Cryptography

Coding Technology

Illés Horváth

2025/11/26

Coding Technology

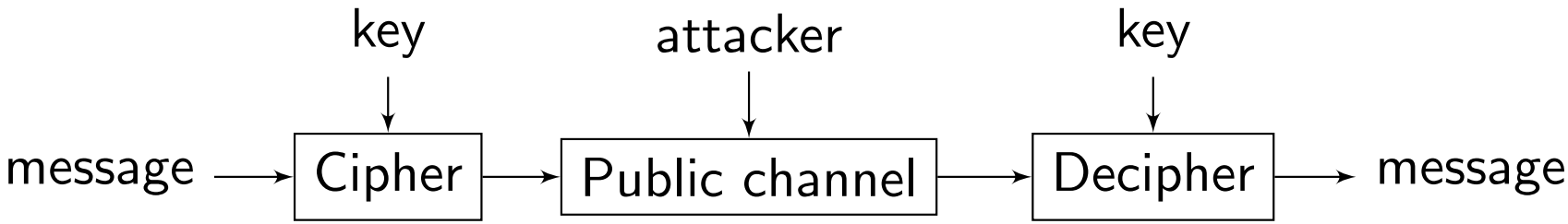
General setup.

During the Coding Technology course, our main focus is on various types of codes:

- Error correction codes adding redundancy to the messages that allow detection and/or correction of errors, allowing reliable communication over noisy channels. Also known as Error Control, or Channel Coding.
- Data compression codes identify patterns and eliminate redundancies in data. Also known as Source Coding.
- **Cryptography protocols** that turn readable information into unintelligible text, allowing secure private communication over public channels.

Cryptography

Main objective: secure communication over a public channel.

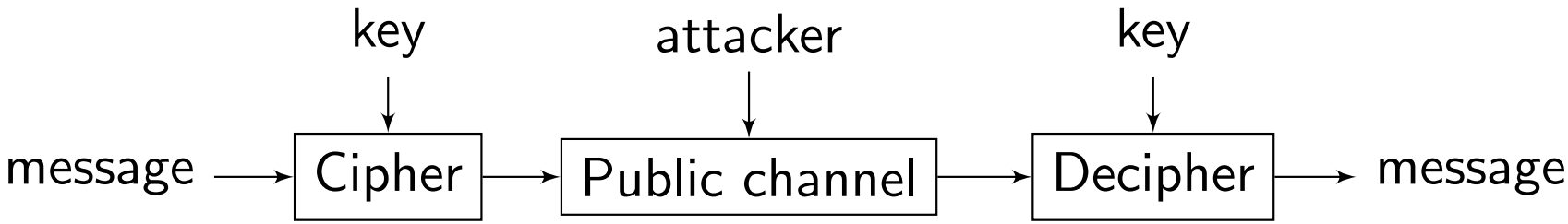


Construct algorithms that turn private information into unintelligible (obscure/nonsense) text (encryption) that can only be read by reversing the process (decryption).

Ideally, decryption should be practically impossible without the key, but low computational complexity for the receiver who has the key.

Cryptography

Main objective: secure communication over a public channel.



Construct algorithms that turn private information into unintelligible (obscure/nonsense) text (encryption) that can only be read by reversing the process (decryption).

Ideally, decryption should be practically impossible without the key, but low computational complexity for the receiver who has the key.

(There are other applications, e.g. authentication, which are similar in spirit and use mostly similar techniques, but the setup is slightly different.)

Symmetric encryption

Symmetric encryption or *conventional* setup for secure communication:

- the key k is from a finite set of size K;
- $x = (x_1 \dots x_M)$: the original message or *plaintext*;
- $y = (y_1 \dots y_N)$: the encrypted message or *ciphertext*.

Symmetric encryption

Symmetric encryption or *conventional* setup for secure communication:

- the key k is from a finite set of size K;
- $x = (x_1 \dots x_M)$: the original message or *plaintext*;
- $y = (y_1 \dots y_N)$: the encrypted message or *ciphertext*.

For each k,

$$y = E_k(x)$$

where E_k is the encryption function, and

$$x = D_k(y)$$

where D_k is the decryption function, which is the inverse of E_k.

Setup

In secure communication between two parties, the sender is often referred to as Alice, while the receiver is Bob.

Setup

In secure communication between two parties, the sender is often referred to as Alice, while the receiver is Bob.

There is also an enemy/adversary Eve, who is listening (eavesdropping) on the public communications channel used by Alice and Bob, and wants to decrypt the messages sent to Bob.

Setup

In secure communication between two parties, the sender is often referred to as Alice, while the receiver is Bob.

There is also an enemy/adversary Eve, who is listening (eavesdropping) on the public communications channel used by Alice and Bob, and wants to decrypt the messages sent to Bob.

Eve's goal is typically to identify either the plaintext x or the key k. In order to do this, she wants to conduct an *attack*.

Setup

In secure communication between two parties, the sender is often referred to as Alice, while the receiver is Bob.

There is also an enemy/adversary Eve, who is listening (eavesdropping) on the public communications channel used by Alice and Bob, and wants to decrypt the messages sent to Bob.

Eve's goal is typically to identify either the plaintext x or the key k. In order to do this, she wants to conduct an *attack*.

We generally assume that the cryptography protocol used is fully known to all parties, except for the key, which is unknown to Eve.

