A primer of modern cryptography

KNUST University

Gabriel Chênevert

February 5, 2024

Grande
école
d'ingénieurs

Overview

whoami

Gabriel Chênevert

JUNIA, Lille (France)

Head of the Computer Science & Mathematics department

Interested in information theory:

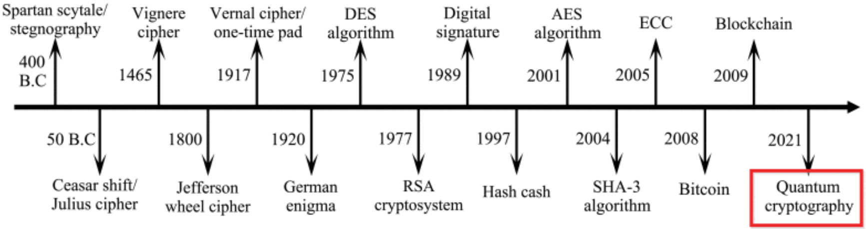
- signal processing
- error correction
- quantum computing
- **cryptography**

Class material available at https://junia.ovh/gch

What is cryptography

Cryptography provides a collection of primitives tand protocols that allow the manipulation of (digital) information securely even in the presence of *adversaries*.

A brief timeline of cryptography (source):



Goal of today: make sense of this

Détails du message

```
Received: from PR0P264MB2258.FRAP264.PROD.OUTLOOK.COM (2603:10a6:102:16a::5)
 by MR1P264MB3554.FRAP264.PROD.OUTLOOK.COM with HTTPS; Mon, 15 Jan 2024
 13:27:58 +0000
Received: from PA7P264CA0046.FRAP264.PROD.OUTLOOK.COM (2603:10a6:102:34a::11)
 by PR0P264MB2258.FRAP264.PROD.OUTLOOK.COM (2603:10a6:102:16a::5) with
 Microsoft SMTP Server (version=TLS1_2,
 cipher=TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384) id 15.20.7181.23; Mon, 15 Jan
 2024 13:27:54 +0000
Received: from PR2FRA01FT011.eop-fra01.prod.protection.outlook.com
 (2603:10a6:102:34a:cafe::67) by PA7P264CA0046.outlook.office365.com
 (2603:10a6:102:34a::11) with Microsoft SMTP Server (version=TLS1_2,
 cipher=TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384) id 15.20.7181.23 via Frontend
 Transport; Mon, 15 Jan 2024 13:27:54 +0000
Authentication-Results: spf=pass (sender IP is 77.238.177.145)
 smtp.mailfrom=yahoo.co.uk; dkim=pass (signature was verified)
 header.d=yahoo.co.uk;dmarc=pass action=none
 header.from=yahoo.co.uk;compauth=pass reason=100
Received-SPF: Pass (protection.outlook.com: domain of yahoo.co.uk designates
 77.238.177.145 as permitted sender) receiver=protection.outlook.com;
 client-ip=77.238.177.145; helo=sonic314-19.consmr.mail.ir2.yahoo.com; pr=C
Received: from sonic314-19.consmr.mail.ir2.yahoo.com (77.238.177.145) by
```

Shannon's communication model

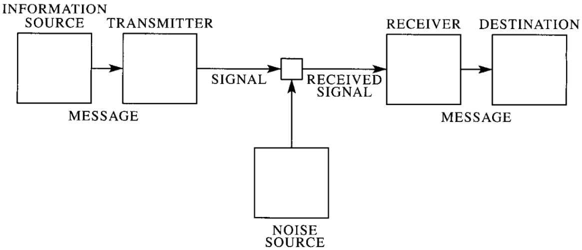


Fig. 1—Schematic diagram of a general communication system.

Claude Shannon, *A mathematical theory of communication* (1948)

Encoding

In order to be sent through the communication channel, messages need to be **encoded** in a suitable way (and decoded on the other side).

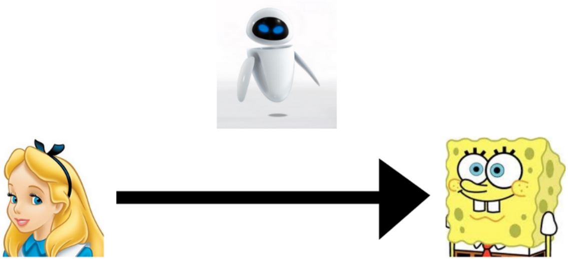
Encodings may achieve different desirable properties:

- existence
- compression
- integrity resistance
- confidentiality
- authentication
- non-repudiation

