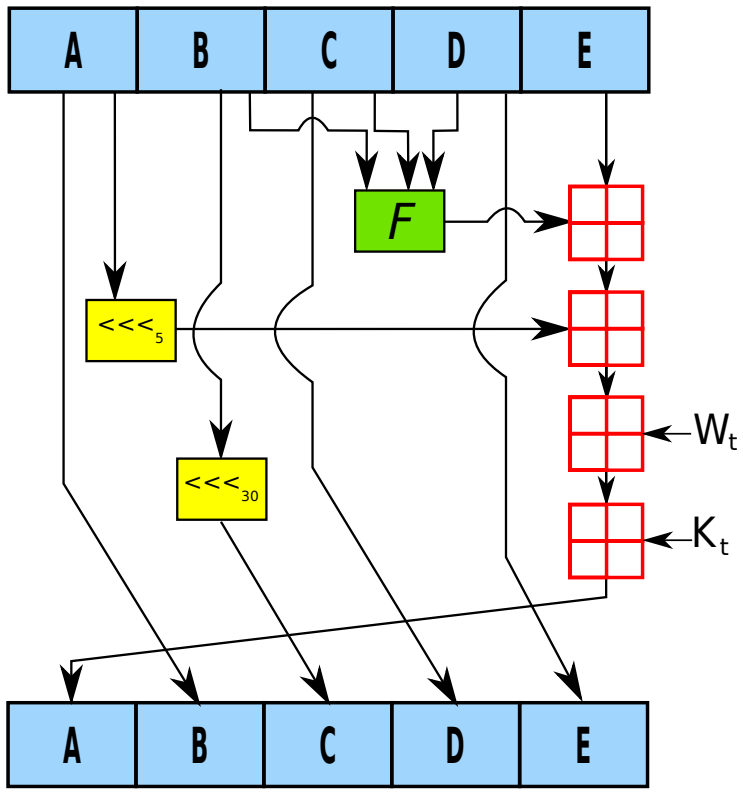
SHA-1 structure

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 $\boxplus$ 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:



commons.wikimedia.org, CC SA-BY

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 $H(X) \in \{0,1\}^\ell$ 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

A 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.

Davies–Meyer construction – security proof

$$C(K, X) = E_K(X) \oplus X$$

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

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

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

$$c_i = C_i(K_i, X_i) = E_{K_i}(X_i) \oplus X_i.$$

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

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

$\mathcal{A}$ needs to find $c_i = c_j$ with $i > j$.

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:

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

Each query will reveal a new uniformly distributed ℓ-bit value, except that it may be constrained by (at most) $i - 1$ previous query results (since E_{K_i} 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

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

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

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

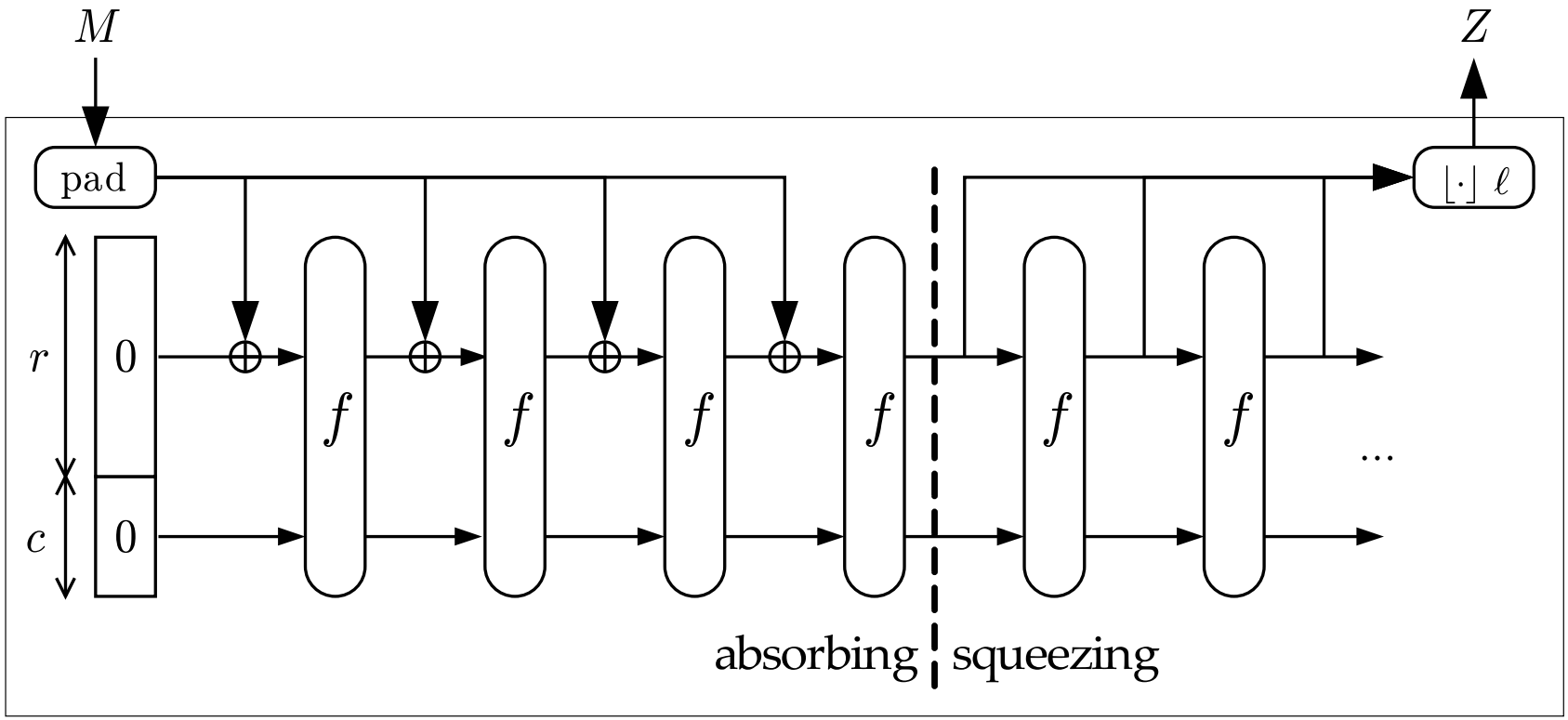
- Real hash algorithms are publicly known. Anyone can query them privately as often as they want, and look for shortcuts.
- No good justification to believe that proofs in the random oracle model say anything about the security of a scheme when implemented with practical hash functions (or pseudo-random 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 $H(M) = Z$:



sponge

https://keccak.team/sponge_duplex.html

$(r + c)$-bit internal state, XOR r-bit input blocks at a time, stir with *fixed* permutation f, output r-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 $b = r + c = 1600 = 5 \times 5 \times 64$ 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. `https://doi.org/10.6028/NIST.FIPS.202`

“Birthday attacks”

If a hash function outputs ℓ-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 $\Longrightarrow$ 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}

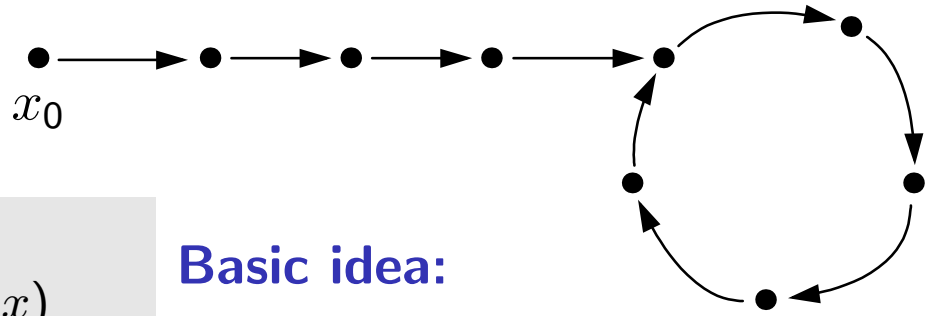
Both A and B can be phrased in 2^3 variants each $\Longrightarrow$ 2^6 pairs of phrases.

With a 64-bit hash over an entire letter, we need only such sentences for a good chance to find a collision in steps.

Low-memory collision search

A normal search for an ℓ-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
  $i := i + 1$
  $x := H(x)$ // $x = H^i(x_0)$
  $x' := H(H(x'))$ // $x' = H^{2i}(x_0)$
until $x = x'$
$x := x_0$
for $j = 1, 2, \ldots, i$
  if $H(x) = H(x')$ **return** (x, x')
  $x := H(x)$ // $x = H^j(x_0)$
  $x' := H(x')$ // $x' = H^{i+j}(x_0)$

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
  $\Rightarrow x'$ now i steps ahead of x
  $\Rightarrow i$ is now an integer multiple of the cycle length
- loop 2: x back at start, x' is i steps ahead, same speed
  $\Rightarrow$ meet at cycle entry point

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.

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

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.

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

Hash and MAC

A secure hash function can be combined with a fixed-length MAC to provide a variable-length MAC $\mathsf{Mac}_k(H(m))$. More formally:

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

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

If Π offers existential unforgeability and Π_H is collision resistant, then Π' will offer existential unforgeability.

Proof outline: If an adversary used Mac' to get tags on a set Q of messages, and then can produce a valid tag for $m^* \notin Q$, then there are two cases:

- $\exists m \in Q$ with $H_s(m) = H_s(m^*) \Rightarrow H_s$ not collision resistant
- $\forall m \in Q : H_s(m) \neq H_s(m^*) \Rightarrow$ Mac failed existential unforgeability

Hash-based message authentication code

Initial idea: hash a message M prefixed with a key K to get

$$\mathrm{MAC}_K(M) = h(K \| M)$$

This construct is secure in the random oracle model (where h is a random function). Is 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

$$\mathrm{MAC}_K(M) = h(h(K \| M))$$

or

$$\mathrm{MAC}_K(M) = h(K \| h(M))$$

could be used to terminate the iteration of the compression function in a way that the adversary cannot continue. $\Rightarrow$ HMAC

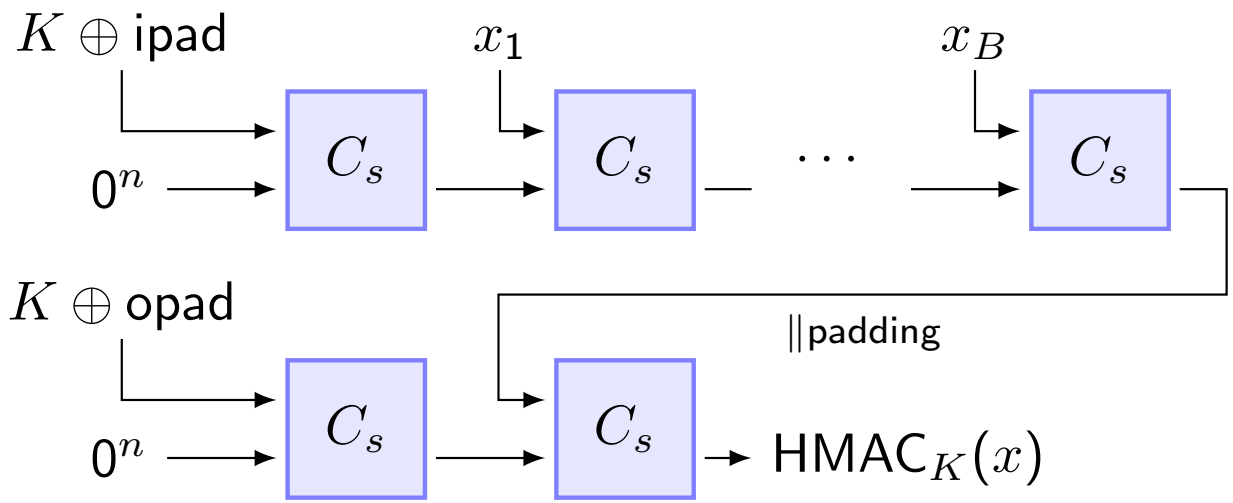
HMAC

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:

$$\mathrm{HMAC}_K(x) = h(K \oplus \mathrm{opad} \| h(K \oplus \mathrm{ipad} \| x))$$

Fixed padding values ipad, opad extend the key to the input size of the compression function, to permit precomputation of its first iteration.

$$x \| \mathrm{padding} = x_1 \| x_2 \| x_3 \| \ldots \| x_{B-1} \| x_B$$



`https://www.ietf.org/rfc/rfc2104.txt`

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.

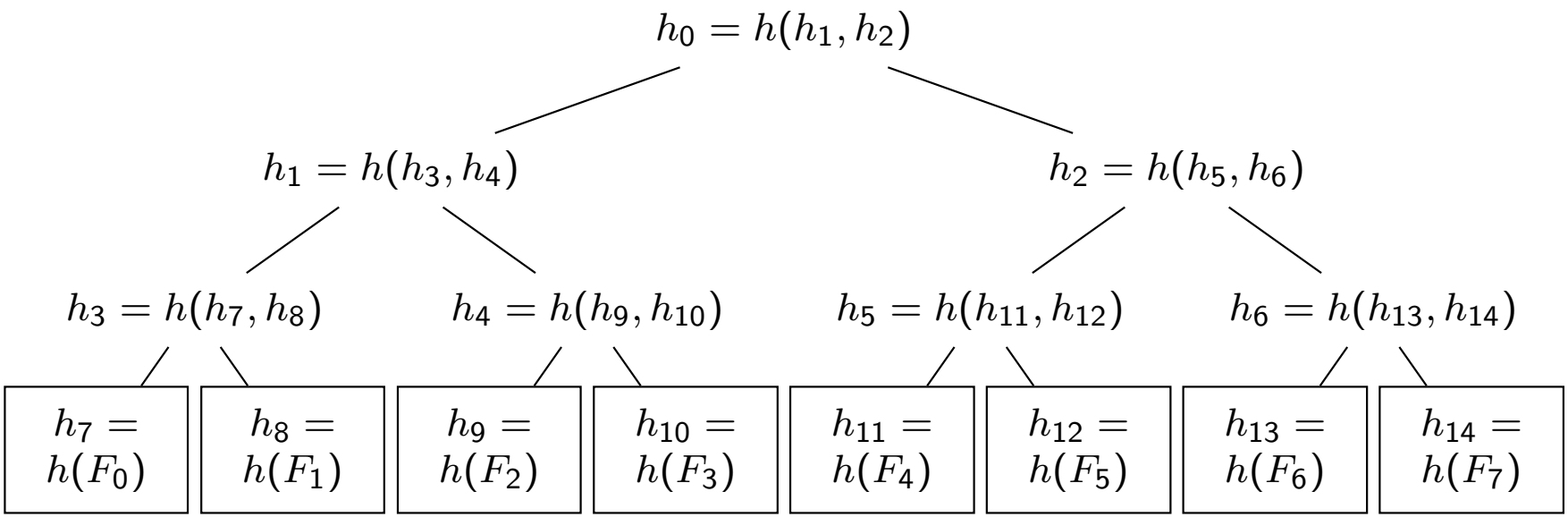
Merkle tree

Problem: Untrusted file store, small trusted memory. Solution: hash 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).