Types of attacks

Passive attacks:

- ciphertext only attack: Eve has a series of $E_k(x_1), \ldots, E_k(x_L)$ ciphertexts encrypted with a common key k;

Types of attacks

Passive attacks:

- ciphertext only attack: Eve has a series of $E_k(x_1), \ldots, E_k(x_L)$ ciphertexts encrypted with a common key k;
- known plaintext attack: Eve has a series of $(x_1, E_k(x_1)), \ldots, (x_L, E_k(x_L))$ pairs encrypted with a common key k;

Types of attacks

Passive attacks:

- ciphertext only attack: Eve has a series of $E_k(x_1), \ldots, E_k(x_L)$ ciphertexts encrypted with a common key k;
- known plaintext attack: Eve has a series of $(x_1, E_k(x_1)), \ldots, (x_L, E_k(x_L))$ pairs encrypted with a common key k;
- chosen plaintext attack: for any x, Eve can obtain $E_k(x)$;

Types of attacks

Passive attacks:

- ciphertext only attack: Eve has a series of $E_k(x_1), \ldots, E_k(x_L)$ ciphertexts encrypted with a common key k;
- known plaintext attack: Eve has a series of $(x_1, E_k(x_1)), \ldots, (x_L, E_k(x_L))$ pairs encrypted with a common key k;
- chosen plaintext attack: for any x, Eve can obtain $E_k(x)$;
- chosen text attack: for any x, Eve can obtain $E_k(x)$, and for any y, Eve can obtain $D_k(y)$.

Types of attacks

Passive attacks:

- ciphertext only attack: Eve has a series of $E_k(x_1), \ldots, E_k(x_L)$ ciphertexts encrypted with a common key k;
- known plaintext attack: Eve has a series of $(x_1, E_k(x_1)), \ldots, (x_L, E_k(x_L))$ pairs encrypted with a common key k;
- chosen plaintext attack: for any x, Eve can obtain $E_k(x)$;
- chosen text attack: for any x, Eve can obtain $E_k(x)$, and for any y, Eve can obtain $D_k(y)$.

Passive attacks usually exploit statistical patterns in either the plaintext or the ciphertext.

Types of attacks

Active attacks:

- ciphertext modification: replacing a ciphertext by directly accessing the channel;

Types of attacks

Active attacks:

- ciphertext modification: replacing a ciphertext by directly accessing the channel;
- impersonation/spoofing attack: impersonating either Alice or Bob to obtain information;

Types of attacks

Active attacks:

- ciphertext modification: replacing a ciphertext by directly accessing the channel;
- impersonation/spoofing attack: impersonating either Alice or Bob to obtain information;
- man-in-the-middle attack: getting between the communication of Alice and Bob and changing it in both directions;

Types of attacks

Active attacks:

- ciphertext modification: replacing a ciphertext by directly accessing the channel;
- impersonation/spoofing attack: impersonating either Alice or Bob to obtain information;
- man-in-the-middle attack: getting between the communication of Alice and Bob and changing it in both directions;
- possibly other types of manipulation.

Additive cipher

Historical examples.

Additive cipher, also known as Caesar cipher. If the size of the alphabet is n (e.g. $n = 26$ for English texts),

$$E_k(x) = y = x + k \mod n,$$

where k is the value of the key.

Additive cipher

Historical examples.

Additive cipher, also known as Caesar cipher. If the size of the alphabet is n (e.g. $n = 26$ for English texts),

$$E_k(x) = y = x + k \mod n,$$

where k is the value of the key.

Example. If $x = \text{ABRACA}$, and $k = 2$, then

$$y = E_k(x) = \text{CDTCEC}$$

Additive cipher

Historical examples.

Additive cipher, also known as Caesar cipher. If the size of the alphabet is n (e.g. $n = 26$ for English texts),

$$E_k(x) = y = x + k \mod n,$$

where k is the value of the key.

Example. If $x = \text{ABRACA}$, and $k = 2$, then

$$y = E_k(x) = \text{CDTCEC}$$

Decoding is

$$D_k(y) = x = y - k \mod n$$

Additive cipher

Issue: the additive cipher is vulnerable to attacks.

The ciphertext FDHVDU has been encrypted with an additive cipher. Attack it.

Additive cipher

Issue: the additive cipher is vulnerable to attacks.

The ciphertext FDHVDU has been encrypted with an additive cipher. Attack it.

We could try out all 26 possible keys. Any ideas about what order?

Additive cipher

Issue: the additive cipher is vulnerable to attacks.

The ciphertext FDHVDU has been encrypted with an additive cipher. Attack it.

We could try out all 26 possible keys. Any ideas about what order?

The ciphertext contains 2 D's. Using that the most common letters in English texts are E, T, A, O, I, N, we could try to match D to those letters first.

Additive cipher

Issue: the additive cipher is vulnerable to attacks.

The ciphertext FDHVDU has been encrypted with an additive cipher. Attack it.

We could try out all 26 possible keys. Any ideas about what order?

The ciphertext contains 2 D's. Using that the most common letters in English texts are E, T, A, O, I, N, we could try to match D to those letters first.

$$E \to D \quad \Longleftrightarrow \quad k = 25 \quad \to \quad x = \text{GEIWEV}$$

Additive cipher

Issue: the additive cipher is vulnerable to attacks.

The ciphertext FDHVDU has been encrypted with an additive cipher. Attack it.

We could try out all 26 possible keys. Any ideas about what order?

The ciphertext contains 2 D's. Using that the most common letters in English texts are E, T, A, O, I, N, we could try to match D to those letters first.

$$\begin{array}{lcl} E \to D & \iff & k = 25 \to x = \text{GEIWEV} \\ T \to D & \iff & k = 10 \to x = \text{VTXLTK} \end{array}$$

Additive cipher

Issue: the additive cipher is vulnerable to attacks.