Overview

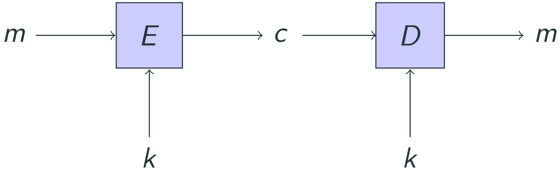
The secure channel problem

Alice wants to send a message to Bob, but doesn't want Eve to be able to read it



Secret-key cryptography

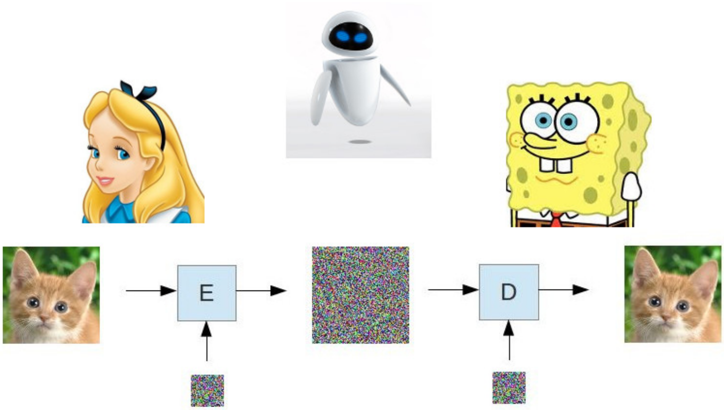
A **symmetric cipher** (or **cryptosystem**) consists of a pair of functions



where

- m: original message (**plaintext**)
- c: encrypted message (**ciphertext**)
- k: secret shared key

Illustration



Security level

Definition

The **security level** of a cryptosystem is (roughly) the $\log_2$ of the time complexity of the best known attack against it.

- Can change abruptly if new attack is discovered!
- No greater than key length (brute-force attack)
- Can be smaller. . .

Aside: orders of magnitude

- 2^5: number of persons in this room
- 2^{17}: number of students at KNUST
- 2^{21}: number of persons in Kumasi
- 2^{25}: number of persons in Ghana
- 2^{33}: total world population
- 2^{34}: number of views of the most popular video on YouTube

Computing resources

- 2^{70}: estimated number of operations / second performed by general-purpose computers
- 2^{73}: total digital memory available worldwide (in bits)

  cf. Hilbert & Lopez (2011)
- 2^{67}: number of SHA256 hashes computed per second by the Bitcoin network

Current consensus: 128-bit should be un-brute-forceable for the next 30 years

Security levels

Requirements of a symmetric cryptosystem

- **Correct decryption** : for all $k \in \mathcal{K}$ and $m \in \mathcal{M}$,

$$D(k, E(k, m)) = m.$$

- **Confidentiality** : knowledge of the ciphertext should not help an attacker guess or understand what the message is

  (can be formalized)

  i.e. there exists no efficient ciphertext-only attacks

Removeable masks

Definition (binary stream cipher)

Any cryptosystem with $\mathcal{M} = \mathcal{C} = \{0,1\}^n$, $\mathcal{K} = \{0,1\}^\ell$ and

$$\begin{cases} E(k,m) = m \oplus \mathrm{pad}(k) \\ D(k,c) \;=\; c \;\oplus \mathrm{pad}(k) \end{cases}$$

where $\mathrm{pad} : \{0,1\}^\ell \to \{0,1\}^n$ generates a **keystream** from the key k

in which $\oplus$ is the bitwise XOR operator

Example ($n = 8$)

Alice:

$m = 1110\,0011 = \texttt{e3}$

$\text{pad} = 0110\,1101 = \texttt{6d}$

$c = m \oplus \text{pad} = 1000\,1110 = \texttt{8e}$

Bob:

$c = 1000\,1110 = \texttt{8e}$

$\text{pad} = 0110\,1101 = \texttt{6d}$

$m = c \oplus \text{pad} = 1110\,0011 = \texttt{e3}$

Example ($n = 128$)

```python
from os import urandom

def xor(a,b):

    return bytes([x^y for x,y in zip(a,b)])

p = urandom(16)
```

```python
# Alice

m = b"Mask on 128 bits"

c = xor(m,p)

print(" m  = ", m.hex())
print("pad = ", p.hex())
print(" c  = ", c.hex())
```

```
 m  =  4d61736b206f6e203132382062697473
pad =  f31ed60c2c59c9ee4ba855cd71822d4a
 c  =  be7fa5670c36a7ce7a9a6ded13eb5939
```

```python
# Bob

mm = xor(c,p)

print(" c  = ", c.hex())
print("pad = ", p.hex())
print(" m  = ", mm.hex())
```

```
 c  =  be7fa5670c36a7ce7a9a6ded13eb5939
pad =  f31ed60c2c59c9ee4ba855cd71822d4a
 m  =  4d61736b206f6e203132382062697473
```

Security requirements (1/2)

Theorem

Stream ciphers decrypt correctly.

Proof.

$$\begin{aligned} D(k, E(k, m)) &= (m \oplus \mathrm{pad}(k)) \oplus \mathrm{pad}(k) \\ &= m \oplus (\mathrm{pad}(k) \oplus \mathrm{pad}(k)) \\ &= m \oplus 0 \\ &= m. \end{aligned}$$

$\square$

Security requirements (2/2)

Theorem (Shannon, 1949)

In the special case $\ell = n$, $pad(k) = k$, a stream cipher provides perfect secrecy (in the information-theoretic sense).