One-time passwords from a hash chain

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

$$\begin{aligned} R_0 &\leftarrow \text{random} \\ R_1 &:= h(R_0) \\ &\vdots \\ R_{n-1} &:= h(R_{n-2}) \\ R_n &:= h(R_{n-1}) \end{aligned}$$

Equivalently: $R_i := \underbrace{h(h(h(\ldots h(}_{i \text{ times}} R_0)\ldots))) = h^i(R_0) \quad (0 < i \leq n)$

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 $h(R_i) \stackrel{?}{=} H$. If they match:

- Update $H := R_i$ on host
- grant access to user

Leslie Lamport: *Password authentication with insecure communication*. CACM 24(11)770–772, 1981. `https://doi.acm.org/10.1145/358790.358797`

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

$$\begin{aligned} C_1 &= h(C_2, M_1) \\ C_2 &= h(C_3, M_2) \\ C_3 &= h(C_4, M_3) \\ &\ldots \\ C_{n-2} &= h(C_{n-1}, M_{n-2}) \\ C_{n-1} &= h(C_n, M_{n-1}) \\ C_n &= h(0, M_n) \end{aligned}$$

and then broadcasts the stream

$$C_1, \mathsf{Sign}(C_1), (C_2, M_1), (C_3, M_2), \ldots, (0, M_n).$$

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

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

$$
\begin{aligned}
G_1 &= (M_1, M_2, M_3) \\
G_2 &= (M_4, M_5, M_6, M_7) \\
G_3 &= (M_8, M_9) \\
&\dots
\end{aligned}
$$

and then publishes the hash chain (with timestamps t_i)

$$
\begin{aligned}
B_1 &= (G_1, t_1, 0) \\
B_2 &= (G_2, t_2, h(B_1)) \\
B_3 &= (G_3, t_3, h(B_2)) \\
&\dots \\
B_i &= (G_i, t_i, h(B_{i-1}))
\end{aligned}
$$

New blocks are broadcast to and archived by clients. Clients can

- verify that $t_{i-1} \leq t_i \leq$ 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** crypto currency 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 ≈ 68 bits)

`https://blockchain.info/` `https://en.bitcoin.it/`

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:

$$K_1 \| K_2 = h(A, K, \langle 1 \rangle), \quad K_3 \| K_4 = h(A, K, \langle 2 \rangle), \quad \ldots$$

ISO/IEC 11770-6

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" `https://ophcrack.sourceforge.io/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/1Il/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.
  Stored random string where practical, otherwise e.g. user name.

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.

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., $h = \text{MD5}$, $|P| = 95^8 \approx 2^{53}$ 8-char ASCII strings)

Idea: define "reduction" function $r : \{0,1\}^{128} \to P$, 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.

$$m \left\downarrow \begin{array}{l} x_0 \xrightarrow{r} p_1 \xrightarrow{h} x_1 \xrightarrow{r} p_2 \xrightarrow{h} \cdots \xrightarrow{h} x_{n-1} \xrightarrow{r} p_n \xrightarrow{h} x_n \quad \Rightarrow \quad L[x_n] := x_0 \\ \vdots \end{array} \right.$$

$\textsc{Precompute}(h, r, m, n)$:
- **for** $j := 1$ **to** m
  - $x_0 \in_R \{0,1\}^{128}$
  - **for** $i := 1$ **to** n
    - $p_i := r(x_{i-1})$
    - $x_i := h(p_i)$
  - **store** $L[x_n] := x_0$
- **return** L

$\textsc{invert}(h, r, L, x)$:
- $y := x$
- **while** $L[y] = \text{not found}$
  - $y := h(r(y))$
- $p = r(L[y])$
- **while** $h(p) \neq x$
  - $p := r(h(p))$
- **return** p

Trade-off

time:

$$n \approx |P|^{1/2}$$

memory:

$$m \approx |P|^{1/2}$$

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

M.E. Hellman: A cryptanalytic time–memory trade-off. IEEE Trans. Information Theory, July 1980. `https://doi.org/10.1109/TIT.1980.1056220`

Quick demonstration of the problem:

```
#!/usr/bin/env julia
function reach(P, n, m)
    h = [ rand(1:P) for i = 1:P ] # define hash function as mapping table
    b = zeros(Bool, P)            # bitmap of reached passwords
    for j = 1:m
        p = j                     # start a new hash chain
        for i = 1:n
            b[p] = 1              # been there
            p = h[p]              # continue a hash chain
        end
    end
    return sum(b)/P
end

P = 1_000_000      # number of possible passwords
n =    10_000      # length of each chain
m =    10_000      # number of chains

println("Overrun factor: ", n*m/P);
println("Reach: $(100*reach(P,n,m))%")
```

Output:

```
Overrun factor: 100.0
Reach: 13.5356%
```

Even if we hash $100\times$ more often than the number of possible passwords: the hash chains still reach merely $< 14\%$ of all 10^6 possible passwords.

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., $h = \text{MD5}$, $|P| = 95^8 \approx 2^{53}$ 8-char ASCII strings)

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

$$m \left\downarrow \begin{array}{l} \\ \vdots \end{array} \right. \quad x_0 \xrightarrow{r_1} p_1 \xrightarrow{h} x_1 \xrightarrow{r_2} p_2 \xrightarrow{h} \cdots \xrightarrow{h} x_{n-1} \xrightarrow{r_n} p_n \xrightarrow{h} x_n \quad \Rightarrow \quad L[x_n] := x_0$$

PRECOMPUTE(h, r, m, n) :
  for $j := 1$ **to** m
    $x_0 \in_R \{0,1\}^{128}$
    for $i := 1$ **to** n
      $p_i := r_i(x_{i-1})$
      $x_i := h(p_i)$
    store $L[x_n] := x_0$
  return L

INVERT(h, r, n, L, x) :
  for $k := n$ **downto** 1
    $x_{k-1} := x$
    for $i := k$ **to** n
      $p_i := r_i(x_{i-1})$
      $x_i := h(p_i)$
    if $L[x_n]$ exists
      $p_1 := r_1(L[x_n])$
      for $j := 1$ **to** n
        if $h(p_j) = x$
          return p_j
        $p_{j+1} := r_{j+1}(h(p_j))$

Trade-off

time:

$$n \approx |P|^{1/3}$$

memory:

$$m \approx |P|^{2/3}$$

Philippe Oechslin: Making a faster cryptanalytic time–memory trade-off. CRYPTO 2003. https://doi.org/10.1007/978-3-540-45146-4_36

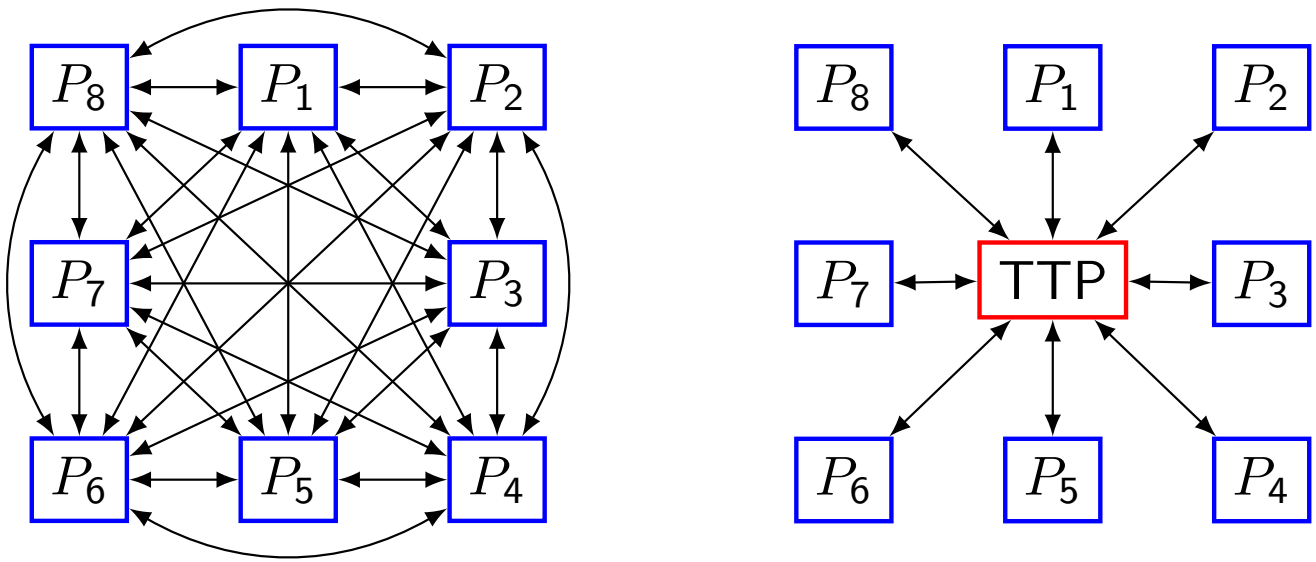
Other applications of secure hash functions

- deduplication – quickly identify in a large collection of files duplicates, without having to compare all pairs of files, just compare the hash of each files 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.



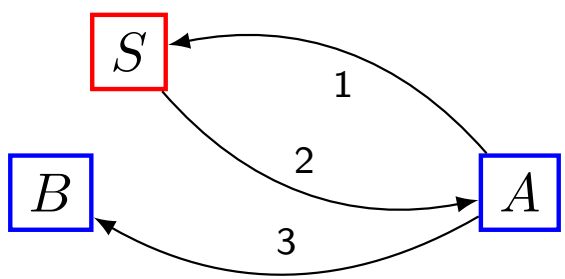
Alternative 1: introduce an intermediary "trusted third party"

Trusted third party – key distribution centre

Needham–Schroeder protocol

Communal trusted server S shares key K_{PS} with each participant P.

1. A informs S that it wants to communicate with B.
2. S generates K_{AB} and replies to A with
   $\mathsf{Enc}_{K_{AS}}(B, K_{AB}, \mathsf{Enc}_{K_{BS}}(A, K_{AB}))$
   Enc is a symmetric authenticated-encryption scheme
3. A checks name of B, stores K_{AB}, and forwards the "ticket" $\mathsf{Enc}_{K_{BS}}(A, K_{AB})$ to B
4. B also checks name of A and stores K_{AB}.
5. A and B now share K_{AB} and communicate via $\mathsf{Enc}_{K_{AB}}/\mathsf{Dec}_{K_{AB}}$.



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

1. A trusted third party generates a global key K and embeds it securely in tamper-resistant hardware tokens (e.g., smartcard)
2. Every participant receives such a token, which also knows the identity of its owner and that of any groups they might belong to.
3. Each token offers its holder authenticated encryption operations $\mathrm{Enc}_K(\cdot)$ and $\mathrm{Dec}_K(A, \cdot)$.
4. Each encrypted message $\mathrm{Enc}_K(A, M)$ contains the name of the intended recipient A (or the name of a group to which A belongs).
5. 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 $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that

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

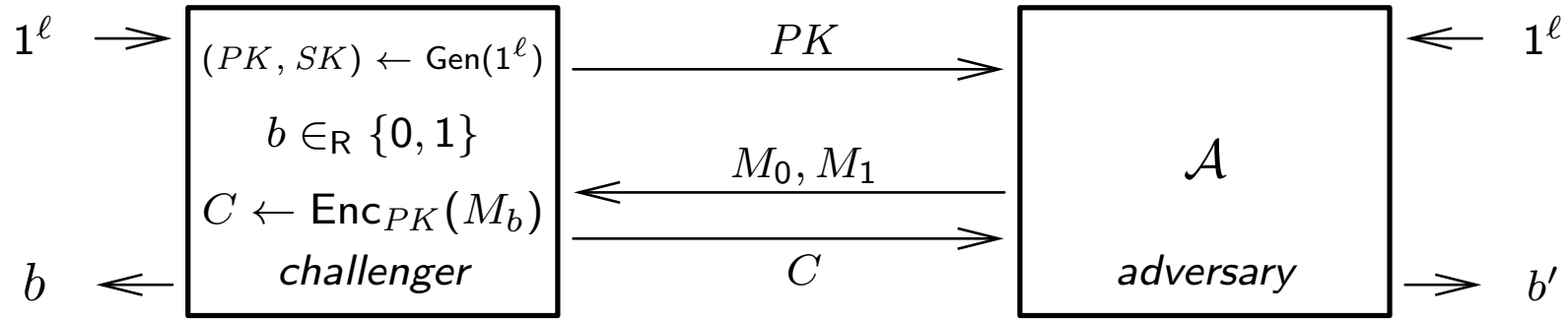
In practice, the message space $\mathcal{M}$ may depend on PK.

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

Security against chosen-plaintext attacks (CPA)

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

Experiment/game $\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(\ell)$:



Setup:

1. The challenger generates a bit $b \in_R \{0,1\}$ and a key pair $(PK, SK) \leftarrow \mathsf{Gen}(1^\ell)$.
2. The adversary $\mathcal{A}$ is given input 1^ℓ

Rules for the interaction:

1. The adversary $\mathcal{A}$ is given the public key PK
2. The adversary $\mathcal{A}$ outputs a pair of messages: $M_0, M_1 \in \{0,1\}^m$.
3. The challenger computes $C \leftarrow \mathsf{Enc}_{PK}(M_b)$ and returns C to $\mathcal{A}$

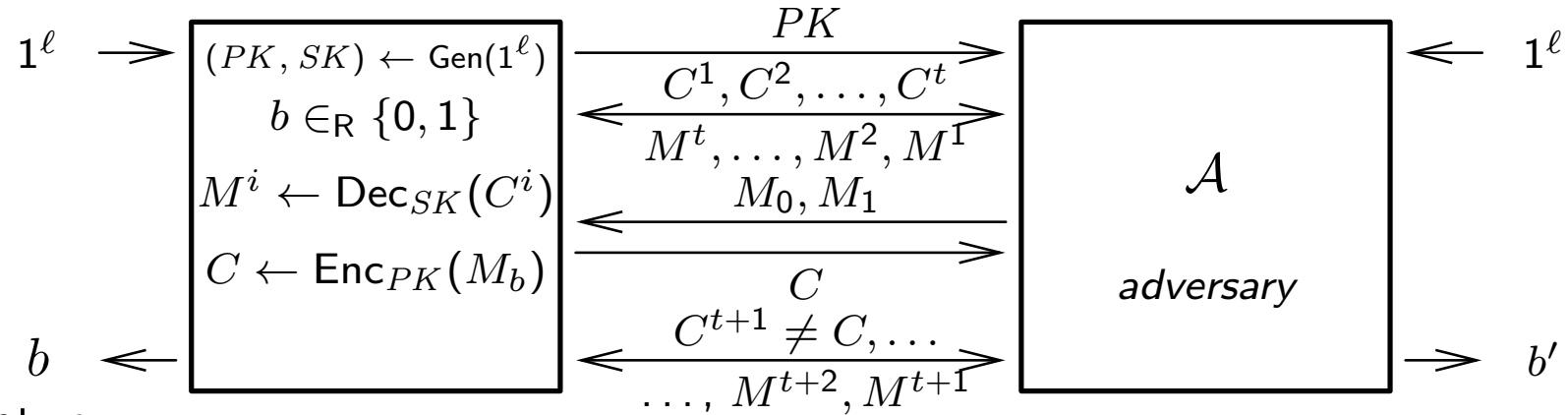
Finally, $\mathcal{A}$ outputs b'. If $b' = b$ then $\mathcal{A}$ has succeeded $\Rightarrow \mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(\ell) = 1$

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

Security against chosen-ciphertext attacks (CCA)

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

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



Setup:

- handling of ℓ, b, PK, SK as before

Rules for the interaction:

1. The adversary $\mathcal{A}$ is given PK and oracle access to Dec_{SK}:
   $\mathcal{A}$ outputs C^1, gets $\mathsf{Dec}_{SK}(C^1)$, outputs C^2, gets $\mathsf{Dec}_{SK}(C^2)$, ...
2. The adversary $\mathcal{A}$ outputs a pair of messages: $M_0, M_1 \in \{0,1\}^m$.
3. The challenger computes $C \leftarrow \mathsf{Enc}_{PK}(M_b)$ and returns C to $\mathcal{A}$
4. The adversary $\mathcal{A}$ continues to have oracle access to Dec_{SK}
   but is not allowed to ask for $\mathsf{Dec}_{SK}(C)$.