The ciphertext FDHVDU has been encrypted with an additive cipher. Attack it.

We could try out all 26 possible keys. Any ideas about what order?

The ciphertext contains 2 D's. Using that the most common letters in English texts are E, T, A, O, I, N, we could try to match D to those letters first.

$$
\begin{array}{lcll}
E \to D & \iff & k = 25 \to & x = \text{GEIWEV} \\
T \to D & \iff & k = 10 \to & x = \text{VTXLTK} \\
A \to D & \iff & k = 3 \to &
\end{array}
$$

Additive cipher

Issue: the additive cipher is vulnerable to attacks.

The ciphertext FDHVDU has been encrypted with an additive cipher. Attack it.

We could try out all 26 possible keys. Any ideas about what order?

The ciphertext contains 2 D's. Using that the most common letters in English texts are E, T, A, O, I, N, we could try to match D to those letters first.

$$
\begin{array}{lcll}
E \to D & \iff & k = 25 & \to \quad x = \text{GEIWEV} \\
T \to D & \iff & k = 10 & \to \quad x = \text{VTXLTK} \\
A \to D & \iff & k = 3 & \to \quad x = \text{CAESAR}
\end{array}
$$

Linear cipher

Linear cipher:

$$E_k(x) = y = ax + b \mod n,$$

where $k = (a, b)$ is the value of the key. $\gcd(a, n) = 1$ must hold!

Linear cipher

Linear cipher:

$$E_k(x) = y = ax + b \mod n,$$

where $k = (a, b)$ is the value of the key. $\gcd(a, n) = 1$ must hold!

Decryption is also linear:

$$D_k(y) = a^{-1}y - a^{-1}b \mod n.$$

Linear cipher

Linear cipher:

$$E_k(x) = y = ax + b \mod n,$$

where $k = (a, b)$ is the value of the key. $\gcd(a, n) = 1$ must hold!

Decryption is also linear:

$$D_k(y) = a^{-1}y - a^{-1}b \mod n.$$

Just slightly better than additive ciphers, but still vulnerable to attacks.

Linear block cipher

Linear block cipher. $k = (A, b)$, where A is an invertible $N \times N$ binary matrix and b is a binary column vector of length N.

x and y are binary column vectors of length N.

Linear block cipher

Linear block cipher. $k = (A, b)$, where A is an invertible $N \times N$ binary matrix and b is a binary column vector of length N.

x and y are binary column vectors of length N.

Encryption is

$$E_k(x) = y = Ax + b,$$

decryption is

$$D_k(y) = x = A^{-1}(y - b)$$

Linear block cipher

Linear block cipher. $k = (A, b)$, where A is an invertible $N \times N$ binary matrix and b is a binary column vector of length N.

x and y are binary column vectors of length N.

Encryption is

$$E_k(x) = y = Ax + b,$$

decryption is

$$D_k(y) = x = A^{-1}(y - b)$$

Attack it using a known plaintext attack.

Linear block cipher

A known plaintext attack means that pairs $(x_1, y_1), (x_2, y_2), \ldots$ are available to Eve.

Linear block cipher

A known plaintext attack means that pairs $(x_1, y_1), (x_2, y_2), \ldots$ are available to Eve.

We can eliminate b:

$$\begin{aligned} y_2 - y_1 &= A(x_2 - x_1) \\ y_3 - y_1 &= A(x_3 - x_1) \\ &\vdots \end{aligned}$$

Linear block cipher

A known plaintext attack means that pairs $(x_1, y_1), (x_2, y_2), \ldots$ are available to Eve.

We can eliminate b:

$$\begin{aligned} y_2 - y_1 &= A(x_2 - x_1) \\ y_3 - y_1 &= A(x_3 - x_1) \\ &\vdots \end{aligned}$$

Can we compute A?

Linear block cipher

A known plaintext attack means that pairs $(x_1, y_1), (x_2, y_2), \ldots$ are available to Eve.

We can eliminate b:

$$\begin{aligned} y_2 - y_1 &= A(x_2 - x_1) \\ y_3 - y_1 &= A(x_3 - x_1) \\ &\vdots \end{aligned}$$

Can we compute A?

Not from a single equation, but if we put together several, then we get

$$Y = AX$$

where Y and X are matrices with columns $y_2 - y_1, y_3 - y_1, \ldots$ and $x_2 - x_1, x_3 - x_1, \ldots$ respectively.

Linear block cipher

A known plaintext attack means that pairs $(x_1, y_1), (x_2, y_2), \ldots$ are available to Eve.

We can eliminate b:

$$\begin{aligned} y_2 - y_1 &= A(x_2 - x_1) \\ y_3 - y_1 &= A(x_3 - x_1) \\ &\vdots \end{aligned}$$

Can we compute A?

Not from a single equation, but if we put together several, then we get

$$Y = AX$$

where Y and X are matrices with columns $y_2 - y_1, y_3 - y_1, \ldots$ and $x_2 - x_1, x_3 - x_1, \ldots$ respectively.

If X is $N \times N$ and invertible, then

$$A = YX^{-1}$$

Linear block cipher

Once we have A, we can compute b from

$$b = y_1 - Ax_1$$

Linear block cipher

Once we have A, we can compute b from

$$b = y_1 - Ax_1$$

Can this attack be prevented by using the same key k only a limited number of times?