This special case called **one-time pad** or **Vernam cipher**.

But... the key need to be as long as the message!

In practice: stream ciphers

We want to keep $\mathcal{K} = \{0,1\}^\ell$ small and work with $\mathcal{M} = \mathcal{C} = \{0,1\}^*$ arbitrarily large:

use for keystream function

$$\text{pad} : \{0,1\}^\ell \longrightarrow \{0,1\}^*$$

a *cryptographically secure pseudo-random number generator* (CSPRNG)

with the secret key as seed

Pseudo-random number generators

In [1]:
```python
import random

# uses *insecure* but efficient Mersenne Twister PRNG

random.seed(12345)

for i in range(16):

    print(hex(random.randint(0,2**128))[2:-1])
```

```
6facaa5090e5e945452ec40a3193ca5
6ed4e94bdfc9e3b11fcff4545f811cb
bc428d42fa88269287f26aee175f0cd
25ece8452aa4857e8101e89a95c5fb9
d64a3ce030a1f6d513ed748bb80e3b0
56eaa3017576714a06057c82527122d
94820a06c555663f29ef41d0deea959
6a1eccdaa70ce1b51978cec0495cfa4
df8960ad1eab5cd83b788b660a4de3e
96af0dea41fad2962f927291ab721ab
213f191ff56ae7eaea80db0684ab561
f70ae8c026784184026530cdd50b612
282fe557578b24268a04f74f5987baf
9f3180427b1427081f1af1fac2e1dac
265015788e7ae9af1e8fcb74b2d4f32
f79fcaa0e47b342b2a3a46677eb14f8
```

Security requirement for CSPRNGs

To be used as a keystream generator, a PRNG needs to be *unpredictable* : an attacker cannot efficiently guess future outputs from previous ones

- *All* PRNGs are eventually periodic

  (deterministic stateful functions with a finite number of internal states)

  $\implies$ in particular, need to have long ($> 2^{128}$) period

- Beware: most "standard" PRNGs are easily predictable!

  $\implies$ **related-key attacks** on the underlying OTP

Example: Kaspersky's guessable passwords fiasco (2021)

Current recommendations

The eSTREAM project (ECRYPT 2008) proposes

- HC-128, Rabbit, Salsa/Chacha20, SOSEMANUK (software-oriented)
- Grain, MICKEY, Trivium (hardware-oriented)

(all force the PRNG to use a **nonce** as initial value)

Still need to be careful to seed the CSPRNG with enough entropy: using PID or timestamps is not a good idea!

$\Longrightarrow$ better to use dedicated entropy sources *e.g.* `/dev/urandom`, random.org, ...

The problem with stream ciphers

Mask reuse is a problem: if

$$\begin{cases} c_1 = m_1 \oplus \mathsf{pad} \\ c_2 = m_2 \oplus \mathsf{pad} \end{cases}$$

then

$$c_1 \oplus c_2 = m_1 \oplus m_2.$$

This means that:

1. Alice shouldn't use the same pad twice (ok using nonces)
2. it can be possible to manipulate the message through the encryption!

$$E(k, m \oplus \Delta) = E(k, m) \oplus \Delta \qquad (\textbf{malleability})$$

Different attackers

Eve (a passive attacker): sees the ciphertext, learns nothing ✓

Oscar (an active attacker): is able to modify the ciphertext, may have a different goal !

A different approach: block ciphers

Consider (E, D) a symmetric cipher with $\mathcal{M} = \mathcal{C}$.

For given $k \in \mathcal{K}$,

$$E_k := E(k, \cdot\,) : \mathcal{M} \longrightarrow \mathcal{M}$$

admits $D_k := D(k, \cdot\,)$ as inverse

hence E_k is a **permutation** of $\mathcal{M}$ (bijection from $\mathcal{M}$ to $\mathcal{M}$)

E_k as a permutation

e.g. with $|\mathcal{M}| = 2^8$:

$$\begin{array}{ccc}
\mathcal{M} & & \mathcal{M} \\
00000000 & & 11100110 \\
00000001 & & 01000101 \\
00000010 & \xrightarrow{\qquad E_k \qquad} & 01001011 \\
00000011 & & 11111110 \\
\vdots & & \vdots \\
11111101 & & 11101110 \\
11111110 & & 00010100 \\
11111111 & & 11011110
\end{array}$$

Idea

E_k should be thought of as a *pseudo-random permutation* of $\mathcal{M}$.

In practice: undistinguishable from a random *function* $\mathcal{M} \to \mathcal{M}$.

Allows one to:

- reuse keys (with some care!)
- work with small messages (**blocks**)

Note: typically $|\mathcal{K}| \ll |\text{permutations of } \mathcal{M}| = |\mathcal{M}|! \approx |\mathcal{M}|^{|\mathcal{M}|}$

ex.: $|\mathcal{K}| = |\mathcal{M}| = 2^{128}$, $|\mathcal{S}_{\mathcal{M}}| \approx |\mathcal{M}|^{|\mathcal{M}|} \approx 2^{43556142965880123323311949751266331066368}$ (!)