Finally, $\mathcal{A}$ outputs b'. If $b' = b$ then $\mathcal{A}$ has succeeded $\Rightarrow \mathsf{PubK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(\ell) = 1$

Security against chosen-plaintext attacks (cont'd)

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

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

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

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

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.

Number theory: integers, divisibility, primes, 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 $n = \prod_i p_i^{e_i}$, with primes p_i and positive integers e_i

The *greatest common divisor* $\gcd(a, b)$ is the largest c with $c \mid a$ and $c \mid b$.

- examples: $\gcd(18, 12) = 6$, $\gcd(15, 9) = 3$, $\gcd(15, 8) = 1$
- if $\gcd(a, b) = 1$ we say a and b are "relatively prime"
- $\gcd(a, b) = \gcd(b, a)$, $\quad \gcd(a, 0) = a$
- $\gcd(a, b) = \gcd(a, b - a)$
- if $c|ab$ and $\gcd(a, c) = 1$ then $c|b$
- if $a|n$ and $b|n$ and $\gcd(a, b) = 1$ then $ab|n$

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:

$$a \bmod b = r \quad \Rightarrow \quad 0 \leq r < b \ \wedge \ \exists q \in \mathbb{Z} : a - qb = r$$

Examples: $7 \bmod 5 = 2$, $-1 \bmod 10 = 9$

If

$$a \bmod n = b \bmod n$$

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

$$a \equiv b \pmod{n}$$

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

- reflexive: $a \equiv a \pmod{n}$
- symmetric: $a \equiv b \pmod{n} \Rightarrow b \equiv a \pmod{n}$
- transitive: $a \equiv b \pmod{n} \wedge b \equiv c \pmod{n} \Rightarrow a \equiv c \pmod{n}$

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:

$$(a - bc) \bmod n = \Big((a \bmod n) - \big((b \bmod n)(c \bmod n)\big) \bmod n\Big) \bmod n$$

Reduction modulo n limits intermediate values to

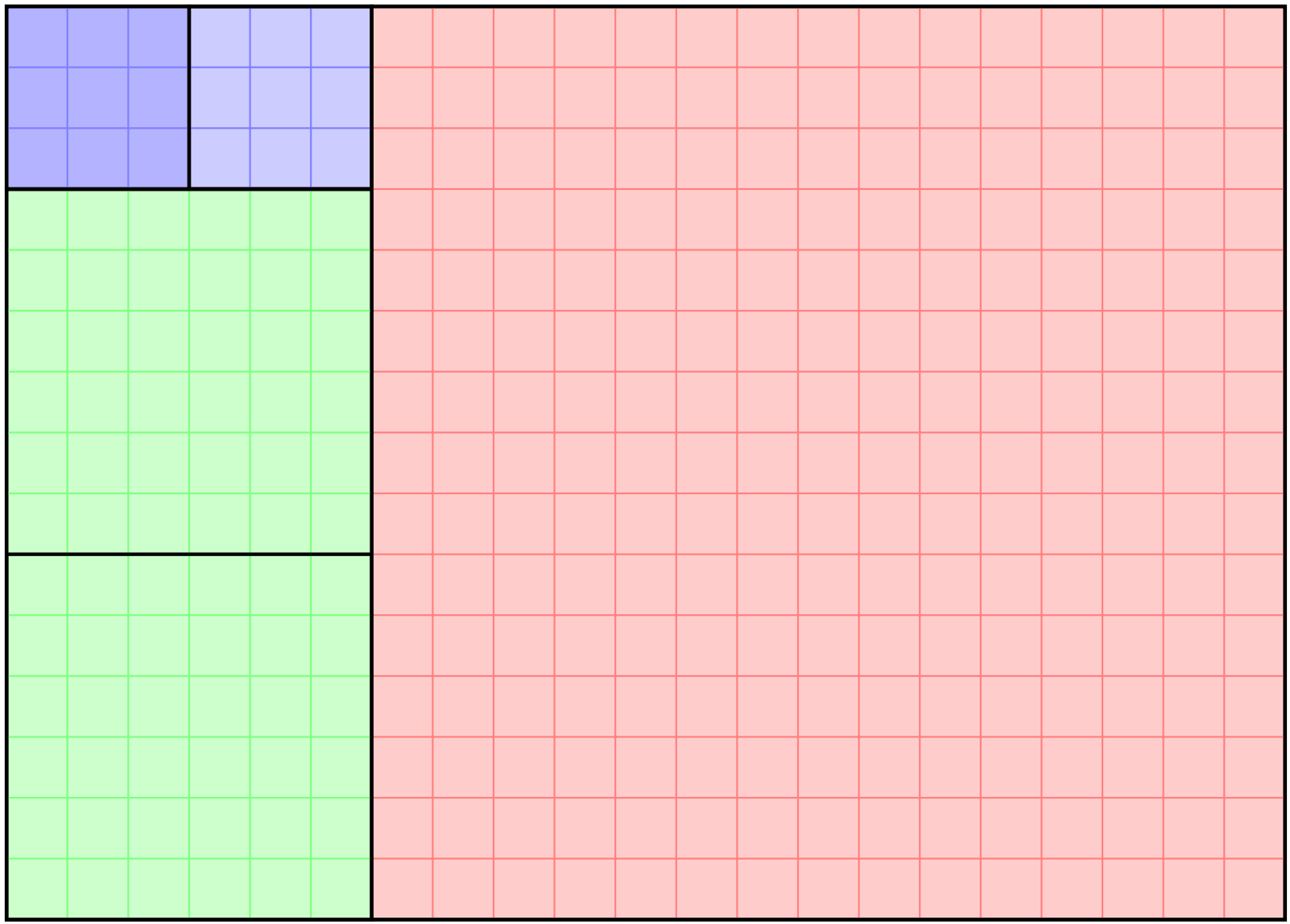
$$\mathbb{Z}_n := \{0, 1, 2, \ldots, n - 1\},$$

the "set of integers modulo n".

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

Euclid's algorithm

$\gcd(21, 15) = \gcd(15, 21 \bmod 15) = \gcd(15, 6) = \gcd(6, 15 \bmod 6) = \gcd(6, 3) = 3 = -2 \times 21 + 3 \times 15$



Euclid's algorithm

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

$$\gcd(a, b) = \begin{cases} b, & \text{if } b \mid a \\ \gcd(b, a \bmod b), & \text{otherwise} \end{cases}$$

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

Euclid's extended algorithm also provides x and y: (WLOG $a \geq b > 0$)

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

Groups

A **group** $(\mathbb{G}, \bullet)$ is a set $\mathbb{G}$ and an operator $\bullet : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ that have

closure: $a \bullet b \in \mathbb{G}$ for all $a, b \in \mathbb{G}$

associativity: $a \bullet (b \bullet c) = (a \bullet b) \bullet c$ for all $a, b, c \in \mathbb{G}$

neutral element: there exists an $e \in \mathbb{G}$ such that for all $a \in \mathbb{G}$:
$a \bullet e = e \bullet a = a$

inverse element: for each $a \in \mathbb{G}$ there exists some $b \in \mathbb{G}$ such that
$a \bullet b = b \bullet a = e$

If $a \bullet b = b \bullet a$ for all $a, b \in \mathbb{G}$, 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 $a + b := (a + b) \bmod n$
- $(\{0,1\}^n, \oplus)$ where $a_1 a_2 \ldots a_n \oplus b_1 b_2 \ldots b_n = c_1 c_2 \ldots c_n$ with $(a_i + b_i) \bmod 2 = c_i$ (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, $(\mathbb{G}, \bullet)$ 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

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 $(\mathbb{G}, \bullet)$, map each $g \in \mathbb{G}$ to a function $f_g : x \mapsto g \bullet x$. Since $g^{-1} \in \mathbb{G}$, f_g is a permutation, and the set of all f_g for $g \in \mathbb{G}$ forms a permutation group isomorphic to $\mathbb{G}$. ("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 $(\mathbb{Z}_8, +)$

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 ("$+$", "$\times$", "$\cdot$", "gh") 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 $g \cdot i := \underbrace{g \bullet g \bullet \cdots \bullet g}_{i \text{ times}}$ $(g \in \mathbb{G}, i \in \mathbb{Z})$

"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 the group operation as a dot "$g \cdot h$" or juxtaposition "gh"
- write $g^i := \underbrace{g \bullet g \bullet \cdots \bullet g}_{i \text{ times}}$ $(g \in \mathbb{G}, i \in \mathbb{Z})$

Some of these notations may similarly also be used for monoids or other algebraic structures where the behaviour of the operator in some way resembles that of addition or multiplication.

Rings

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

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

If also $a \boxtimes b = b \boxtimes a$, then we have a **commutative ring**.

Examples for rings:

- $(\mathbb{Z}[x], +, \cdot)$, where

$$\mathbb{Z}[x] := \left\{ \sum_{i=0}^{n} a_i x^i \,\middle|\, a_i \in \mathbb{Z}, n \geq 0 \right\}$$

  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

Fields

A **field** $(\mathbb{F}, \boxplus, \boxtimes)$ is a set $\mathbb{F}$ and two associative and commutative 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 $0_{\mathbb{F}}$
- $(\mathbb{F}, \boxtimes)$ is a commutative monoid with neutral element $1_{\mathbb{F}} \neq 0_{\mathbb{F}}$
- $(\mathbb{F} \setminus \{0_{\mathbb{F}}\}, \boxtimes)$ is also an abelian group (with neutral element $1_{\mathbb{F}}$)
- $a \boxtimes (b \boxplus c) = (a \boxtimes b) \boxplus (a \boxtimes c)$ and $(a \boxplus b) \boxtimes c = (a \boxtimes c) \boxplus (b \boxtimes c)$ (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 (infinitely large) fields: $(\mathbb{Q}, +, \cdot)$, $(\mathbb{R}, +, \cdot)$, $(\mathbb{C}, +, \cdot)$

For cryptographic applications, we are interested in *finite* fields, where we can pick elements uniformly at random. The *order* of a field is the number of elements it contains.

If we have $\underbrace{1 + 1 + \cdots + 1}_{i \text{ times}} = 0$ in a ring or field, then we call the smallest such i its *characteristic*.

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 $n - a \equiv -a \pmod{n}$

$(\mathbb{Z}_n, \cdot)$ is a monoid:

- neutral element of multiplication is 1

$(\mathbb{Z}_n, +, \cdot)$, with its "mod n" operators, is a ring, which means commutative, associative and distributive law works just like over $\mathbb{Z}$.

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 $\mathbb{Z}_5$: $4 + 3 = 2$, $4 \cdot 2 = 3$, $4^2 = 1$

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 $\mathbb{Z}_7^*$:
$1 \cdot 1 = 1$, $2 \cdot 4 = 1$, $3 \cdot 5 = 1$, $4 \cdot 2 = 1$, $5 \cdot 3 = 1$, $6 \cdot 6 = 1$

Finite fields (Galois fields)

$(\mathbb{Z}_p, +, \cdot)$ is a finite field with p elements, where p is a prime number. Also written as $\mathbb{F}_p$, or as GF(p), the "Galois field of order p".

We can also construct finite fields $\mathbb{F}_{p^n}$ (or GF(p^n)) with p^n elements:

Let $\mathbb{F}_q$ be a finite field with q elements. Then we can create an *extension field* $\mathbb{F}_{q^n}$ with q^n elements as follows:

- **Elements:** polynomials over variable x with degree less than n and coefficients from the finite field $\mathbb{F}_q$
- **Modulus:** select an *irreducible* polynomial $T(x) \in \mathbb{F}_q[x]$ of degree n

$$T(x) = c_n x^n + \cdots + c_2 x^2 + c_1 x + c_0$$

  where $c_i \in \mathbb{F}_q$ for all $0 \leq i \leq n$. An irreducible polynomial cannot be factored into two lower-degree polynomials from $\mathbb{F}_q[x] \setminus \{0, 1\}$.
- **Addition:** $\oplus$ is normal polynomial addition (i.e., pairwise addition of the coefficients in $\mathbb{F}_q$)
- **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 (for some prime p, $n > 0$)
Theorem: all finite fields of the same size are isomorphic

$\mathbb{F}_{2^n}$ – binary fields (fields of characteristic 2)

$\mathbb{F}_2$ is particularly easy to implement in hardware:

- addition = 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 extension fields of the form $\mathbb{F}_{2^n}$ (also written as GF(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 $\mathbb{F}_{2^{64}}$ or $\mathbb{F}_{2^{128}}$ more efficiently.

$\mathbb{F}_{2^8}$ example

The finite field $\mathbb{F}_{2^8}$ consists of the 256 polynomials of the form

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

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

$$T(x) = x^8 + x^4 + x^3 + x + 1 \quad \text{or} \quad 1\,0001\,1011$$

Example operations:

- $(x^7 + x^5 + x + 1) \oplus (x^7 + x^6 + 1) = x^6 + x^5 + x$
  or equivalently $1010\,0011 \oplus 1100\,0001 = 0110\,0010$
- $(x^6 + x^4 + 1) \otimes_T (x^2 + 1) = [(x^6 + x^4 + 1)(x^2 + 1)] \bmod T(x) =$
  $(x^8 + x^4 + x^2 + 1) \bmod (x^8 + x^4 + x^3 + x + 1) =$
  $(x^8 + x^4 + x^2 + 1) \ominus (x^8 + x^4 + x^3 + x + 1) = x^3 + x^2 + x$
  or equivalently
  $0101\,0001 \otimes_T 0000\,0101 = 1\,0001\,0101 \oplus 1\,0001\,1011 = 0000\,1110$

Multiplication and modular reduction in $\mathbb{F}_{2^n}$

Let $a(x) = \sum_{i=0}^{n-1} a_i x^i$ and $b(x) = \sum_{i=0}^{n-1} b_i x^i$ with $a_i, b_i \in \mathbb{Z}_2$ be polynomials of degree less than n that represent elements of $\mathbb{F}_{2^n}$.

Let $f(x) = x^n + r(n)$ be the irreducible modulus.

Algorithm for calculating $c(x) = [a(x) \cdot b(x)] \bmod f(x) = \sum_{i=0}^{n-1} c_i x^i$:

```
RIGHT_TO_LEFT_MULTIPLY(a, b):
    if a_0 = 1 then c := b else c := 0
    for i := 1 to n − 1 do
        if b_{n−1} = 1 then
            b := b ≪ 1 ⊕ r      ← shift
        else
            b := b ≪ 1
        if a_i = 1 then
            c := c ⊕ b          ← add
    return c
```

```
LEFT_TO_RIGHT_MULTIPLY(a, b):
    c := 0
    for i := n − 1 downto 0 do
        if a_i = 1 then
            c := c ⊕ b          ← add
        if i = 0 then return c
        if c_{n−1} = 1 then
            c := c ≪ 1 ⊕ r      ← shift
        else
            c := c ≪ 1
```

The left-to-right method can be accelerated by precomputing $B_u = [b \cdot u] \bmod f$ for all 2^w polynomials u of degree less than w, and then adding the B_u selected by w bits of a at a time.