Linear block cipher

Once we have A, we can compute b from

$$b = y_1 - Ax_1$$

Can this attack be prevented by using the same key k only a limited number of times?

Actually yes; for the matrix inversion, we needed $N + 1$ (x_i, y_i) pairs. With only N pairs available, this attack cannot fully obtain A and b.

Linear block cipher

Once we have A, we can compute b from

$$b = y_1 - Ax_1$$

Can this attack be prevented by using the same key k only a limited number of times?

Actually yes; for the matrix inversion, we needed $N + 1$ (x_i, y_i) pairs. With only N pairs available, this attack cannot fully obtain A and b.

So if each $k = (A, b)$ is used only N times, the success of this attack can be limited.

Linear block cipher

Once we have A, we can compute b from

$$b = y_1 - Ax_1$$

Can this attack be prevented by using the same key k only a limited number of times?

Actually yes; for the matrix inversion, we needed $N + 1$ (x_i, y_i) pairs. With only N pairs available, this attack cannot fully obtain A and b.

So if each $k = (A, b)$ is used only N times, the success of this attack can be limited.

Two practical issues:

- some information is obtained about the key;
- the key needs to be changed frequently.

We will address both issues soon.

One-time pad

Information-theoretic security or *unconditional security* means that using a ciphertext only attack, the attacker can obtain no information about the key or the plaintext.

One-time pad

Information-theoretic security or *unconditional security* means that using a ciphertext only attack, the attacker can obtain no information about the key or the plaintext.

One time pad (OTP): both the sender and the receiver have the same random bit sequence k; the encryption is bitwise addition of the message and the key.

Example:

$$\begin{array}{rcl} \mathrm{x} & = & 01001101 \\ +\mathrm{k} & = & 11010000 \\ \hline \mathrm{y} & = & 10011101 \end{array}$$

One-time pad

Theorem

As long as the key is used only once, OTP offers unconditional security.

One-time pad

Theorem

As long as the key is used only once, OTP offers unconditional security.

Proof. If the key is a random bit sequence, for a ciphertext y, any x plaintexts will be equally likely.

One-time pad

Theorem

As long as the key is used only once, OTP offers unconditional security.

Proof. If the key is a random bit sequence, for a ciphertext y, any x plaintexts will be equally likely.

Theorem

OTP is essentially the only cryptography protocol that offers unconditional security.

One-time pad

Theorem

As long as the key is used only once, OTP offers unconditional security.

Proof. If the key is a random bit sequence, for a ciphertext y, any x plaintexts will be equally likely.

Theorem

OTP is essentially the only cryptography protocol that offers unconditional security.

Proof (sketch). The key must have entropy at least equal to entropy of the message; in other words, the key must be at least as long as the plaintext and at least as random as the plaintext.

Computational security

The issue with one-time pad is that when encrypting, the same key can only be used once; if it is used more than once, some information may be derived from it.

Computational security

The issue with one-time pad is that when encrypting, the same key can only be used once; if it is used more than once, some information may be derived from it.

So either Alice and Bob both need to know a long common secret key, or they have to arrange a secure key exchange often. Neither is very practical.

Computational security

The issue with one-time pad is that when encrypting, the same key can only be used once; if it is used more than once, some information may be derived from it.

So either Alice and Bob both need to know a long common secret key, or they have to arrange a secure key exchange often. Neither is very practical.

Unconditional security is too strict. In practice, we settle for *computational security*: a protocol is computationally secure if attacking it would require an infeasibly long time with available computing systems and known algorithms.

Computational security

The issue with one-time pad is that when encrypting, the same key can only be used once; if it is used more than once, some information may be derived from it.

So either Alice and Bob both need to know a long common secret key, or they have to arrange a secure key exchange often. Neither is very practical.

Unconditional security is too strict. In practice, we settle for *computational security*: a protocol is computationally secure if attacking it would require an infeasibly long time with available computing systems and known algorithms.

Computationally secure protocols are usually based on mathematical problems that are known to be computationally complex.

Computational security

Computational security is not a rigorous definition. It reflects the current state of the art, both in terms of known algorithms to solve specific problems and also in terms of currently available computational power.

Computational security

Computational security is not a rigorous definition. It reflects the current state of the art, both in terms of known algorithms to solve specific problems and also in terms of currently available computational power.

As time progresses, either of those might change (e.g. due to quantum computers), and the definition of computational security might need to be updated accordingly.

Computational security

Computational security is not a rigorous definition. It reflects the current state of the art, both in terms of known algorithms to solve specific problems and also in terms of currently available computational power.

As time progresses, either of those might change (e.g. due to quantum computers), and the definition of computational security might need to be updated accordingly.

It is also not a precise definition as it depends on what mathematical problems are considered computationally complex. Generally, it is required that no polynomial time algorithm is known.

Computational security

Computational security is not a rigorous definition. It reflects the current state of the art, both in terms of known algorithms to solve specific problems and also in terms of currently available computational power.

As time progresses, either of those might change (e.g. due to quantum computers), and the definition of computational security might need to be updated accordingly.

It is also not a precise definition as it depends on what mathematical problems are considered computationally complex. Generally, it is required that no polynomial time algorithm is known.

NP-hard problems are typically good candidates, but even in that case, $\mathrm{P} \neq \mathrm{NP}$ is not actually proven, just a conjecture.

Computationally complex problems

Some computationally complex mathematical problems that are used in cryptography:

- Integer factorization. Given an integer number n that is the product of two primes, compute the two primes. Used in the RSA algorithm.