Do these things actually exist?

Shannon's paradigm: *confusion* and *diffusion* (stream ciphers miss the diffusion part)

Essentially all modern examples use an iterative design where the plaintext is encrypted a certain number of times by a **round function** performing (a small amount of) confusion and diffusion

$$\begin{cases} x_0 = m, \\ x_{i+1} = R(k_{i+1}, x_i), \quad 0 \leq i < r \\ E(k, m) = x_r \end{cases}$$

preceded by a **key scheduling** process $k \mapsto (k_1, \cdots, k_r)$.

Famous examples

n-bit block, ℓ-bit key, r rounds

- **Lucifer** (IBM, 1971) $n = \ell = 128$, $r = 16$
- **Data Encryption Standard** (NIST, 1977) $n = 64$, $\ell = 56$, $r = 16$

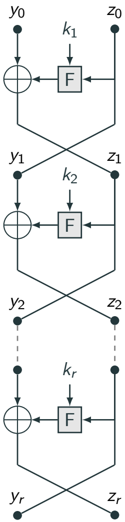
  Successful brute force attack in 1997

  Still survived in the form of Triple DES in legacy hardware/software
- **Rijndael** (KU Leuven, 1998) *aka* **Advanced Encryption Standard** (NIST, 2000)

  $n = 128$, $\ell \in \{128, 192, 256\}$, $r \in \{10, 12, 14\}$.

But also: RC5/RC6, IDEA, Serpent, Blowfish/Twofish, ...

Design of DES



16-round **Feistel network** :

Write each $x_i = y_i \,\|\, z_i$ left and right parts

Round function:

$$\begin{cases} y_{i+1} = z_i \\ z_{i+1} = y_i \oplus F(k_{i+1}, z_i) \end{cases}$$

Easy to implement in hardware
(and invert – exercice!)

Security proof

Theorem (Luby-Rackhoff, 1988)

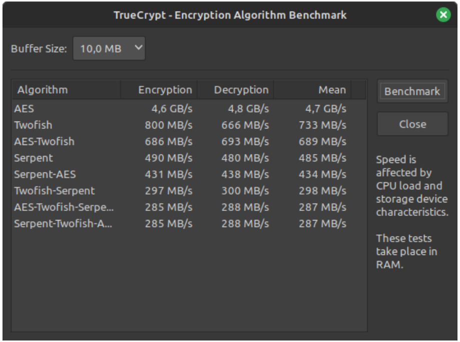
Three rounds of a Feistel network with inner function F a CSPRNG using k as a seed is computationally undistinguishable from a random permutation.

In practice: increase the number of rounds to take into account the fact that F might not be a provably good CSPRNG.

Still: the original DES can now be broken by exhaustive key search in a couple of hours with COPACOBANA

Today

In practice: **use AES** or some other NIST finalist



Algorithm	Encryption	Decryption	Mean
AES	4,6 GB/s	4,8 GB/s	4,7 GB/s
Twofish	800 MB/s	666 MB/s	733 MB/s
AES-Twofish	686 MB/s	693 MB/s	689 MB/s
Serpent	490 MB/s	480 MB/s	485 MB/s
Serpent-AES	431 MB/s	438 MB/s	434 MB/s
Twofish-Serpent	297 MB/s	300 MB/s	298 MB/s
AES-Twofish-Serpe...	285 MB/s	288 MB/s	287 MB/s
Serpent-Twofish-A...	285 MB/s	288 MB/s	287 MB/s

Modes of operation

Now suppose the message to be encrypted is longer than a single block:

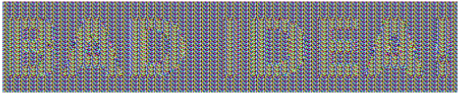
$$m = m_1 \parallel m_2 \parallel m_3 \parallel \ \cdots$$

How to use a block cipher (*e.g.* AES) to encrypt m ?

- **Electronic Code Book (ECB) mode**:

$$\begin{cases} c_i = E(k, m_i) \\ m_i = D(k, c_i) \end{cases}$$

ECB mode?



Problem: equal blocks yield equal ciphertexts

Should use (pseudo-)**probabilistic encryption**:

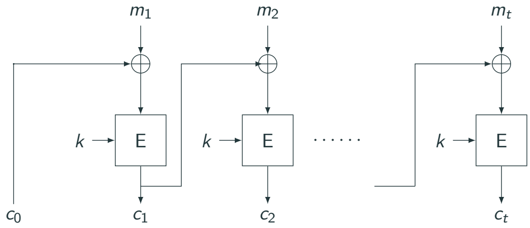
a given block shouldn't always have the same encryption

$\rightsquigarrow$ use of either *random value* or *nonce* (counter)

Side effect: need to have $|\mathcal{C}| > |\mathcal{M}|$ to make room for redundancy in the ciphertexts

Cipher Block Chaining (CBC) mode

$$\begin{cases} c_0 = \text{random } \mathbf{I}\text{nitial } \mathbf{V}\text{alue} \\ c_i = E(k, m_i \oplus c_{i-1}) \end{cases}$$