Squaring in $\mathbb{F}_{2^n}$

In finite fields of characteristic 2, we have

$$\begin{aligned} 2 &= 0 \\ -1 &= 1 \\ a^2 &= a \quad \text{for all } a \in \mathbb{Z}_2 \end{aligned}$$

and as a result some expressions become much simpler.

For example: squaring of polynomials

$$(a_1x + a_0)^2 = (a_1x + a_0)(a_1x + a_0) = a_1^2x^2 + 2a_1a_0x + a_0^2 = a_1x^2 + a_0$$

More generally: if

$$a(x) = \sum_{i=0}^{n-1} a_i x^i \qquad \in \mathbb{F}_{2^n}$$

then

$$[a(x)]^2 = \sum_{i=0}^{n-1} a_i x^{2i}.$$

Finite groups

Let $(\mathbb{G}, \bullet)$ be a group with a finite number of elements $|\mathbb{G}|$.

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

Terminology:

- The *order of a group* $\mathbb{G}$ is its size $|\mathbb{G}|$
- *order of group element* g in $\mathbb{G}$ is $\mathrm{ord}_{\mathbb{G}}(g) = \min\{i > 0 \mid g^i = 1\}$.

Related notion: the *characteristic* of a ring or field is the order of 1 in its additive group, i.e. the smallest i with $\underbrace{1 + 1 + \cdots + 1}_{i \text{ times}} = 0$.

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

1. $g^m = 1$, $g^x = g^{x \bmod m}$
2. $g^x = g^{x \bmod \mathrm{ord}(g)}$
3. $g^x = g^y \Leftrightarrow x \equiv y \pmod{\mathrm{ord}(g)}$
4. $\mathrm{ord}(g) \mid m$ "Lagrange's theorem"
5. 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 $ed \bmod m = 1$

Proofs

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

1. Let $\mathbb{G}$ 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 $g \in \mathbb{G}$, because $gg_i = gg_j \Rightarrow g_i = g_j$ (see 0), which implies that each of the (gg_i) is distinct, and since there are only m elements of $\mathbb{G}$, 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: $g^m = 1 \Rightarrow g^x = g^x \cdot (g^m)^n = g^{x-nm} = g^{x \bmod m}$ for any $n \in \mathbb{Z}$.

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

3. Let $i = \mathrm{ord}(g)$.
   "$\Leftarrow$": $x \equiv y \pmod{i} \Leftrightarrow x \bmod i = y \bmod i \Rightarrow g^x = g^{x \bmod i} = g^{y \bmod i} = g^y$.
   "$\Rightarrow$": Say $g^x = g^y$, then $1 = g^{x-y} = g^{(x-y) \bmod i}$. Since $(x - y) \bmod i < i$, but i is the smallest positive integer with $g^i = 1$, we must have $(x - y) \bmod i = 0$. $\Rightarrow x \equiv y \pmod{i}$.

4. $g^m = 1 = g^0$ therefore $m \equiv 0 \pmod{\mathrm{ord}(g)}$ from 3, and so $\mathrm{ord}(g) | m$.

5. $(g^e)^d = g^{ed} = g^{ed \bmod m} = g^1 = g$ means that $g \mapsto g^d$ is indeed the inverse of $g \mapsto g^e$ if $ed \bmod m = 1$. And since $\mathbb{G}$ is finite, the existence of an inverse operation implies that $g \mapsto g^e$ is a permutation.

Katz/Lindell (2nd ed.), sections 8.1 and 8.3

Cyclic groups

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

For $g \in \mathbb{G}$ consider the set

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

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

Definitions:

- We call g a *generator* of $\mathbb{G}$ if $\langle g \rangle = \mathbb{G}$.
- We call $\mathbb{G}$ *cyclic* if it has a generator.

Useful facts:

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

   Recall that $\mathsf{ord}(g) \mid |\mathbb{G}|$. We have $\mathsf{ord}(g) \in \{1, |\mathbb{G}|\}$ if $|\mathbb{G}|$ is prime, which makes g either 1 or a generator.

Katz/Lindell (2nd ed.), 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 $|\langle g \rangle| \stackrel{?}{=} m$ is infeasible for crypto-sized m.
- Fast test: if $m = \prod_i p_i^{e_i}$ is composite, then $g \in \mathbb{G}$ is a generator if and only if $g^{m/p_i} \neq 1$ 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.

$(\mathbb{Z}_p, +)$ is a cyclic group

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

$$\mathbb{Z}_p = \langle g \rangle = \{g \cdot i \bmod p \,|\, 0 \leq i \leq p-1\}$$

Note that this follows from fact 3 on slide 176: $\mathbb{Z}_p$ is of order p, which is prime.

Example in $\mathbb{Z}_7$:

$$(0\cdot 0,\ 0\cdot 1,\ 0\cdot 2,\ 0\cdot 3,\ 0\cdot 4,\ 0\cdot 5,\ 0\cdot 6,\ 0\cdot 7,\ \ldots) = (0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ \ldots)$$
$$(1\cdot 0,\ 1\cdot 1,\ 1\cdot 2,\ 1\cdot 3,\ 1\cdot 4,\ 1\cdot 5,\ 1\cdot 6,\ 0\cdot 7,\ \ldots) = (0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 0,\ \ldots)$$
$$(2\cdot 0,\ 2\cdot 1,\ 2\cdot 2,\ 2\cdot 3,\ 2\cdot 4,\ 2\cdot 5,\ 2\cdot 6,\ 0\cdot 7,\ \ldots) = (0,\ 2,\ 4,\ 6,\ 1,\ 3,\ 5,\ 0,\ \ldots)$$
$$(3\cdot 0,\ 3\cdot 1,\ 3\cdot 2,\ 3\cdot 3,\ 3\cdot 4,\ 3\cdot 5,\ 3\cdot 6,\ 0\cdot 7,\ \ldots) = (0,\ 3,\ 6,\ 2,\ 5,\ 1,\ 4,\ 0,\ \ldots)$$
$$(4\cdot 0,\ 4\cdot 1,\ 4\cdot 2,\ 4\cdot 3,\ 4\cdot 4,\ 4\cdot 5,\ 4\cdot 6,\ 0\cdot 7,\ \ldots) = (0,\ 4,\ 1,\ 5,\ 2,\ 6,\ 3,\ 0,\ \ldots)$$
$$(5\cdot 0,\ 5\cdot 1,\ 5\cdot 2,\ 5\cdot 3,\ 5\cdot 4,\ 5\cdot 5,\ 5\cdot 6,\ 0\cdot 7,\ \ldots) = (0,\ 5,\ 3,\ 1,\ 6,\ 4,\ 2,\ 0,\ \ldots)$$
$$(6\cdot 0,\ 6\cdot 1,\ 6\cdot 2,\ 6\cdot 3,\ 6\cdot 4,\ 6\cdot 5,\ 6\cdot 6,\ 0\cdot 7,\ \ldots) = (0,\ 6,\ 5,\ 4,\ 3,\ 2,\ 1,\ 0,\ \ldots)$$

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

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

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

$$\mathbb{Z}_p^* = \{g^i \bmod p \,|\, 0 \leq i \leq p-2\}$$

Note that this does **not** follow from fact 3 on slide 176: $\mathbb{Z}_p^*$ is of order $p-1$, which is even (for $p > 3$), not prime.

Example in $\mathbb{Z}_7^*$:

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

- 3 and 5 are generators of $\mathbb{Z}_7^*$

Fast generator test (p. 177), using $|\mathbb{Z}_7^*| = 6 = 2 \cdot 3$:
$3^{6/2} = 6, 3^{6/3} = 2, 5^{6/2} = 6, 5^{6/3} = 4$, all $\neq 1$.

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

The *order* of g in $\mathbb{Z}_p^*$ is the size of the subgroup $\langle g \rangle$.
Lagrange's theorem: $\text{ord}_{\mathbb{Z}_p^*}(g) \mid p-1$ for all $g \in \mathbb{Z}_p^*$

Fermat's and Euler's theorem

Fermat's little theorem: (1640)

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

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)

$$\gcd(a,n) = 1 \quad \Leftrightarrow \quad a^{\varphi(n)} \bmod n = 1$$

- this implies that in $\mathbb{Z}_n$: $a^x = a^{x \bmod \varphi(n)}$ for any $a \in \mathbb{Z}_n^*, x \in \mathbb{Z}$

Recall from Lagrange's theorem: for $a \in \mathbb{Z}_n^*$, $\text{ord}(a) | \varphi(n)$ 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

$$f : \mathbb{Z}_n \leftrightarrow \mathbb{Z}_p \times \mathbb{Z}_q \qquad f(x) = (x \bmod p, x \bmod q)$$

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 $x_p = x \bmod p$ and $x_q = x \bmod q$ to x, we first use Euclid's extended algorithm to find a, b such that $ap + bq = 1$, and then $x = (x_p bq + x_q ap) \bmod n$.

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.

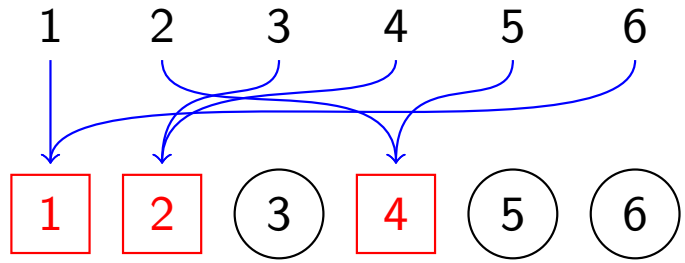
Example: $n = pq = 3 \times 5 = 15$

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$x \bmod 3$	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
$x \bmod 5$	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4

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 $x \in \mathbb{Z}_p^*$, that is if y has a square root in $\mathbb{Z}_p^*$, we call y 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:

$$c^{(p-1)/2} \bmod p = 1 \quad \Leftrightarrow \quad c \text{ is a quadratic residue in } \mathbb{Z}_p^*$$

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

$c^{(p-1)/2}$ is also called the *Legendre symbol*

Taking roots in $\mathbb{Z}_p^*$

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

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

Proof: $(c^d)^e = c^{de} = c^{de \bmod \varphi(p)} = c^{de \bmod (p-1)} = c^1 = c$.

Method 2: if $e = 2$ (taking square roots)
$\gcd(2, p-1) \neq 1$ if p odd prime $\Rightarrow$ Euclid's extended alg. no help here.

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

  Proof: $\left[c^{(p+1)/4}\right]^2 = c^{(p+1)/2} = \underbrace{c^{(p-1)/2}}_{=1} \cdot c = c.$

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

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

Solution: $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

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

Taking roots in $\mathbb{Z}_n^*$: 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 $\mathbb{Z}_p^*$.

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.

Let $\mathbb{G} = \{g^2 \mid g \in \mathbb{Z}_p^*\}$ be the set of quadratic residues of $\mathbb{Z}_p^*$. $\mathbb{G}$ with operator "multiplication mod p" is a subgroup of $\mathbb{Z}_p^*$, with order $|\mathbb{G}| = q$.

$\mathbb{G}$ has prime order $|\mathbb{G}| = q$ and ord$(g)|q$ for all $g \in \mathbb{G}$ (Lagrange's theorem):
$\Rightarrow$ ord$(g) \in \{1, q\} \Rightarrow$ ord$(g) = q$ for all $g > 1 \Rightarrow$ for all $g \in \mathbb{G} \setminus \{1\}$ $\langle g \rangle = \mathbb{G}$.

If p is a strong prime, then each quadratic residue in $\mathbb{Z}_p^*$ other than 1 is a generator of the subgroup of quadratic residues of $\mathbb{Z}_p^*$.

GENERATE_GROUP(1^ℓ):
- $p \in_R \{(\ell+1)$-bit strong primes$\}$
- $q := (p-1)/2$
- $x \in_R \mathbb{Z}_p^* \setminus \{-1, 1\}$
- $g := x^2 \bmod p$
- return p, q, g

Example: $p = 11$, $q = 5$

$g \in \{2^2, 3^2, 4^2, 5^2\} = \{4, 9, 5, 3\}$

$\langle 4 \rangle = \{4^0, 4^1, 4^2, 4^3, 4^4\} = \{1, 4, 5, 9, 3\}$

$\langle 9 \rangle = \{9^0, 9^1, 9^2, 9^3, 9^4\} = \{1, 9, 4, 3, 5\}$

$\langle 5 \rangle = \{5^0, 5^1, 5^2, 5^3, 5^4\} = \{1, 5, 3, 4, 9\}$

$\langle 3 \rangle = \{3^0, 3^1, 3^2, 3^3, 3^4\} = \{1, 3, 9, 5, 4\}$

Modular exponentiation

In cyclic group $(\mathbb{G}, \bullet)$ (e.g., $\mathbb{G} = \mathbb{Z}_p^*$):

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

Naive algorithm: $g^e = \underbrace{g \bullet g \bullet \cdots \bullet g}_{e \text{ times}}$

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

Square-and-multiply algorithm:

Binary representation: $e = \sum_{i=0}^{n} e_i \cdot 2^i$,

with $n = \lfloor \log_2 e \rfloor$ and $e_i = \left\lfloor \frac{e}{2^i} \right\rfloor \bmod 2$

Computation:

$$g^{2^0} := g, \quad g^{2^i} := \left(g^{2^{i-1}}\right)^2$$

$$g^e := \prod_{i=0}^{n} \left(g^{2^i}\right)^{e_i}$$

RtoL_square_and_mult(g, e):

$a := g$
$b := 1$
for $i := 0$ to n do
  if $\lfloor e/2^i \rfloor \bmod 2 = 1$ then
    $b := b \bullet a$ ← multiply
  $a := a \bullet a$ ← square
return b

LtoR_square_and_mult(g, e):

$a := 1$
for $i := n$ downto 0 do
  $a := a^2$
  if $e_i = 1$ then
    $a := a \bullet g$
return a

Safer square-and-multiply algorithms

Basic square-and-multiply algorithms are vulnerable to side-channel attacks (e.g., analysis of unintended power-line or electromagnetic signal emissions by microcontrollers). If an eavesdropper can recognize the function-call sequence

square, multiply, square, square, square, multiply, square, multiply, ...

then that suggests for $\text{LtoR\_square\_and\_mult}$: $e = 10011\ldots$.

There are often faster algorithms for squaring than just multiplying a group element with itself.

These variants are slower (more multiplications), but branch free:

$\text{Square\_and\_mult\_always}(g, e)$:
- $a[0] := 1$
- for $i := n$ downto 0 do
  - $a[0] := a[0]^2$
  - $a[1] := a[0] \bullet g$
  - $a[0] := a[e_i]$
- return $a[0]$

$\text{Montgomery\_Ladder}(g, e)$:
- $a[0] := g$; assuming $e_n = 1$
- $a[1] := g^2$
- for $i := n - 1$ downto 0 do
  - $a[\neg e_i] := a[0] \bullet a[1]$
  - $a[\ e_i] := a[e_i]^2$
- return $a[0]$

Dummy write operations like $a[0] := a[0]$ may still be recognizeable.