Computationally complex problems

Some computationally complex mathematical problems that are used in cryptography:

- Integer factorization. Given an integer number n that is the product of two primes, compute the two primes. Used in the RSA algorithm.
- Discrete logarithm problem. Given g and g^x mod n, compute x. Used in the Digital signature algorithm (DSA).

Computationally complex problems

Some computationally complex mathematical problems that are used in cryptography:

- Integer factorization. Given an integer number n that is the product of two primes, compute the two primes. Used in the RSA algorithm.
- Discrete logarithm problem. Given g and g^x mod n, compute x. Used in the Digital signature algorithm (DSA).
- Multivariate quadratic problem. Solve a system of multivariate quadratic equations over GF(q). Used in the Rainbow signature scheme.

Computationally complex problems

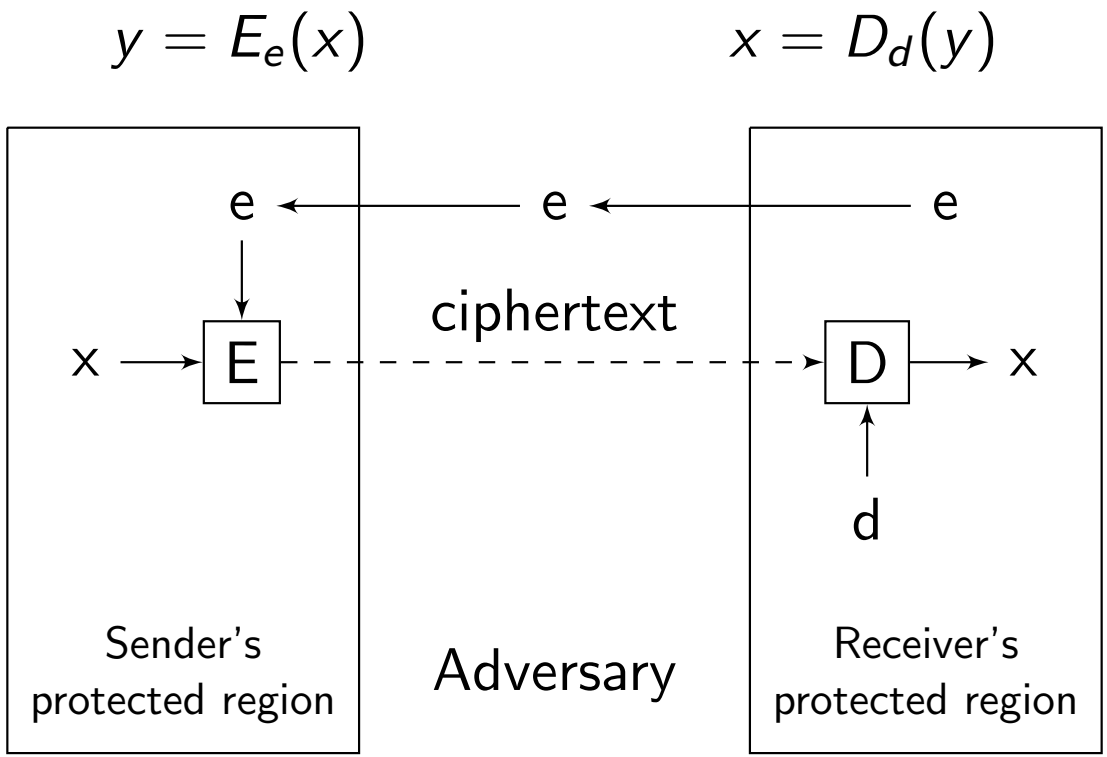
Some computationally complex mathematical problems that are used in cryptography:

- Integer factorization. Given an integer number n that is the product of two primes, compute the two primes. Used in the RSA algorithm.
- Discrete logarithm problem. Given g and g^x mod n, compute x. Used in the Digital signature algorithm (DSA).
- Multivariate quadratic problem. Solve a system of multivariate quadratic equations over GF(q). Used in the Rainbow signature scheme.
- Shortest vector problem. Given an n-dimensional lattice, find the shortest lattice vector. Used in post-quantum secure cryptosystems.

Public key cryptography

Instead of a common key k which is known by both the sender and the receiver, *public key cryptography* works the following way:

- the receiver has a $k = (d, e)$ pair of keys
- d is a private key known only by the receiver
- e is a public key known by everyone



RSA algorithm

The Rivest–Shamir–Adleman (RSA) algorithm is a public key protocol.

RSA algorithm

The Rivest–Shamir–Adleman (RSA) algorithm is a public key protocol.

Key generation:

- select 2 large primes p and q; $n = pq$.
- $m = (p-1)(q-1)$.
- Select a coding exponent e so that $\gcd(e, m) = 1$ and $1 < e < m$.
- Solve $de = 1 \mod m$ to obtain the decoding key d.
- (n, e) is the public key;
- p, q, m and d are kept secret.

RSA algorithm

Encryption (using the public key):

- the plaintext is cut into sections which can be turned into numbers x such that $0 \leq x < n$.
- the ciphertext is $y = x^e \mod n$.

RSA algorithm

Encryption (using the public key):

- the plaintext is cut into sections which can be turned into numbers x such that $0 \leq x < n$.
- the ciphertext is $y = x^e \mod n$.

Decryption:

- $x = y^d \mod n$.

RSA algorithm

Why does the RSA algorithm work?

RSA algorithm

Why does the RSA algorithm work?