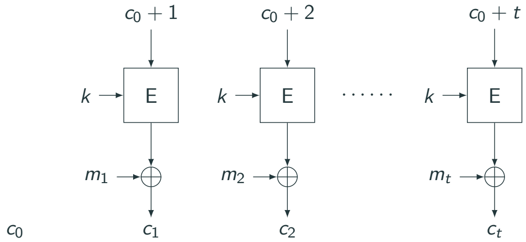
- Encryption is sequential (but decryption can be parallelized)

$$m_i = D(k, c_i) \oplus c_{i-1}$$

- Message has to be padded to a multiple of the block length
- Crucial that random IV is non-predictable (chosen plaintext attack)

Randomized counter (CTR) mode

$$\begin{cases} c_0 = \text{random IV} \\ c_i = m_i \oplus E(k, c_0 + i) \end{cases}$$



- Block cipher is effectively turned into a stream cipher
- No padding problem
- Highly parallelizable
- Random IV prevents reuse of key stream

Other modes

Many other modes that achieve specific goals exist.

- feedback modes: CFB, OFB, ...
- device encryption: LRW, XEX, XTS, ...
- authenticated encryption: OCB, EAX, GCM, ...

In most modern communication systems, we tend to use (if possible)

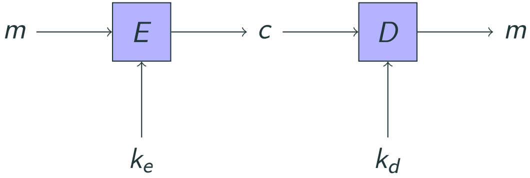
AEAD = **A**uthenticated **E**ncryption with **A**ssociated **D**ata

in order to guarantee confidentiality + integrity of messages and prevent replay attacks

Overview

Asymmetric encryption

In general, a cipher might use different keys for encryption and decryption:



(includes symmetric ciphers as the special case $k_e = k_d$)

if knowledge about one gives no useful information about the other

then one of them can be made public

Public-key encryption

The encryption key k_e is made public (k_d kept private)

anyone can write to Bob, but only he can read

As implemented by *e.g.* PGP/GPG

NB : Public-key encryption is very rarely used, if at all, in modern systems

Modular arithmetic

Recall (?)

Definition

We say that $a \underset{n}{\equiv} b$ when n divides $b - a$, *i.e.* $b = a + kn$ for some integer k

i.e. a and b are equal, up to ("modulo") a multiple of n

Remarks:

- $a \underset{n}{\equiv} b$ if and only if $a \,\%\, n = b \,\%\, n$
- If $a \underset{n}{\equiv} b$ and $c \underset{n}{\equiv} d$, then $(a + c) \underset{n}{\equiv} (b + d)$ and $(ac) \underset{n}{\equiv} (bd)$

Rivest-Shamir-Adleman (1977)

Fix some integer, product of two distinct (large) prime numbers $n = p \cdot q$

$\mathcal{M} = \mathcal{C} = \mathbb{Z}/n\mathbb{Z}$, identified with $[\![0, n[\![$

$$\begin{cases} E(e, m) :\underset{n}{\equiv} m^e \\ D(d, c) :\underset{n}{\equiv} c^d \end{cases}$$

with $d \cdot e \underset{\varphi(n)}{\equiv} 1$ where $\varphi(n) = (p-1)(q-1)$.

A small (thus very insecure) working example

```
n   = 74989
phi = 69600
e   = 52027
d   = 10963

d*e mod phi = 1

message:    60211
encryption: 13247
decryption: 60211
```

Try here

Remarks on RSA

- Modular exponentiation can be efficiently computed (`pow(m,e,n)` in Python)
- The public exponent can be chosen be small (often $e = 65537$)
- Knowing $\varphi(n)$, it is easy to deduce d from e (Euclidean algorithm)
- The security of RSA relies on the computational hardness of factoring n

  without knowing p and q, the attacker doesn't know the value of $\varphi(n)$
- Knowledge of the full pair (d, e) is equivalent to knowing the factors (p, q)

  RSA moduli should never be reused

Attacks on RSA (aka factorization algorithms)

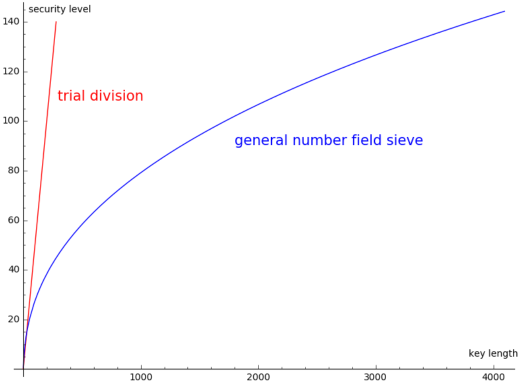
There is a very large litterature devoted to the subject of integer factorization.