Discrete logarithm problem

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

$$g^x = \underbrace{g \bullet g \bullet \cdots \bullet g}_{x \text{ times}} = y$$

Squaring allows use of faster algorithms than multiplication.

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

For cryptographic applications, we require groups with

- a probabilistic polynomial-time group-generation algorithm $\mathcal{G}(1^\ell)$ that outputs a description of $\mathbb{G}$ with $\lceil \log_2 |\mathbb{G}| \rceil = \ell$;
- a description that defines how each element of $\mathbb{G}$ is represented uniquely as a bit pattern;
- efficient (polynomial time) algorithms for $\bullet$, for picking an element of $\mathbb{G}$ uniformly at random, and for testing whether a bit pattern represents an element of $\mathbb{G}$;

Hard discrete logarithm problems

The discrete logarithm experiment $\mathsf{DLog}_{\mathcal{G},\mathcal{A}}(\ell)$:

1. Run $\mathcal{G}(1^\ell)$ to obtain $(\mathbb{G}, q, g)$, where $\mathbb{G}$ is a cyclic group of order q ($2^{\ell-1} < q \leq 2^\ell$) and g is a generator of $\mathbb{G}$
2. Choose uniform $h \in \mathbb{G}$.
3. Give $(\mathbb{G}, q, g, h)$ to $\mathcal{A}$, 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 $\mathcal{G}$" if for all probabilistic polynomial-time algorithms $\mathcal{A}$ there exists a negligible function negl, such that $\mathbb{P}(\mathsf{DLog}_{\mathcal{G},\mathcal{A}}(\ell) = 1) \leq \mathsf{negl}(\ell)$.

Diffie–Hellman problems

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

$$\mathsf{DH}(h_1, h_2) := g^{\log_g h_1 \cdot \log_g h_2}$$

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

These two problems are related to the discrete logarithm problem:

- **Computational Diffie–Hellman (CDH) problem:** the adversary is given uniformly chosen $h_1, h_2 \in \mathbb{G}$ and has to output $\mathsf{DH}(h_1, h_2)$.

  The problem is hard if for all PPT $\mathcal{A}$ we have $\mathbb{P}(\mathcal{A}(\mathbb{G}, q, g, g^x, g^y) = g^{xy}) \leq \mathsf{negl}(\ell)$.

- **Decision Diffie–Hellman (DDH) problem:** the adversary is given $h_1, h_2 \in \mathbb{G}$ chosen uniformly at random, plus another value $h' \in \mathbb{G}$, which is either equal to $\mathsf{DH}(h_1, h_2)$, or was chosen uniformly at random, and has to decide which of the two cases applies.

  The problem is hard if for all PPT $\mathcal{A}$ and uniform $x, y, z \in \mathbb{G}$ we have $|\mathbb{P}(\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1) - \mathbb{P}(\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1)| \leq \mathsf{negl}(\ell)$.

If the discrete-logarithm problem is not hard for $\mathbb{G}$, then neither will be the CDH problem, and if the latter is not hard, neither will be the DDH problem.

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 $\mathbb{G}$ of order q and a generator $g \in \mathbb{G}$, 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:

$$\begin{aligned} A \to B: &\qquad g^x \\ B \to A: &\qquad g^y \end{aligned}$$

Now both can form $(g^x)^y = (g^y)^x = g^{xy}$ 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.

W. Diffie, M.E. Hellman: New Directions in Cryptography. IEEE IT-22(6), 1976-11, pp 644–654.

Discrete logarithm algorithms

Several generic algorithms are known for solving the discrete logarithm problem for any cyclic group $\mathbb{G}$ of order q:

- **Trivial brute-force algorithm:** try all g^i, time $|\langle g \rangle| = \mathrm{ord}(g) \leq q$.
- **Pohlig–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 $\mathbb{G}$ to discrete-logarithm problems for prime-order subgroups of $\mathbb{G}$.
  $\Rightarrow$ the difficulty of finding the discrete logarithm in a group of order q is no greater than that of finding it in a group of order q', where q' is the largest prime factor dividing q.
- **Shank's baby-step/giant-step algorithm:** requires $O(\sqrt{q} \cdot \mathrm{polylog}(q))$ time and $O(\sqrt{q})$ memory.
- **Pollard's rho algorithm:** requires $O(\sqrt{q} \cdot \mathrm{polylog}(q))$ time and $O(1)$ memory.

$\Rightarrow$ choose $\mathbb{G}$ 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. $q \gg 2^{200}$).

Baby-step/giant-step algorithm

Given generator $g \in \mathbb{G}$ ($|\mathbb{G}| = q$) and $y \in \mathbb{G}$, find $x \in \mathbb{Z}_q$ with $g^x = y$.

- Powers of g form a cycle $1 = g^0, g^1, g^2, \ldots, g^{q-2}, g^{q-1}, g^q = 1$, and $y = g^x$ sits on this cycle.
- Go around cycle in "giant steps" of $n = \lfloor \sqrt{q} \rfloor$:

  $$g^0, g^n, g^{2n}, \ldots, g^{\lceil q/n \rceil n}$$

  Store all values encountered in a lookup table $L[g^{kn}] := k$.
  Memory: $\sqrt{q}$, runtime: $\sqrt{q}$, (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: $\sqrt{q}$ (times log. table lookup)
- Now we know $y \cdot g^i = g^{kn}$, therefore $y = g^{kn-i}$ and can return $x := (kn - i) \bmod q = \log_g y$.

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.

There are groups believed to be not vulnerable to the Index Calculus Algorithm, obtained by defining a group operator over points of an *elliptic curve* (EC) with coordinates in $\mathbb{Z}_p$ or $\mathbb{F}_{p^n}$.

Equivalent key lengths: (NIST)

	RSA	Discrete logarithm problem		
private key	factoring $n = pq$	in $\mathbb{Z}_p^*$		in EC
length	modulus n	modulus p	order q	order q
80 bits	1024 bits	1024 bits	160 bits	160 bits
112 bits	2048 bits	2048 bits	224 bits	224 bits
128 bits	3072 bits	3072 bits	256 bits	256 bits
192 bits	7680 bits	7680 bits	384 bits	384 bits
256 bits	15360 bits	15360 bits	512 bits	512 bits

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 $p = qr + 1$ for $r \in \mathbb{N}$
2. Choose $1 < h < p$ with $h^r \bmod p \neq 1$
3. Use $g := h^r \bmod p$ as generator for $\mathbb{G} = \langle g \rangle = \{h^r \bmod p | h \in \mathbb{Z}_p^*\}$

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 $g^x \bmod p$ (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 184 where $r = 2$.

Schnorr groups (proofs)

Let $p = rq + 1$ with p, q prime and $\mathbb{G} = \{h^r \bmod p | h \in \mathbb{Z}_p^*\}$. 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 $x^r y^r \bmod p = (xy)^r \bmod p = (xy \bmod p)^r \bmod p \in \mathbb{G}$ as $(xy \bmod p) \in \mathbb{Z}_p^*$.
   In addition, $\mathbb{G}$ includes the neutral element $1^r = 1$
   For each h^r, it also includes the inverse element $(h^{-1})^r \bmod p$.

2. $\mathbb{G}$ has $q = (p-1)/r$ elements.

   Proof: The idea is to show that the function $f_r : \mathbb{Z}_p^* \to \mathbb{G}$ with $f_r(x) = x^r \bmod p$ is an r-to-1 function, and then since $|\mathbb{Z}_p^*| = p - 1$ this will show that $|\mathbb{G}| = q = (p-1)/r$.
   Let g be a generator of $\mathbb{Z}_p^*$ such that $\{g^0, g^1, \ldots, g^{p-2}\} = \mathbb{Z}_p^*$. Under what condition for i, j is $(g^i)^r \equiv (g^j)^r \pmod p$? $(g^i)^r \equiv (g^j)^r \pmod p \Leftrightarrow ir \equiv jr \pmod{p-1} \Leftrightarrow (p-1)|(ir - jr) \Leftrightarrow rq|(ir - jr) \Leftrightarrow q|(i-j)$.
   For any fixed $j \in \{0, \ldots, p-2\} = \mathbb{Z}_{p-1}$, what values of $i \in \mathbb{Z}_{p-1}$ fulfill the condition $q|(i-j)$, and how many such values i are there? For each j, there are exactly the r different values $i \in \{j, j+q, j+2q, \ldots, j+(r-1)q\}$ in $\mathbb{Z}_{p-1}$, as $j + rq \equiv j \pmod{p-1}$. 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: $h^r \in \mathbb{G}$ (by definition) and $|\mathbb{G}|$ prime $\Rightarrow \mathrm{ord}_{\mathbb{G}}(h^r) \in \{1, |\mathbb{G}|\}$ (Lagrange).

4. $h \in \mathbb{G} \Leftrightarrow h \in \mathbb{Z}_p^* \wedge h^q \bmod p = 1$. (Useful security check!)

   Proof: Let $h = g^i$ with $\langle g \rangle = \mathbb{Z}_p^*$ and $0 \leq i < p - 1$. Then
   $h^q \bmod p = 1 \Leftrightarrow g^{iq} \bmod p = 1 \Leftrightarrow iq \bmod (p-1) = 0 \Leftrightarrow rq|iq \Leftrightarrow r|i$.

Katz/Lindell (2nd ed.), section 8.3.3

Elliptic curves – the Weierstrass equation

An *elliptic curve* E over a field $\mathbb{K}$ is defined by the *Weierstrass equation*

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

with coefficients $a_1, a_2, a_3, a_4, a_6 \in \mathbb{K}$ such that $\Delta(a_1, a_2, a_3, a_4, a_6) \neq 0$.

The *discriminant* Δ of E is defined as $\Delta = -d_2^2 d_8 - 8d_4^3 - 27d_6^2 + 9d_2 d_4 d_6$ with $d_2 = a_1^2 + 4a_2$, $d_4 = 2a_4 + a_1a_3$, $d_6 = a_3^2 + 4a_6$, and $d_8 = a_1^2 a_6 + 4a_2 a_6 - a_1 a_3 a_4 + a_2 a_3^2 - a_4^2$.
If $\Delta \neq 0$ then the curve is *smooth*, i.e. it has no points with more than one tangent.

If $\mathbb{L}$ is any extension field of $\mathbb{K}$, then the set of ("$\mathbb{L}$-rational") points on curve E is defined as

$$\mathbb{E}(\mathbb{L}) = \{(x, y) \in \mathbb{L} \times \mathbb{L} : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\mathcal{O}\}$$

The additional element $\mathcal{O}$ is called the "point at infinity".

It will act as the neutral element when we define a group structure over $\mathbb{E}(\mathbb{L})$.

Elliptic curves were originally studied over the fields $\mathbb{C}$, $\mathbb{R}$, and $\mathbb{Q}$, in the context of elliptic integrals. In cryptography, they are used instead over finite fields, in particular $\mathbb{K} = \mathbb{L} = \mathbb{Z}_p$ as well as $\mathbb{K} = \mathbb{L} = \mathbb{F}_{2^n}$.

Elliptic curves – simplified Weierstrass equations

An elliptic curve defined over a field $\mathbb{K}$ by the Weierstrass equation

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

can be turned into an equivalent (isomorphic) one by changing variables:

$$(x, y) \mapsto \left(\frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1x}{216} - \frac{a_1^3 + 4a_1a_2 - 12a_3}{24} \right)$$

This simplifies the curve equation significantly:

$$y^2 = x^3 + ax + b$$

where $a, b \in \mathbb{K}$ and $\Delta = -16(4a^3 + 27b^3) \neq 0$.

However, due to the divisions by $36 = 2^2 3^3$, $216 = 2^3 3^3$ and $24 = 2^3 3$ in the above change of variables, this trick does not work (would lead to division by zero) if the characteristic of $\mathbb{K}$ is 2 or 3 (i.e., if $1 + 1 = 0$ or $1 + 1 + 1 = 0$).

Simplified Weierstrass equations if $1 + 1 = 0$

If $\mathbb{K}$ has characteristic 2, two other changes of variable can be used to simplify the Weierstrass equation:

- If $a_1 \neq 0$ then

$$(x, y) \mapsto \left(a_1^2 x + \frac{a_3}{a_1}, a_1^3 y + \frac{a_1^2 a_4 + a_3^2}{a_1^3} \right)$$

leads to the "non-supersingular" curve

$$y^2 + xy = x^3 + ax^2 + b$$

with discriminant $\Delta = b \neq 0$.

- If $a_1 = 0$ then

$$(x, y) \mapsto (x + a_2, y)$$

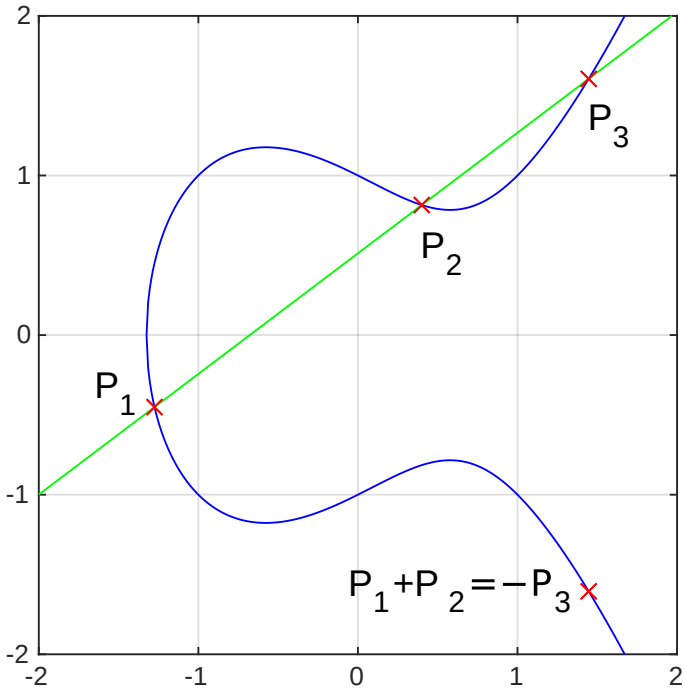
leads to the "supersingular" curve

$$y^2 + cy = x^3 + ax + b$$

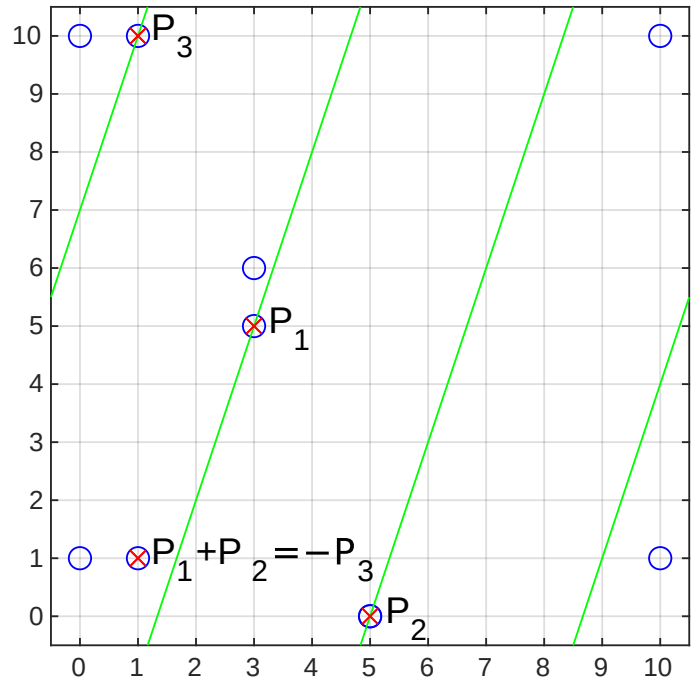
with $a, b, c \in \mathbb{K}$ and $\Delta = c^4 \neq 0$.