RSA is useful only if ALL of the following properties hold:

1. Key generation is computationally simple

RSA algorithm

Why does the RSA algorithm work?

RSA is useful only if ALL of the following properties hold:

1. Key generation is computationally simple
2. Encryption using the public key is computationally simple

RSA algorithm

Why does the RSA algorithm work?

RSA is useful only if ALL of the following properties hold:

1. Key generation is computationally simple
2. Encryption using the public key is computationally simple
3. Decryption using the private key is computationally simple

RSA algorithm

Why does the RSA algorithm work?

RSA is useful only if ALL of the following properties hold:

1. Key generation is computationally simple
2. Encryption using the public key is computationally simple
3. Decryption using the private key is computationally simple
4. Encryption and decryption are indeed inverse operations

RSA algorithm

Why does the RSA algorithm work?

RSA is useful only if ALL of the following properties hold:

1. Key generation is computationally simple
2. Encryption using the public key is computationally simple
3. Decryption using the private key is computationally simple
4. Encryption and decryption are indeed inverse operations
5. Attacking without the private key is computationally complex

RSA algorithm

1. Key generation is computationally simple:
   - Primality testing (checking whether a given number is a prime or not) is computationally simple.
   - There are many primes even among large numbers: the Prime Number Theorem states that among numbers of order N, on average 1 out of $\log(N)$ numbers is a prime.
   - So *finding* large prime numbers for p and q is easy: we can just start prime checking large numbers randomly, and we will soon find two large prime numbers.
   - gcd and $de = 1 \mod m$ can be solved fast using the Extended Euclidean Algorithm.

Extended Euclidean Algorithm

The Extended Euclidean Algorithm can be used to find $\gcd(a, b)$ and also to solve

$$\gcd(a, b) = s \cdot a + t \cdot b.$$

Extended Euclidean Algorithm

The Extended Euclidean Algorithm can be used to find $\gcd(a, b)$ and also to solve

$$\gcd(a, b) = s \cdot a + t \cdot b.$$

Assume $a > b$; initialize $r_0 = a, r_1 = b$ and also $s_0 = 1, t_0 = 0, s_1 = 0, t_1 = 1$. In each step, we write

$$r_{k-1} = r_k \cdot q_{k+1} + r_{k+1} \qquad r_k = s_k \cdot a + t_k \cdot b,$$

where $0 \leq r_{k+1} < r_k$, and s_{k+1} and t_{k+1} are computed from

$$s_{k+1} = s_{k-1} - q_k s_k, \qquad t_{k+1} = t_{k-1} - q_k t_k.$$

Extended Euclidean Algorithm

The Extended Euclidean Algorithm can be used to find $\gcd(a, b)$ and also to solve

$$\gcd(a, b) = s \cdot a + t \cdot b.$$

Assume $a > b$; initialize $r_0 = a, r_1 = b$ and also $s_0 = 1, t_0 = 0, s_1 = 0, t_1 = 1$. In each step, we write

$$r_{k-1} = r_k \cdot q_{k+1} + r_{k+1} \qquad r_k = s_k \cdot a + t_k \cdot b,$$

where $0 \leq r_{k+1} < r_k$, and s_{k+1} and t_{k+1} are computed from

$$s_{k+1} = s_{k-1} - q_k s_k, \qquad t_{k+1} = t_{k-1} - q_k t_k.$$

The algorithm stops when $r_{k+1} = 0$; then $r_k = \gcd(a, b)$, and $\gcd(a, b) = s_k \cdot a + t_k \cdot b$.

Extended Euclidean Algorithm

The Extended Euclidean Algorithm can be used to find $\gcd(a, b)$ and also to solve

$$\gcd(a, b) = s \cdot a + t \cdot b.$$

Assume $a > b$; initialize $r_0 = a, r_1 = b$ and also $s_0 = 1, t_0 = 0, s_1 = 0, t_1 = 1$. In each step, we write

$$r_{k-1} = r_k \cdot q_{k+1} + r_{k+1} \qquad r_k = s_k \cdot a + t_k \cdot b,$$

where $0 \leq r_{k+1} < r_k$, and s_{k+1} and t_{k+1} are computed from

$$s_{k+1} = s_{k-1} - q_k s_k, \qquad t_{k+1} = t_{k-1} - q_k t_k.$$

The algorithm stops when $r_{k+1} = 0$; then $r_k = \gcd(a, b)$, and $\gcd(a, b) = s_k \cdot a + t_k \cdot b$.

For $\gcd(n, e) = 1$, the algorithm gives $1 = \gcd(n, e) = s \cdot n + t \cdot e$, so $e^{-1} = t \bmod n$.

Extended Euclidean Algorithm

Compute the greatest common divisor (gcd) of $a = 8387$ and $b = 1243$, and also compute s and t so that

$$\gcd(8387, 1243) = s \cdot 8387 + t \cdot 1243.$$

Extended Euclidean Algorithm

Compute the greatest common divisor (gcd) of $a = 8387$ and $b = 1243$, and also compute s and t so that

$$\gcd(8387, 1243) = s \cdot 8387 + t \cdot 1243.$$

Solution.

$$8387 = 1243 \cdot 6 + 929 \qquad 929 = a - 6b$$

Extended Euclidean Algorithm

Compute the greatest common divisor (gcd) of $a = 8387$ and $b = 1243$, and also compute s and t so that

$$\gcd(8387, 1243) = s \cdot 8387 + t \cdot 1243.$$

Solution.