As of 2024, the best general purpose algorithm is the *General Number Field Sieve* (GNFS) that factors an ℓ-bit integer in

$$\approx 5.5^{\,\ell^{1/3}(\ln \ell)^{2/3}} \text{ time.}$$

Public factorization record: RSA-250 (2020)

Consequence on key length



According to RSA Security, Inc.

Symmetric key size	Equivalent RSA key size
80	1024
112	2048
128	3072
256	15360

Real-world RSA

The *plain RSA* described above has all sorts of problems:

- malleability: $E(e, m_1) \cdot E(e, m_2) = E(e, m_1 \cdot m_2)$
- lack of randomness
- fixed size of plaintext
- ...

In practice, a suitable padding scheme needs to be used.

$\implies$ use a library!

Overview

Secret sharing

Public-key encryption provides a partial solution to the problem of setting up a shared private key for symmetric encryption on an insecure channel:

- Alice chooses secret k,
- encrypts it with Bob's public encryption key,
- and sends it to him;
- Bob recovers k using his private decryption key.

Are there problems with that? (hint: yes, some)

"Symmetric" version

- Alice chooses k_A and sends it to Bob using his public encryption key;
- Bob chooses k_B and sends it to Alice using her public encryption key;
- Shared secret is $k := k_A \oplus k_B$.

Better: neither Alice nor Bob fully controls the final secret.

But two public encryption key pairs are needed...

Diffie-Hellman (1976)

- Alice and Bob agree on "safe" parameters n and g.
- Alice chooses α, computes $a \underset{n}{\equiv} g^{\alpha}$ and sends it to Bob.
- Bob chooses β, computes $b \underset{n}{\equiv} g^{\beta}$ and sends it to Alice.

Shared secret is

$$k :\underset{n}{\equiv} g^{\alpha\beta} \equiv a^{\beta} \equiv b^{\alpha}.$$

Diffie-Hellman problem

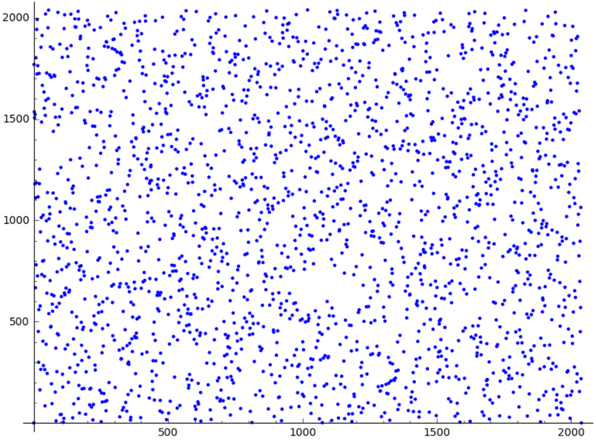
Eve is faced with the problem:

given a and b, recover k.

We *believe* that her best line of attack is:

- solve $a \underset{n}{\equiv} g^{\alpha}$ for α (or $b \underset{n}{\equiv} g^{\beta}$ for β) **discrete logarithm problem**
- then easily deduce $k \underset{n}{\equiv} g^{\alpha\beta}$ as Alice (or Bob) would.

Example: $a \underset{2039}{\equiv} 1769^{\alpha}$



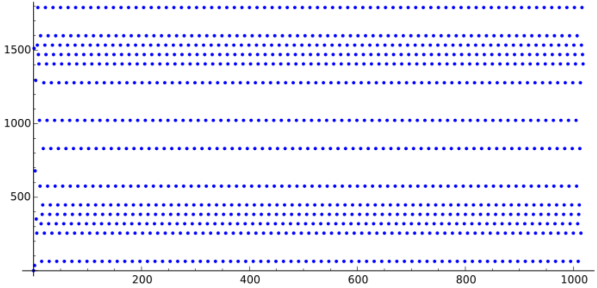
DH caveats

- Should **always** be used in conjunction with authentication to prevent *man-in-the-middle attacks*

DH caveats

- Bob should check that Alice does not provide a value of a for which the discrete log is easy (same on Alice's side)

Example: $a \underset{1856}{\equiv} 1514^{\alpha}$



Attacks on DH (aka DLP algorithms)

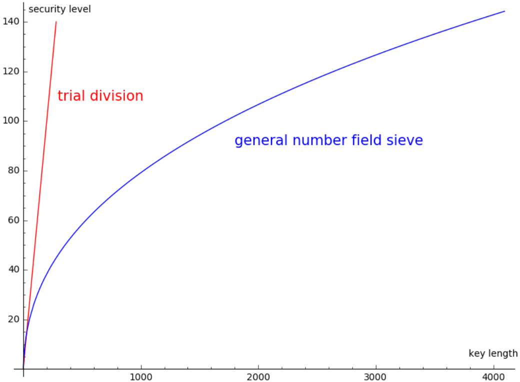
There also exists a general-purpose probabilistic algorithm for the DLP that takes (on average) $\mathcal{O}(\sqrt{\nu})$ steps (and $\mathcal{O}(1)$ memory)