Similar tricks exist for $\mathbb{K}$ with characteristic 3, but such $\mathbb{K}$ are not commonly used in cryptography.

Elliptic-curve group operation



elliptic curve over $\mathbb{R}$ ($a = -1$, $b = 1$)



elliptic curve over $\mathbb{Z}_{11}$ ($a = -1$, $b = 1$)

Elliptic curves over $\mathbb{R}$ or $\mathbb{Z}_p$ ($p > 3$) are sets of 2-D coordinates (x, y) with

$$y^2 = x^3 + ax + b \qquad \text{where } 4a^3 + 27b^2 \neq 0$$

plus one additional "point at infinity" $\mathcal{O}$.

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: $\mathcal{O}$ – intersects any vertical line. Inverse: $-(x, y) = (x, -y)$

Curve compression: for any given x, encoding y requires only one bit

Elliptic-curve group operation over $\mathbb{E}(\mathbb{Z}_p)$

$\mathbb{E}(\mathbb{Z}_p) = \{(x, y) \mid x, y \in \mathbb{Z}_p \text{ and } y^2 \equiv x^3 + ax + b \pmod{p}\} \cup \{\mathcal{O}\}$
where $p > 3$ prime, parameters $a, b \in \mathbb{Z}_p$ with $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$.

- Neutral element: $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in \mathbb{E}(\mathbb{Z}_p)$
- Negation: if $P = (x, y)$ then $-P = (x, -y)$ since $P - P = \mathcal{O}$; $-\mathcal{O} = \mathcal{O}$
- Addition: for $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, $P_1, P_2 \neq \mathcal{O}$, $x_1 \neq x_2$:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \qquad \text{line slope}$$

$$y = m \cdot (x - x_1) + y_1 \qquad \text{line equation}$$

$$\big(m \cdot (x - x_1) + y_1\big)^2 = x^3 + ax + b \qquad \text{intersections}$$

$$x_3 = m^2 - x_1 - x_2 \qquad \text{third-point solution}$$

$$y_3 = m \cdot (x_3 - x_1) + y_1$$

$$(x_1, y_1) + (x_2, y_2) = (m^2 - x_1 - x_2, m \cdot (x_1 - x_3) - y_1) \qquad \text{(all of this mod } p\text{)}$$

If $x_1 = x_2$ but $y_1 \neq y_2$ then $P_1 = -P_2$ and $P_1 + P_2 = \mathcal{O}$.

- Doubling:
  If $P_1 = P_2$ and $y_1 = 0$ then $P_1 + P_2 = 2P_1 = \mathcal{O}$.
  If $P_1 = P_2$ and $y_1 \neq 0$ then add using tangent $m = (3x_1^2 + a)/2y_1$.

Non-supersingular curve group operation over $\mathbb{E}(\mathbb{F}_{2^n})$

$\mathbb{E}(\mathbb{F}_{2^n}) = \{(x, y) \mid x, y \in \mathbb{F}_{2^n} \text{ and } y^2 + xy = x^3 + ax^2 + b\} \cup \{\mathcal{O}\}$
where parameters $a, b \in \mathbb{F}_{2^n}$ with $b \neq 0$.

- Neutral element: $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in \mathbb{E}(\mathbb{F}_{2^n})$
- Negation: if $P = (x, y)$ then $-P = (x, x + y)$; $-\mathcal{O} = \mathcal{O}$
- Addition: for $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, $P_1, P_2 \neq \mathcal{O}$, $x_1 \neq x_2$:

$$m = \frac{y_1 + y_2}{x_1 + x_2}$$
$$x_3 = m^2 + m + x_1 + x_2 + a$$
$$y_3 = m \cdot (x_1 + x_3) + x_3 + y_1$$
$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$$

  If $x_1 = x_2$ but $y_1 \neq y_2$ then $P_1 = -P_2$ and $P_1 + P_2 = \mathcal{O}$.
- Doubling (i.e. $P_1 = P_2 = P$):
  If $y_1 = 0$ then $P = -P$, i.e. $P_1 + P_2 = 2P = \mathcal{O}$.
  If $y_1 \neq 0$ then $P \neq -P$ and add using tangent $m = x_1 + y_1/x_1$:

$$x_3 = m^2 + m + a = x_1^2 + \frac{b}{x_1^2} \qquad \text{and} \qquad y_3 = x_1^2 + mx_3 + x_3$$

Projective coordinates for points on $\mathbb{E}(\mathbb{F}_{2^n})$

When points P_1 and P_2 are represented as "affine" coordinates (x, y) in curve equation $y^2 + xy = x^3 + ax^2 + b$, the point addition and doubling operations involve expensive field divisions.

Several other "projective" 3D coordinate systems (X, Y, Z) have been proposed that make the group operation cheaper, by avoiding division:

- Standard projective coordinates: $(x, y) = (X/Z, Y/Z)$

  Curve: $Y^2Z + XYZ = X^3 + aX^2Z + bZ^3$,
  $\mathcal{O} = (0, 1, 0)$ and $-(X, Y, Z) = (X, X + Y, Z)$.

- Jacobian projective coordinates: $(x, y) = (X/Z^2, Y/Z^3)$

  Curve: $Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6$,
  $\mathcal{O} = (1, 1, 0)$ and $-(X, Y, Z) = (X, X + Y, Z)$.

- López-Dahab (LD) projective coordinates: $(x, y) = (X/Z, Y/Z^2)$

  Curve: $Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$,
  $\mathcal{O} = (1, 0, 0)$ and $-(X, Y, Z) = (X, X + Y, Z)$.

For $Z = 1$ projective and affine coordinates are identical, i.e. $(x, y) = (X, Y, 1)$.

Equivalent projective coordinate systems also exist for $\mathbb{E}(\mathbb{Z}_p)$: standard $(x, y) = (X/Z, Y/Z)$, Jacobian $(x, y) = (X/Z^2, Y/Z^3)$ and Chudnovsky (like Jacobian, but also store Z^2 and Z^3).

These all have slightly different performance trade-offs regarding the number of field additions, multiplications and division required for point add and double operations.

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: $p + 1 - 2\sqrt{p} \leq |\mathbb{E}(\mathbb{Z}_p, a, b)| \leq p + 1 + 2\sqrt{p}$

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.

E.g. Schoof's algorithm for $\mathbb{E}(\mathbb{Z}_p)$ and Satoh's algorithm for $\mathbb{E}(\mathbb{F}_{2^n})$.

Generate a cyclic elliptic-curve group (p, q, a, b, G) with:

1. Choose n-bit prime p
2. Choose $a, b \in \mathbb{Z}_p$ with $4a^3 + 27b^2 \neq 0 \pmod{p}$, determine $q = |\mathbb{E}(\mathbb{Z}_p, a, b)|$, repeat until q is an n-bit prime
3. Choose $G \in \mathbb{E}(\mathbb{Z}_p, a, b) \setminus \{\mathcal{O}\}$ 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 $y = \sqrt{f(x)}$.

Elliptic-curve discrete-logarithm problem

The elliptic-curve operation is traditionally written as an additive group, so the "exponentiation" of the elliptic-curve discrete-logarithm problem (ECDLP) becomes multiplication:

$$x \cdot G = \underbrace{G + G + \cdots + G}_{x \text{ times}} \qquad x \in \mathbb{Z}_q$$

So the square-and-multiply algorithm becomes double-and-add, and Diffie–Hellman becomes $\mathsf{DH}(x \cdot G, y \cdot G) = xy \cdot G$ for $x, y \in \mathbb{Z}_q^*$.

Many curve parameters and cyclic subgroups for which ECDLP is believed to be hard have been proposed or standardised.

Example: NIST P-256

$$\begin{aligned}
p &= \texttt{0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff} \\
q &= \texttt{0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551} \\
a &= 3 \\
b &= \texttt{0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b} \\
G &= (\texttt{0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296}, \\
&\qquad \texttt{0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5})
\end{aligned}$$

Note: $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$ and $q \approx 2^{256} - 2^{224} + 2^{192}$ here are *generalized* resp. *pseudo Mersenne primes*, for fast mod calculation on 32-bit CPUs and good use of the 256-bit space.

Commonly used standard curves

NIST FIPS 186-4 has standardized five such elliptic curves over integer field ($\mathbb{Z}_p$) coordinates: **P-192, P-224, P-256, P-384, P-521**.

Also: five random curves of the form $y^2 + xy = x^3 + x^2 + b$ over binary field ($\mathbb{F}_{2^n}$) 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 $y^2 + xy = x^3 + ax^2 + 1$ ($a \in \{0, 1\}$, with cofactors 4 or 2, resp.), also over $\mathbb{F}_{2^n}$:
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. `https://safecurves.cr.yp.to/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 $\mathbb{Z}_p$ (**secp{192,224,256}{k,r}1, secp{384,521}r1**) and 12 over $\mathbb{F}_{2^n}$ (**sect163k1,...,sect571r1**). (Vers. 2.0 dropped smaller **secp{112,128,160}r{1,2}**, **secp160k1** and **sect{113,131}r{1,2}**.)

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 $((\mathbb{G}, \cdot), q, g)$ are fixed for all participants.

A chooses *secret key* $x \in \mathbb{Z}_q^*$ and publishes $g^x \in \mathbb{G}$ as her *public key*. B generates for each message a new nonce $y \in \mathbb{Z}_q^*$ and then sends

$$B \to A: \qquad g^y,\ (g^x)^y \cdot M$$

where $M \in \mathbb{G}$ is the message that B sends to A in this asymmetric encryption scheme. Then A calculates

$$[(g^x)^y \cdot M] \cdot [(g^y)^{q-x}] = M$$

to decrypt M.

In practice, this scheme is rarely used because of the difficulty of fitting M into $\mathbb{G}$. Instead, B only sends g^y. Then both parties calculate $K = h(g^{xy})$ and use that as the private session key for an efficient blockcipher-based authenticated encryption scheme that protects the confidentiality and integrity of the bulk of the message M:

$$B \to A: \qquad g^y,\ \mathrm{Enc}_K(M)$$

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 $a = g^x$.
    $\rightarrow$ Discrete Logarithm Problem
  - given values $g^x, g^y \in \mathbb{Z}_p^*$: find g^{xy}.
    $\rightarrow$ Computational Diffie–Hellman Problem
  - given values $g^x, g^y, z \in \mathbb{Z}_p^*$: tell whether $z = g^{xy}$.
    $\rightarrow$ Decision Diffie–Hellman Problem
- given a random $n = p \cdot q$, where p and q are ℓ-bit primes ($\ell \geq 1024$):
  - find integers p and q such that $n = p \cdot q$ in $\mathbb{N}$
    $\rightarrow$ Factoring Problem
  - given a polynomial $f(x)$ of degree > 1:
    find $x \in \mathbb{Z}_n$ such that $f(x) = 0$ in $\mathbb{Z}_n$ (if p and q are unknown)

"Textbook" RSA encryption

Key generation

- Choose random prime numbers p and q (each ≈ 1024 bits long)
- $n := pq$ (≈ 2048 bits = key length) $\varphi(n) = (p-1)(q-1)$
- pick integer values e, d such that: $ed \bmod \varphi(n) = 1$
- public key $PK := (n, e)$
- secret key $SK := (n, d)$

Encryption

- input plaintext $M \in \mathbb{Z}_n^*$, public key (n, e)
- $C := M^e \bmod n$

Decryption

- input ciphertext $C \in \mathbb{Z}_n^*$, secret key (n, d)
- $M := C^d \bmod n$

In $\mathbb{Z}_n$: $(M^e)^d = M^{ed} = M^{ed \bmod \varphi(n)} = M^1 = M$.

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 $C := M^e \bmod n$:

- deterministic encryption: cannot be CPA secure
- malleability:
  - adversary intercepts C and replaces it with $C' := X^e \cdot C$
  - recipient decrypts $M' = \mathsf{Dec}_{SK}(C') = X \cdot M \bmod n$
- chosen-ciphertext attack recovers plaintext:
  - adversary intercepts C and replaces it with $C' := R^e \cdot C \bmod n$
  - decryption oracle provides $M' = \mathsf{Dec}_{SK}(C') = R \cdot M \bmod n$
  - adversary recovers $M = M' \cdot R^{-1} \bmod n$
- Small value of M (e.g., 128-bit AES key), small exponent $e = 3$:
  - if $M^e < n$ then $C = M^e \bmod n = M^e$ and then $M = \sqrt[3]{C}$ can be calculated efficiently in $\mathbb{Z}$ (no modular arithmetic!)
- many other attacks exist ...

Trapdoor permutations

A **trapdoor permutation** is a tuple of polynomial-time algorithms $(\mathsf{Gen}, F, F^{-1})$ such that

- the **key generation algorithm** Gen receives a security parameter ℓ and outputs a pair of keys $(PK, SK) \leftarrow \mathsf{Gen}(1^\ell)$, 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 $x := F^{-1}_{SK}(y) \in \mathcal{X}$;
- for all ℓ, $(PK, SK) \leftarrow \mathsf{Gen}(1^\ell)$, $x \in \mathcal{X}$: $F^{-1}_{SK}(F_{PK}(x)) = x$.

In practice, the domain $\mathcal{X}$ may depend on PK.

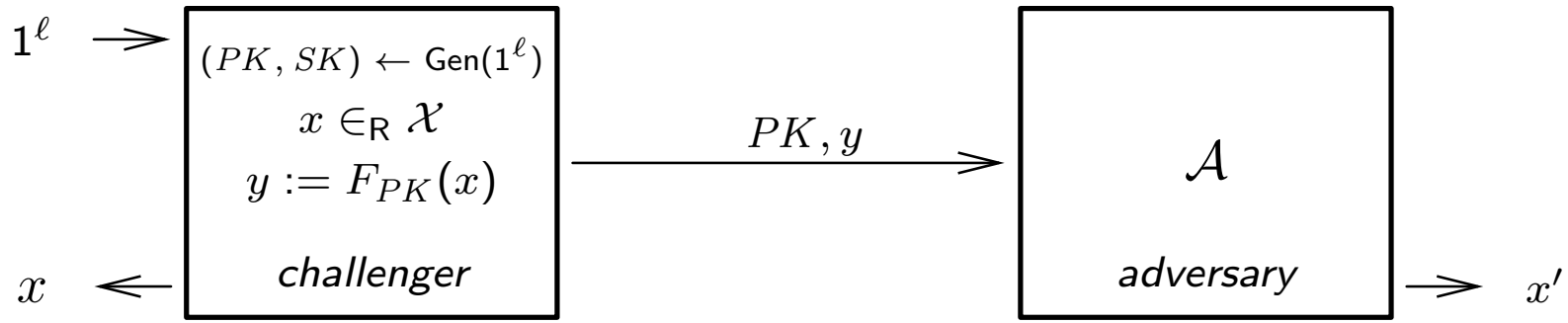
This looks almost like the definition of a public-key encryption scheme, the difference being

- F is deterministic;
- the associated security definition.