$$\begin{aligned} 8387 &= 1243 \cdot 6 + 929 & \qquad 929 &= a - 6b \\ 1243 &= 929 \cdot 1 + 314 & \qquad 314 &= -a + 7b \end{aligned}$$

Extended Euclidean Algorithm

Compute the greatest common divisor (gcd) of $a = 8387$ and $b = 1243$, and also compute s and t so that

$$\gcd(8387, 1243) = s \cdot 8387 + t \cdot 1243.$$

Solution.

$$\begin{aligned} 8387 &= 1243 \cdot 6 + 929 \\ 1243 &= 929 \cdot 1 + 314 \\ 929 &= 314 \cdot 2 + 301 \end{aligned} \qquad \begin{aligned} 929 &= a - 6b \\ 314 &= -a + 7b \\ 301 &= 3a - 20b \end{aligned}$$

Extended Euclidean Algorithm

Compute the greatest common divisor (gcd) of $a = 8387$ and $b = 1243$, and also compute s and t so that

$$\gcd(8387, 1243) = s \cdot 8387 + t \cdot 1243.$$

Solution.

$$\begin{aligned} 8387 &= 1243 \cdot 6 + 929 \\ 1243 &= 929 \cdot 1 + 314 \\ 929 &= 314 \cdot 2 + 301 \\ 314 &= 301 \cdot 1 + 13 \end{aligned} \qquad \begin{aligned} 929 &= a - 6b \\ 314 &= -a + 7b \\ 301 &= 3a - 20b \\ 13 &= -4a + 27b \end{aligned}$$

Extended Euclidean Algorithm

Compute the greatest common divisor (gcd) of $a = 8387$ and $b = 1243$, and also compute s and t so that

$$\gcd(8387, 1243) = s \cdot 8387 + t \cdot 1243.$$

Solution.

$$\begin{aligned} 8387 &= 1243 \cdot 6 + 929 \\ 1243 &= 929 \cdot 1 + 314 \\ 929 &= 314 \cdot 2 + 301 \\ 314 &= 301 \cdot 1 + 13 \\ 301 &= 13 \cdot 23 + 2 \end{aligned} \qquad \begin{aligned} 929 &= a - 6b \\ 314 &= -a + 7b \\ 301 &= 3a - 20b \\ 13 &= -4a + 27b \\ 2 &= 95a - 641b \end{aligned}$$

Extended Euclidean Algorithm

Compute the greatest common divisor (gcd) of $a = 8387$ and $b = 1243$, and also compute s and t so that

$$\gcd(8387, 1243) = s \cdot 8387 + t \cdot 1243.$$

Solution.

$$\begin{aligned}
8387 &= 1243 \cdot 6 + 929 \\
1243 &= 929 \cdot 1 + 314 \\
929 &= 314 \cdot 2 + 301 \\
314 &= 301 \cdot 1 + 13 \\
301 &= 13 \cdot 23 + 2 \\
13 &= 2 \cdot 6 + 1
\end{aligned}
\qquad
\begin{aligned}
929 &= a - 6b \\
314 &= -a + 7b \\
301 &= 3a - 20b \\
13 &= -4a + 27b \\
2 &= 95a - 641b \\
1 &= -574a + 3873b
\end{aligned}$$

Extended Euclidean Algorithm

Compute the greatest common divisor (gcd) of $a = 8387$ and $b = 1243$, and also compute s and t so that

$$\gcd(8387, 1243) = s \cdot 8387 + t \cdot 1243.$$

Solution.

$$\begin{aligned}
8387 &= 1243 \cdot 6 + 929 \\
1243 &= 929 \cdot 1 + 314 \\
929 &= 314 \cdot 2 + 301 \\
314 &= 301 \cdot 1 + 13 \\
301 &= 13 \cdot 23 + 2 \\
13 &= 2 \cdot 6 + 1 \\
2 &= 1 \cdot 2 + 0.
\end{aligned}$$

$$\begin{aligned}
929 &= a - 6b \\
314 &= -a + 7b \\
301 &= 3a - 20b \\
13 &= -4a + 27b \\
2 &= 95a - 641b \\
1 &= -574a + 3873b
\end{aligned}$$

Finally,

$$\gcd(8387, 1243) = 1 = -574 \cdot 8387 + 3873 \cdot 1243.$$

Extended Euclidean algorithm

Compute the multiplicative inverse of 1243 mod 8387.

Extended Euclidean algorithm

Compute the multiplicative inverse of 1243 mod 8387.

From the Extended Euclidean algorithm,

$$\gcd(8387, 1243) = 1 = -574 \cdot 8387 + 3873 \cdot 1243;$$

this means

$$3873 \cdot 1243 = 1 \mod 8387,$$

so

$$1243^{-1} = 3873 \mod 8387.$$

Extended Euclidean algorithm

Compute the multiplicative inverse of 1243 mod 8387.

From the Extended Euclidean algorithm,

$$\gcd(8387, 1243) = 1 = -574 \cdot 8387 + 3873 \cdot 1243;$$

this means

$$3873 \cdot 1243 = 1 \mod 8387,$$

so

$$1243^{-1} = 3873 \mod 8387.$$

Theorem

The Extended Euclidean algorithm takes no more than $\log_{1.618}(\min(a, b))$ *steps.*

(No proof.)

Extended Euclidean algorithm

Compute the multiplicative inverse of 1243 mod 8387.