Also: the General Number Field Sieve solves the *modular* DLP

$\implies$ use same key lengths as for RSA

Current world records

Recall



security level
140
120
100
80
60
40
20
trial division
general number field sieve
key length
1000
2000
3000
4000

Generalized DLP

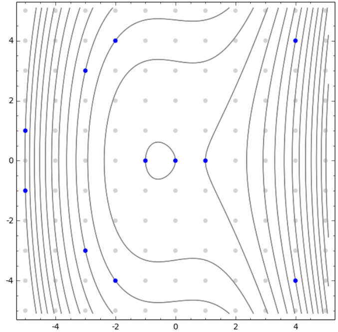
The nice thing about the DLP is that it can be asked for any binary operation $\star$:

Given g and a such that

$$a = g^\alpha = \underbrace{g \star g \star \cdots \star g}_{\alpha}, \qquad \text{find } \alpha \in \mathbb{N}$$

So far we used $\star =$ modular multiplication, but there are other interesting operations...

Elliptic curves



Best known DLP algorithms are the *generic* ones

$\implies$ ℓ-bit security achieved by 2ℓ-bit keys ☺

Overview

Recall: Generalized DLP

Let $(\mathcal{G}, \cdot\,)$ be a finite abelian group.

Given $g \in \mathcal{G}$ and x such that

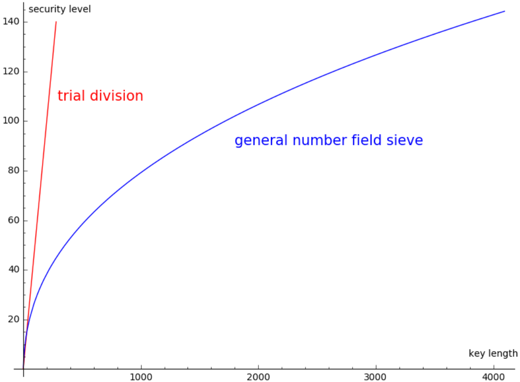
$$x = g^{\ell} = \underbrace{g \cdot g \cdots g}_{\ell} \quad \text{in } \ \mathcal{G},$$

find $\ell \underset{m}{\equiv} \mathrm{dlog}_{\mathcal{G}}(x, g)$, with $m = \mathrm{ord}_{\mathcal{G}}(g)$, the smallest $m > 0$ for which $g^m = 1$.

Best known DL algorithm: $\mathcal{O}(m^{\frac{1}{2}})$ for a generic group $\mathcal{G}$.

(Much smaller for $\mathcal{G} = (\mathbb{Z}/n\mathbb{Z})^{\times}$.)

Recall



security level
140
120
100
80
60
40
20
trial division
general number field sieve
key length
1000
2000
3000
4000

Elliptic curves

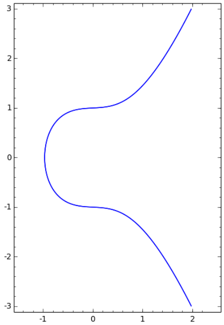
Definition

An *elliptic curve* is a plane curve defined by an equation of the form

$$\mathcal{E}: \; y^2 = x^3 + ax + b.$$

Example

$a = \frac{1}{10}, \; b = 1$



Some famous elliptic curves

Secp256k1 (Bitcoin, Ethereum)

$$y^2 = x^3 + 7$$

Curve25519 (Monero, Zcash, ...)

$$y^2 = x^3 + 486662x^2 + x$$

Addition on an elliptic curve

Given $P, Q \in \mathcal{E}$, the line through P and Q intersects $\mathcal{E}$ at a third point, say $R = (x, y)$.

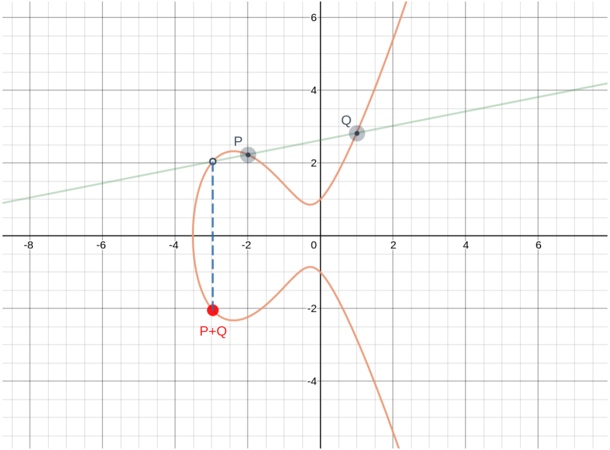
Definition

$$P + Q := (x, -y)$$

Fun fact: This makes $\mathcal{E} \cup \{O\}$ into an abelian group!

(The *point at infinity* $O = (0, \infty)$ being the neutral element)

Addition on an elliptic curve



DLP on an elliptic curve

Given $G \in \mathcal{E}$ of (additive) order m and $P \in \mathcal{E}$ such that

$$P = \ell G = \underbrace{G + \cdots + G}_{\ell} \quad \text{in} \quad \mathcal{E},$$

find $\ell \underset{m}{\equiv} \mathrm{dlog}_{\mathcal{E}}(P, G)$.