Secure trapdoor permutations

Trapdoor permutation: $\Pi = (\mathsf{Gen}, F, F^{-1})$

Experiment/game $\mathsf{TDInv}_{\mathcal{A},\Pi}(\ell)$:



1. The challenger generates a key pair $(PK, SK) \leftarrow \mathsf{Gen}(1^\ell)$ and a random value $x \in_R \mathcal{X}$ 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: $\mathsf{TDInv}_{\mathcal{A},\Pi}(\ell) = 1$.

A trapdoor permutation Π is secure if for all probabilistic polynomial time adversaries $\mathcal{A}$ the probability of success $\mathbb{P}(\mathsf{TDInv}_{\mathcal{A},\Pi}(\ell) = 1)$ 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).

Public-key encryption scheme from trapdoor permutation

Trapdoor permutation: $\Pi_{\mathsf{TD}} = (\mathsf{Gen}_{\mathsf{TD}}, F, F^{-1})$ with $F_{PK} : \mathcal{X} \leftrightarrow \mathcal{X}$
Authentic. encrypt. scheme: $\Pi_{\mathsf{AE}} = (\mathsf{Gen}_{\mathsf{AE}}, \mathsf{Enc}, \mathsf{Dec})$ with key space $\mathcal{K}$
Secure hash function $h : \mathcal{X} \to \mathcal{K}$

We define the public-key encryption scheme $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$:

- Gen': output key pair $(PK, SK) \leftarrow \mathsf{Gen}_{\mathsf{TD}}(1^\ell)$
- Enc': on input of plaintext message M, generate random $x \in_{\mathsf{R}} \mathcal{X}$, $y = F_{PK}(x)$, $K = h(x)$, $C \leftarrow \mathsf{Enc}_K(M)$, output ciphertext (y, C);
- Dec': on input of ciphertext message $C = (y, C)$, recover $K = h(F_{SK}^{-1}(y))$, output $\mathsf{Dec}_K(C)$

Encrypted message: $F_{PK}(x), \mathsf{Enc}_{h(x)}(M)$

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 $\mathsf{PubK}^{\mathsf{cca}}_{\mathcal{A},\Pi'}$ 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 Π' is CCA secure.

Using RSA as a CCA-secure encryption scheme

Solution 1: use only as trapdoor permutation to build encryption scheme

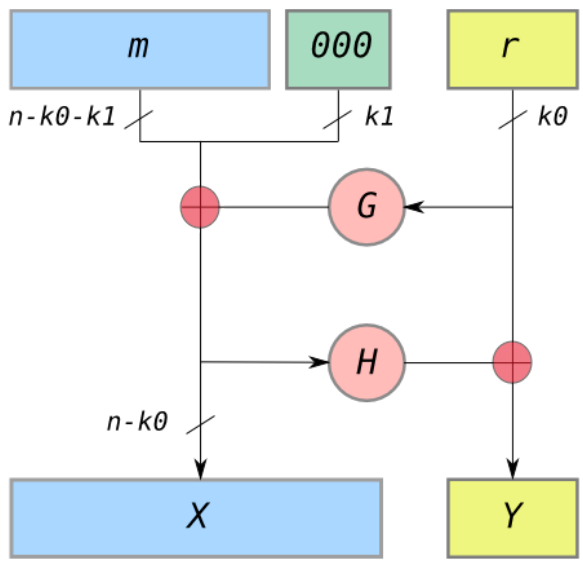
- Pick random value $x \in \mathbb{Z}_n^*$
- Ciphertext is $(x^e \bmod n, \mathrm{Enc}_{h(x)}(M))$, where Enc is from an authenticated encryption scheme

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 $C := M'^e \bmod n$.



Wikipedia/Ozga

PKCS #1 v2.0

Practical pitfalls with implementing RSA

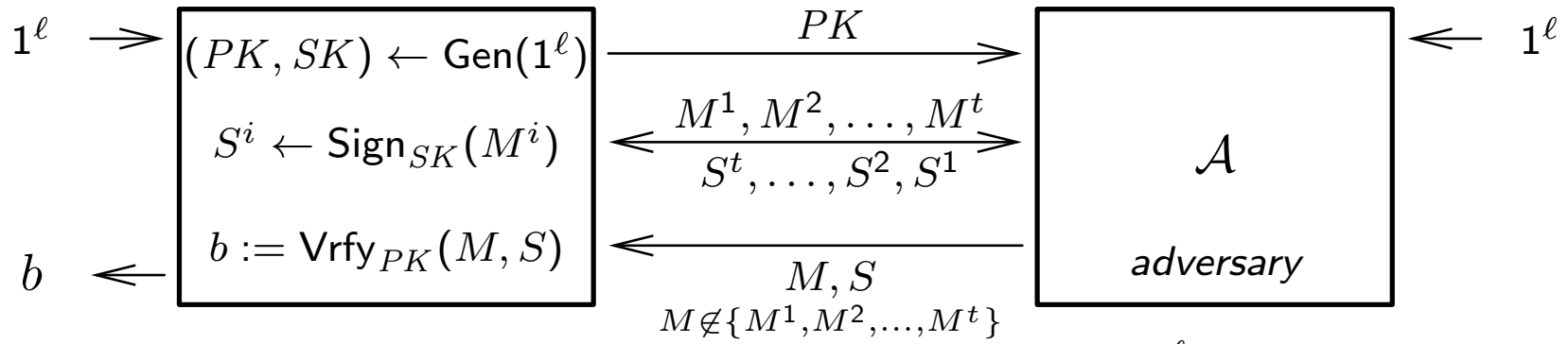
- low entropy of random-number generator seed when generating p and q (e.g. in embedded devices):
  - take public RSA modulus n_1 and n_2 from two devices
  - test $\gcd(n_1, n_2) \stackrel{?}{=} 1 \Rightarrow$ if no, n_1 and n_2 share this number as a common factor
  - February 2012 experiments: worked for many public HTTPS keys

    Lenstra et al.: Public keys, CRYPTO 2012
    Heninger et al.: Mining your Ps and Qs, USENIX Security 2012.

Digital signature schemes: existential unforgeability

Signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$, $\mathcal{M} = \{0,1\}^*$, security parameter ℓ.

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



1. challenger generates key pair $(PK, SK) \leftarrow \mathsf{Gen}(1^\ell)$
2. adversary $\mathcal{A}$ is given PK and oracle access to $\mathsf{Sign}_{SK}(\cdot)$; let $\mathcal{Q} = \{M^1, \ldots, M^t\}$ denote the set of queries that $\mathcal{A}$ asks the oracle
3. adversary outputs (M, S)
4. the experiment outputs 1 if $\mathsf{Vrfy}_K(M, S) = 1$ and $M \notin \mathcal{Q}$

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

$$\mathbb{P}(\text{Sig-forge}_{\mathcal{A},\Pi}(\ell) = 1) \leq \mathsf{negl}(\ell)$$

One-time signatures

A simple digital signature scheme can be built using a one-way function h (e.g., secure hash function):

Secret key: $2n$ random bit strings $R_{i,j}$ $(i \in \{0,1\}, 1 \leq j \leq n)$

Public key: $2n$ bit strings $h(R_{i,j})$

Signature: $(R_{b_1,1}, R_{b_2,2}, \ldots, R_{b_n,n})$, where $h(M) = b_1 b_2 \ldots b_n$

Problem: Can only be used once (i.e., provides no existential unforgability if $t > 1$ oracle queries are allowed).

RSA signatures

Basic idea: $n = pq$, $ed \equiv 1 \pmod{\phi(n)}$, $PK = (e, n)$, $SK = (d, n)$

$$S = \mathsf{Sign}_{SK}(M) := M^d \bmod n$$

$$\mathsf{Vrfy}_{PK}(M, S) := (S^e \bmod n \stackrel{?}{=} M)$$

This "textbook" RSA signature, where adversary has free choice of message $M \in \mathbb{Z}_n^*$, is completely insecure (no existential unforgability):

- No-message attack: pick any S and present (M, S) with $M := S^e \bmod n$ to challenger
- Choose message M, factor it into $M = M_1 M_2$, query oracle for signatures $S_1 \equiv M_1^d$, $S_2 \equiv M_2^d$, present $(M, S_1 S_2 \bmod n)$
- If M and e are small (e.g., $e = 3$, $M < 2^{256}$ SHA-256 hash, $\lceil \log_2 n \rceil = 2048$), then $M^e < n$ and $S = \sqrt[e]{M}$ may be integer

Solution: RSA with full-domain hashing (RSA-FDH, PKCS #1 v2.1). Use a collision-resistant $H : \{0,1\}^* \to \mathbb{Z}_n^*$ and $S := [H(M)]^d \bmod n$.

There is also RSA-PSS which adds a "salt" value for randomization.

Schnorr identification scheme

Cyclic group $\mathbb{G}$ of prime order q with generator g, DLOG hard

Secret key: $x \in_R \mathbb{Z}_q^*$, public key: $y = g^x$

Prover P picks $k \in_R \mathbb{Z}_q$, Verifier V picks $r \in_R \mathbb{Z}_q$

$$\begin{aligned} P \to V: &\qquad I := g^k \\ V \to P: &\qquad r \\ P \to V: &\qquad s := (rx + k) \bmod q \end{aligned}$$

Verifier checks

$$g^s \cdot y^{-r} \stackrel{?}{=} I$$

Works because:

$$g^s \cdot y^{-r} = g^{rx+k} \cdot (g^x)^{-r} = g^{rx+k-rx} = g^k = I$$

As secure as DLOG: an attacker who can find s_1, s_2 for two challenges r_1, r_2 with same I could also calculate any discrete logarithm:

$$\begin{aligned} g^{s_1} \cdot y^{-r_1} = I = g^{s_2} \cdot y^{-r_2} \\ g^{s_1 - s_2} = y^{r_1 - r_2} = g^{x(r_1 - r_2)} \\ \log_g y = (s_1 - s_2)(r_1 - r_2)^{-1} \bmod q \end{aligned}$$

Schnorr signature scheme

Idea ("Fiat-Shamir transform"): a prover can run the identification protocol itself, by generating the challenge r from I using a secure hash function $H : \{0,1\}^* \to \mathbb{Z}_q$, and that can then also incorporate the message M to be signed:

$$r := H(I, M)$$

Secret key $x \in_R \mathbb{Z}_q^*$ and public key $y = g^x \in \mathbb{G}$ in some DLOG hard cyclic group, as before.

Sign: on input of a message $M \in \{0,1\}^*$, generate signature (r, s) with

$$\begin{aligned} k &\in_R \mathbb{Z}_q \\ I &:= g^k \\ r &:= H(I, M) \\ s &:= rx + k \bmod q \end{aligned}$$

Verify: on input of message M and signature (r, s), compute

$$I := g^s \cdot y^{-r}$$

and output 1 iff $H(I, M) = r$.

Digital Signature Algorithm (DSA)

Let $(\mathbb{G}, q, g)$ be system-wide choices of a cyclic group $\mathbb{G}$ of order q with generator g. In addition, we need two functions $H : \{0,1\}^* \to \mathbb{Z}_q$ and $F : \mathbb{G} \to \mathbb{Z}_q$ 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 $y := g^x \in \mathbb{G}$.

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

$$k \cdot s - x \cdot r \equiv H(m) \pmod{q} \tag{1}$$

for $s := k^{-1} \cdot (H(m) + xr) \bmod q$. If $r = 0$ or $s = 0$, restart with a fresh k, otherwise output $\mathrm{Sign}_x(m) \leftarrow (r, s)$.

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:

$$g^{ks}/g^{xr} = g^{H(m)} \tag{2}$$

$$(g^k)^s = g^{H(m)} y^r \tag{3}$$

$$g^k = g^{H(m)s^{-1}} y^{rs^{-1}} \tag{4}$$

$$\Longrightarrow \text{ actually verify:} \qquad r \stackrel{?}{=} F\big(g^{H(m)s^{-1}} y^{rs^{-1}}\big) \tag{5}$$

DSA variants

ElGamal signature scheme

The DSA idea was originally proposed by ElGamal with $\mathbb{G} = \mathbb{Z}_p^*$, $\mathrm{ord}(g) = q = p - 1$ and $F(x) = x$.

Unless the p and g are chosen more carefully, ElGamal signatures can be vulnerable to forgery:
D. Bleichenbacher: Generating ElGamal signatures without knowing the secret key.
EUROCRYPT '96. `https://link.springer.com/chapter/10.1007/3-540-68339-9_2`

NIST DSA

In 1993, the US government standardized the Digital Signature Algorithm, a modification of the ElGamal signature scheme where

- $\mathbb{G}$ is a prime-order subgroup of $\mathbb{Z}_p^*$
- prime number p (1024 bits), prime number q (160 bits) divides $p - 1$
- $g = h^{(p-1)/q} \bmod p$, with $1 < h < p - 1$ so that $g > 1$ (e.g., $h = 2$)
- H is SHA-1
- $F(x) = x \bmod q$

Generate key: random $0 < x < q$, $y := g^x \bmod p$.
Signature $(r, s) := \mathrm{Sign}_x(m)$: random $0 < k < q$,
$r := (g^k \bmod p) \bmod q$, $s := (k^{-1}(H(m) + x \cdot r)) \bmod q$

Later versions of the DSA standard FIPS 186 added larger values for (p, q, g), as well as ECDSA, where $\mathbb{G}$ is one of several elliptic-curve groups over $\mathbb{Z}_p$ or $\mathbb{F}_{2^n}$ and $F((x, y)) = x \bmod q$.

Elliptic-Curve Digital Signature Algorithm (ECDSA)

System-wide domain parameters:
order q of finite field $\mathbb{F}_q$, a representation of $\mathbb{F}_q$, curve parameters a and b, base point $P = (x_P, y_P) \in \mathbb{E}(\mathbb{F}_q)$, prime order n of P, cofactor $h = |\mathbb{E}(\mathbb{F}_q)|/n$.

ECDSA_KeyGen: secret key $d \in_R \mathbb{Z}_n^*$, public key $Q := dP$

ECDSA_Sign(m, d):
select $k \in_R \mathbb{Z}_n^*$
$(x_1, y_1) := kP$, convert x_1 to integer $\bar{x}_1$
$r := \bar{x}_1 \bmod n$, if $r = 0$ restart with new k
$e := H(m)$
$s := k^{-1}(e + dr) \bmod n$, if $s = 0$ restart
return (r, s)

ECDSA_Vrfy(m, r, s, Q):
reject unless $r, s \in \mathbb{Z}_n^*$
$e := H(m)$
$w := s^{-1} \bmod n$, $u_1 := ew \bmod n$, $u_2 := rw \bmod n$
$X := u_1 P + u_2 Q$, reject if $X = \mathcal{O}$
$v := \bar{x}_1 \bmod n$ where $\bar{x}_1$ is x_1 of X converted to integer
accept signature if $v = r$, otherwise reject signature

Fail0verflow Obtains P ×

www.tomsguide.com/print/PlayStation-Console-Private-Cryptography-Key-fail0verflow-linux,news-9542.html

tom's GUIDE

Fail0verflow Obtains PS3 Cryptography Key

By Kevin Parrish , DECEMBER 31, 2010 3:10 PM - Source: Engadget

Wednesday during the 27th annual Chaos Communication Conference, the team behind the Wii's Homebrew Channel-- fail0verflow-- revealed that they figured out the PlayStation 3's private cryptography key. This means hackers could have full access to the console without the need for a USB device or actual software/hardware hacking.