(Easy to solve over the real or complex numbers)

Elliptic curves over finite fields

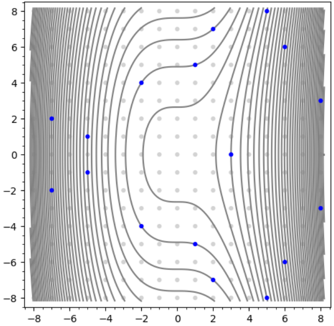
Instead: consider solutions modulo a fixed prime p

$$y^2 \underset{p}{\equiv} x^3 + ax + b$$

$\rightsquigarrow \mathcal{E}(\mathbb{F}_p)$ elliptic curve over the field with p elements

(a finite abelian group!)

$$y^2 \underset{17}{\equiv} x^3 - x$$



Basic computations are easy...

```
p = 32806352226718822643429
```

Evaluate

Language: Sage

Share

```
(9956939019642126506349 : 26680698275736540367982 : 1)
```

Help | Powered by SageMath

...but the DLP is hard!

Size of $\mathcal{E}$

Theorem (Hasse bound)

$$\#\mathcal{E}(\mathbb{F}_p) = 1 + p + \mathcal{O}(\sqrt{p})$$

hence $\#\mathcal{E}(\mathbb{F}_p) \approx p$.

We use elliptic curves with points G of large order $m \approx p$.

ECDH

- Alice and Bob agree on "safe" parameters $\mathcal{E}$ and G.
- Alice chooses a, computes $A = aG$ in $\mathcal{E}$.
- Bob choooses b, computes $B = bG$ in $\mathcal{E}$.
- Shared secret is

$$K := (ab)G = aB = bA.$$

Parameter generation

To get ℓ bits of security:

- choose a 2ℓ-bit prime p
- an elliptic curve $\mathcal{E}$ over $\mathbb{F}_p$
- and a point G on $\mathcal{E}$ of (almost) prime order m that generates (most of) $\mathcal{E}(\mathbb{F}_p)$.

Much harder to manufacture than *e.g.* for RSA – but can be reused.

Overview

To achieve message authentication

- An authenticated encryption mode can be used (e.g. GCM)
- Or a dedicated primitive like HMAC (that **doesn't** provide confidentiality)

but these do not provide *sender authentication* (why ?)

$\Longrightarrow$ message forgery is possible (by Bob)

$\Longrightarrow$ message repudiation is possible (by Alice)

Digital signatures

A **digital signature scheme** consists of a pair of algorithms:

- **signature** $S(k_{\text{priv}}, m)$
- **verification** $V(k_{\text{pub}}, m, s) \in \{0, 1\}$

(along with a **key generation** algorithm that produces pairs $(k_{\text{priv}}, k_{\text{pub}})$)

Requirements of a signature algorithm

- Correct verification:

$$\mathrm{Verify}(k_{\mathrm{pub}}, m, \mathrm{Sign}(k_{\mathrm{priv}}, m)) = \mathtt{true} \qquad \text{for all} \quad m$$

- Non-forgery

  impossible in practice to manufacture a valid signature for a (new) message m without access to k_{priv}

  In particular: not possible to recover k_{priv} from k_{pub}.

- Consequence: non-repudiation

  If Alice keeps k_{priv} private and a valid signature for k_{pub} is encountered, it means she did sign (a signature is binding)

Desirable properties of a signature algorithm

- Efficiency:

  the Sign and Verify algorithms should be reasonably fast

- Signature conciseness:

  the produced signatures s should be reasonably small

- Key conciseness:

  the private and public keys k_{priv} and k_{pub} should not be too large

- Efficient key generation:

  should be easy to come up with new pairs $(k_{\text{priv}}, k_{\text{pub}})$

  e.g. reusable parameters

Hasn-then-sign paradigm

To sign a message m with private key k_e:

- Alice computes $h = H(m)$;
- appends $s = E(k_e, h)$ to m.

Upon reception of a pair (m, s):

- Bob checks with associated public key whether

$$D(k_d, s) \stackrel{?}{=} H(m).$$

Timeline of signature algorithms

- 80's - 90's: RSA-PSS, DSA (integer-based)
- 00's - 20's: ECDSA, EdDSA, Ed25519 (**elliptic curve**-based)
- 30's - ??'s: quantum-resistant signatures (lattice or hash-based)

As of 2024, there is an ongoing standardization process led by NIST to specify:

- ML-DSA (previously known as CRYSTALS-Dilithium)
- SLH-DSA (previously known as SPHINCS+)

as well and alternative lattice-based signature scheme (previously known as FALCON)

and a key establishment primitive ML-KEM (previously known as CRYSTALS-Kyber)

References

- These slides and Jupyter notebook: https://junia.ovh/gch
- JP Aumasson, *Serious Cryptography*, Starch Press (2017)
- D. Boneh, *Cryptography 1*, Coursera