Typically the "magic password" is used by Sony to authorize the execution of code on the gaming console. Now Sony's key is revealed, hackers can develop hack-free apps and games-- literally signing their code--to execute on the PlayStation 3 as if they're licensed developers.

"It's not an exploit, it's an Epic Fail by Sony," the team said during a live demo. "The PS3 is fine. They screwed up in HQ. They gave us their private key basically. They leave their private key mathematically, so we don't have to exploit anything, we just sign things."

According to reports, Sony didn't bother to generate random numbers to secure the key's secrecy. With that said, the fail0verflow team plans to release tools next month that will take advantage of the security flaw. However the tools aren't intended to enable PlayStation 3 piracy. Instead, they'll re-enable the installation of Linux on every unit sold no matter the firmware-- even v3.55 and beyond.

"Yes, we'll release all our tools as soon as we cleaned them up in January or so," the group said via Twitter.

To see the live demo, check out the video pasted below:

PS3 Private Key Exposed

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 $r = r' = F(g^k)$):

$$\begin{aligned} s &\equiv k^{-1} \cdot (H(m) + xr) \\ s' &\equiv k^{-1} \cdot (H(m') + xr) \\ s - s' &\equiv k^{-1} \cdot (H(m) - H(m')) \\ k &\equiv \big(H(m) - H(m')\big)\big(s - s'\big)^{-1} \pmod{q} \end{aligned}$$

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. $k :=$ 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*

$$\mathsf{Cert}_C(A) = (A, PK_A, T, L, N, \mathsf{Sign}_{SK_C}(A, PK_A, T, L, N))$$

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 $\mathsf{Cert}_C(A)$ 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 $\mathsf{Cert}_C(A)$ with the certification authority public key PK_C:

$$PK_C \bullet \mathsf{Cert}_C(A) = \begin{cases} PK_A & \text{if certificate valid} \\ \text{failure} & \text{otherwise} \end{cases}$$

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*:

$$PK_{C_1} \bullet \mathsf{Cert}_{C_1}(C_2) \bullet \mathsf{Cert}_{C_2}(C_3) \bullet \cdots \bullet \mathsf{Cert}_{C_{n-1}}(C_n) \bullet \mathsf{Cert}_{C_n}(B) = PK_B$$

A has received directly a trustworthy copy of PK_{C_1} (which many implementations store locally as a certificate $\mathsf{Cert}_A(C_1)$ 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 $\mathsf{Cert}_{C_1}(C_2)$, who confirmed the national network operator C_3, who confirmed the IT department of B's employer C_3 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.

Example of an X.509 certificate

```
$ openssl s_client -showcerts www.cst.cam.ac.uk:443 </dev/null >cst.crt
depth=2 C = BM, O = QuoVadis Limited, CN = QuoVadis Root CA 2 G3
depth=1 C = BM, O = QuoVadis Limited, CN = QuoVadis Global SSL ICA G3
depth=0 C = GB, ST = Cambridgeshire, L = CAMBRIDGE, O = University of Cambridge, OU = UIS, CN = www.cst.cam.ac.uk
$ openssl x509 -text -noout -in cst.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            03:07:fb:5f:76:c8:1d:8f:7e:e3:5e:d6:ab:71:5d:a5:1a:c0:e6:70
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = BM, O = QuoVadis Limited, CN = QuoVadis Global SSL ICA G3
        Validity
            Not Before: Sep 13 15:33:05 2017 GMT
            Not After : Sep 13 15:43:00 2020 GMT
        Subject: C = GB, ST = Cambridgeshire, L = CAMBRIDGE, O = University of Cambridge, OU = UIS, CN = www.cst.cam.ac.uk
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:c3:cb:9f:77:9d:c8:09:f1:b3:aa:f7:8f:34:e5:
                    8e:92:45:50:a7:35:ee:09:27:ec:b8:7c:af:07:9d:
                    f4:30:79:37:2e:81:ad:85:ff:f9:c4:93:ca:27:28:
                    a0:d1:93:27:9f:0f:b4:c8:1c:55:7b:f9:90:46:c5:
                    69:56:40:b5:90:d6:34:06:8f:cc:b7:e0:31:3e:2f:
                    d9:2f:57:13:f5:4f:7d:ef:d8:7e:16:5a:d7:72:14:
                    e4:0b:13:87:b2:df:2a:0f:30:f1:6d:9a:b6:0b:c8:
                    e0:87:0f:b5:72:a0:b9:07:2e:48:32:bb:12:dd:d3:
                    96:21:1c:c9:94:8f:47:1d:9a:3b:35:ec:20:45:38:
                    06:15:ed:4e:43:80:96:94:90:fc:25:05:88:f6:7b:
                    cb:27:a9:49:e4:80:20:e7:f1:f0:23:05:e8:91:77:
                    c3:04:2e:2e:33:ca:76:fc:00:17:a4:93:88:f4:e1:
                    66:ac:51:56:8d:27:91:2d:d2:5a:e7:01:83:b9:ab:
                    c1:94:38:54:5a:f4:e9:dc:c5:91:96:b7:17:c6:55:
                    bf:ea:85:41:d5:0b:d7:09:62:26:01:c9:e1:fe:b3:
                    57:e0:b8:f9:fc:0b:76:65:6c:43:f0:5f:30:77:38:
                    17:00:df:6a:27:25:1d:30:9e:19:01:e9:0c:78:cc:
                    c9:3d
                Exponent: 65537 (0x10001)
```

```
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Authority Key Identifier:
                keyid:B3:12:89:B5:A9:4B:35:BC:15:00:F0:80:E9:D8:78:87:F1:13:7C:76

            Authority Information Access:
                CA Issuers - URI:http://trust.quovadisglobal.com/qvsslg3.crt
                OCSP - URI:http://ocsp.quovadisglobal.com

            X509v3 Subject Alternative Name:
                DNS:www.cst.cam.ac.uk
            X509v3 Certificate Policies:
                Policy: 1.3.6.1.4.1.8024.0.2.100.1.1
                  CPS: http://www.quovadisglobal.com/repository

            X509v3 Extended Key Usage:
                TLS Web Client Authentication, TLS Web Server Authentication
            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://crl.quovadisglobal.com/qvsslg3.crl

            X509v3 Subject Key Identifier:
                16:19:61:83:6F:AE:05:98:FA:62:9D:B8:AA:99:4B:C5:0A:04:8B:D9
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
    Signature Algorithm: sha256WithRSAEncryption
         8a:94:59:8a:70:00:c5:f9:1b:25:f0:04:c7:b8:79:46:6b:0c:
         2b:d1:01:1a:9e:83:4a:53:f9:c5:45:38:85:1d:f3:32:8a:f8:
         03:a2:bd:6d:f7:e6:5b:97:81:11:7d:c6:9c:d0:78:01:2a:f4:
         8a:d1:51:13:b0:a4:72:cc:40:55:8f:e8:bb:b1:ff:f9:66:0a:
         2d:fe:9c:69:12:2c:32:45:bc:0b:39:02:ff:68:12:94:ad:04:
         a2:4a:f0:5b:c6:cb:6d:d2:38:ad:dd:be:4c:1c:19:d2:69:b3:
         eb:98:e7:8e:bf:84:cb:a2:7c:07:47:de:99:d9:79:6e:e0:7a:
         01:89:a5:93:de:70:b5:69:56:1f:09:8d:15:04:88:e5:59:86:
         65:b3:fb:42:24:db:86:8c:d5:f7:f3:1c:cc:cd:7c:79:d4:32:
         4a:70:b7:f8:87:b3:0e:9b:93:ef:99:7f:a4:27:fb:7d:03:93:
         [... 16 lines deleted ...]
         5c:76:39:6d:51:dc:80:2e:cf:96:90:0c:b8:f1:ed:88:c8:c2:
         27:69:fe:0d:b9:ec:48:da:d4:f3:79:77:e1:3a:15:be:03:58:
         a6:d1:74:d7:4e:ec:d1:17
```

```
$ ls -l /etc/ssl/certs/
```

Outlook

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

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: $|A_1 \times A_2 \times \cdots \times A_n| = |A_1| \cdot |A_2| \cdot \cdots \cdot |A_n|$.
- A^n is the set of all n-tuples $(a_1, a_2, \ldots, a_n) = a_1 a_2 \ldots a_n$ with $a_1, a_2, \ldots, a_n \in A$. If A is finite then $|A^n| = |A|^n$.
- $A^{\leq n} = \bigcup_{i=0}^{n} A^i$ and $A^* = \bigcup_{i=0}^{\infty} A^i$
- Function $f : A \to B$ 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 $f : A_1 \times A_2 \times \cdots \times A_n \to B$ 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 $|\mathrm{Perm}(A)| = |A|! = 1 \cdot 2 \cdot \cdots \cdot |A|$ different permutations over a finite set A.
- B^A is the set of all functions of the form $f : A \to B$. If A and B are finite, there will be $|B^A| = |B|^{|A|}$ such functions.

Bit-sequence notations

We can further simplify some set and tuple notation to represent bit strings/sequences. For example, we can write the set of all $2^3 = 8$ sequences of three bits as

$$\begin{aligned}\{0,1\}^3 &= \{0,1\} \times \{0,1\} \times \{0,1\} \\ &= \{(0,0,0),(0,0,1),(0,1,0),(0,1,1), \\ &\quad (1,0,0),(1,0,1),(1,1,0),(1,1,1)\} \\ &= \{000,001,010,011,100,101,110,111\}\end{aligned}$$

Here we treat tuples of bits as bit sequences, merely dropping the separating commas and enclosing parentheses.

A superscript asterix ("Kleene star") denotes the set of all finite-length bit sequences, including the empty string $\varepsilon = \{0,1\}^0$:

$$\{0,1\}^* = \bigcup_{i=0}^{\infty} \{0,1\}^i$$

These notations are borrowed from formal languages theory.

Bit-sequence notations

It is also customary to use the exponentiation operator on bit sequences to denote repetition, e.g.

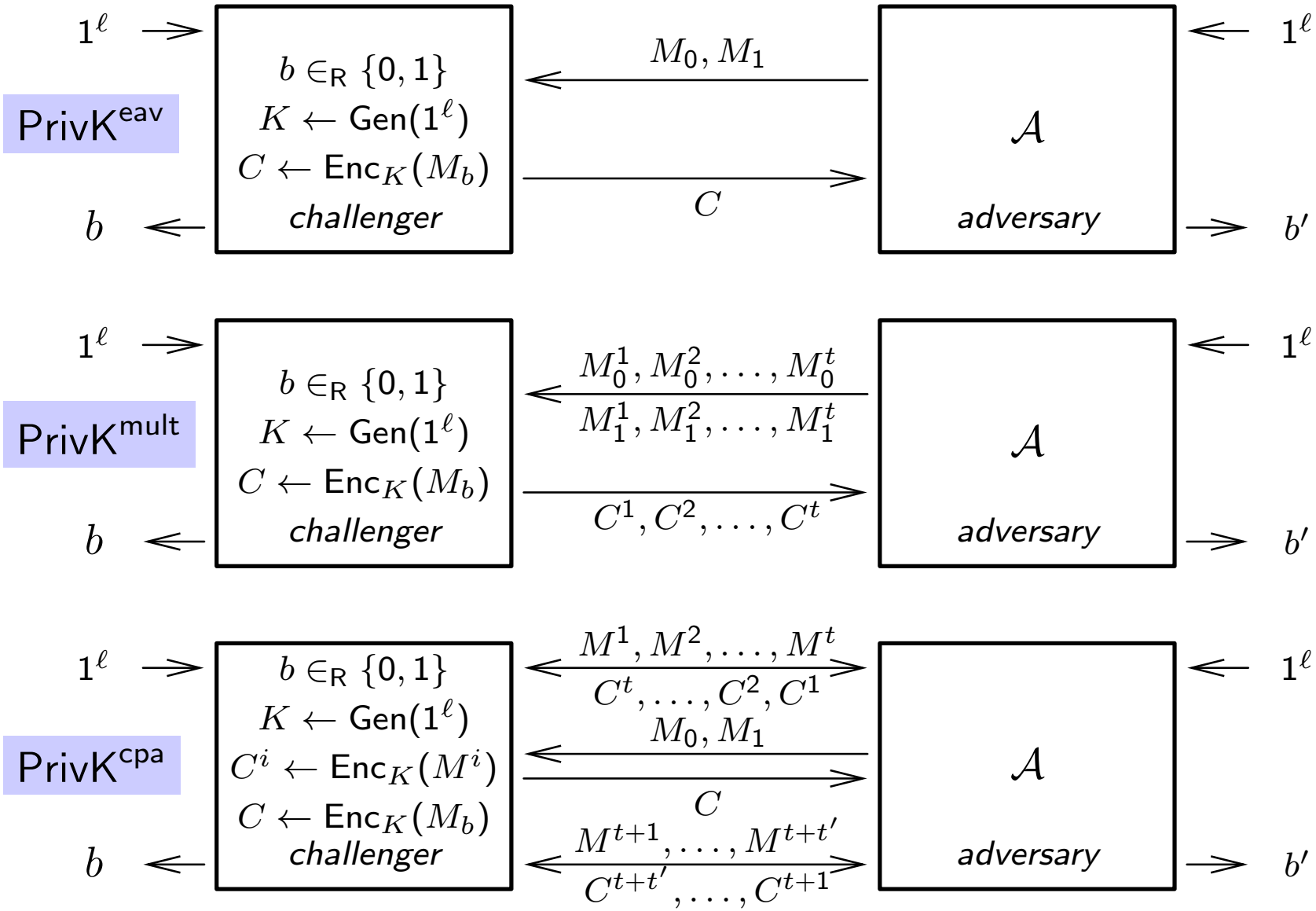
$$1^3 = 111, \qquad 01^3 = 0111, \qquad (10)^3 = 101010$$

Here we can think of operating on the *free monoid* $(\{0,1\}^*, \|)$ of bits with concatenation as the operator (slide 161), while also using multiplicative notation (slide 164), meaning that the monoid operator $\|$ can also be represented through juxtaposition (e.g., $1\|0 = 10$) and its repeated application can be written as exponentiation with a non-negative integer.

Occasionally, we need to represent a non-negative (or "unsigned", in C parlance) integer i in the range $0 \leq i < 2^n$ as an n-bit sequence. We use here the notation $\langle i \rangle_n$, or simply $\langle i \rangle$ if the number of bits n is clear from context. Using the "big-endian" or "most-significant-bit (MSB) first" convention, this means e.g. $\langle 11 \rangle_4 = 1011$.

In practice, the efficient and portable binary representation of integer numbers has to consider additional alignment constraints and memory-layout complications. Computer memory is commonly structured into 8-bit bytes, and different conventions ("big-endian" vs "little-endian") for the order of bytes representing an integer value in memory have been used by different manufacturers. As a result, cryptographic standards often have to spend significant text on unambiguously specifying the exact bit and byte layout of input and output values, to ensure interoperability between different implementations.

Confidentiality games at a glance



Integrity games at a glance