From the Extended Euclidean algorithm,

$$\gcd(8387, 1243) = 1 = -574 \cdot 8387 + 3873 \cdot 1243;$$

this means

$$3873 \cdot 1243 = 1 \mod 8387,$$

so

$$1243^{-1} = 3873 \mod 8387.$$

Theorem

The Extended Euclidean algorithm takes no more than $\log_{1.618}(\min(a, b))$ *steps.*

(No proof.)

Any guess for what pairs of numbers it is slowest for, and where does the value 1.618 come from?

RSA algorithm

2-3. Encryption and decryption are computationally simple.

$x^e \bmod n$ (and $y^d \bmod n$) can be computed using *Fast modular exponentiation*:

- Write e in base 2 as

$$e = 2^{b_1} + 2^{b_2} + \cdots + 2^{b_\ell}$$

  where $b_1 > b_2 > \cdots > b_\ell$ are integers.
- Compute $x^2 \bmod n, x^4 \bmod n, \ldots x^{2^{b_1}} \bmod n$ by taking the square of the previous term mod n in each step.
- Compute $x^e \bmod n$ as

$$x^e = x^{2^{b_1}} \cdot x^{2^{b_2}} \cdots x^{2^{b_\ell}} \mod n$$

Fast modular exponentiation

Compute $23^{42} \bmod 131$.

Fast modular exponentiation

Compute $23^{42} \bmod 131$.

- $42 = 32 + 8 + 2$.

Fast modular exponentiation

Compute $23^{42} \bmod 131$.

- $42 = 32 + 8 + 2$.
- $$23^2 = 529 = 5 \mod 131$$

Fast modular exponentiation

Compute $23^{42} \bmod 131$.

- $42 = 32 + 8 + 2$.
- $$\begin{array}{llll} 23^2 & = & 529 & = 5 \mod 131 \\ 23^4 & = 5^2 & & = 25 \mod 131 \end{array}$$

Fast modular exponentiation

Compute $23^{42} \bmod 131$.

- $42 = 32 + 8 + 2$.
- $$\begin{array}{llll} 23^2 & = & 529 & = 5 \mod 131 \\ 23^4 & = 5^2 & & = 25 \mod 131 \\ 23^8 & = 25^2 = & 625 & = 101 \mod 131 \end{array}$$

Fast modular exponentiation

Compute $23^{42} \bmod 131$.

- $42 = 32 + 8 + 2$.
-

$$
\begin{array}{llll}
23^2 & = & 529 & = 5 \mod 131 \\
23^4 & = 5^2 & & = 25 \mod 131 \\
23^8 & = 25^2 = & 625 & = 101 \mod 131 \\
23^{16} & = 101^2 = & 10201 & = 114 \mod 131
\end{array}
$$

Fast modular exponentiation

Compute $23^{42} \bmod 131$.

- $42 = 32 + 8 + 2$.
-

$$
\begin{array}{llll}
23^2 & = & 529 & = 5 \mod 131 \\
23^4 & = 5^2 & & = 25 \mod 131 \\
23^8 & = 25^2 = & 625 & = 101 \mod 131 \\
23^{16} & = 101^2 = & 10201 & = 114 \mod 131 \\
23^{32} & = 114^2 = & 12296 & = 27 \mod 131
\end{array}
$$

Fast modular exponentiation

Compute $23^{42} \bmod 131$.

- $42 = 32 + 8 + 2$.
- $$\begin{array}{llll} 23^2 & = & 529 & = 5 \mod 131 \\ 23^4 & = 5^2 & & = 25 \mod 131 \\ 23^8 & = 25^2 = & 625 & = 101 \mod 131 \\ 23^{16} & = 101^2 = & 10201 & = 114 \mod 131 \\ 23^{32} & = 114^2 = & 12296 & = 27 \mod 131 \end{array}$$
- $$23^{42} = 23^{32} \cdot 23^8 \cdot 23^2 = 27 \cdot 101 \cdot 5 = 11 \mod 131$$

RSA algorithm

4. Encryption and decryption are indeed inverse operations.

Let $\phi(n)$ be Euler's totient function:

$$\begin{aligned} n &= p_1^{k_1} \cdots p_r^{k_r} \\ \phi(n) &= p_1^{k_1-1}(p_1-1) \cdots p_r^{k_r-1}(p_r-1), \end{aligned}$$

specifically, if $n = pq$, then $\phi(n) = (p-1)(q-1)$.

RSA algorithm

4. Encryption and decryption are indeed inverse operations.

Let $\phi(n)$ be Euler's totient function:

$$\begin{aligned} n &= p_1^{k_1} \cdots p_r^{k_r} \\ \phi(n) &= p_1^{k_1-1}(p_1-1) \cdots p_r^{k_r-1}(p_r-1), \end{aligned}$$

specifically, if $n = pq$, then $\phi(n) = (p-1)(q-1)$.

Theorem (Euler–Fermat)

If gcd(x, n)=1, then

$$x^{\phi(n)} = 1 \mod n$$

(No proof.)

RSA algorithm

4. Encryption and decryption are indeed inverse operations.

Let $\phi(n)$ be Euler's totient function:

$$\begin{aligned} n &= p_1^{k_1} \cdots p_r^{k_r} \\ \phi(n) &= p_1^{k_1-1}(p_1-1) \cdots p_r^{k_r-1}(p_r-1), \end{aligned}$$

specifically, if $n = pq$, then $\phi(n) = (p-1)(q-1)$.

Theorem (Euler–Fermat)

If gcd(x, n)=1, then

$$x^{\phi(n)} = 1 \mod n$$

(No proof.)

For RSA, decryption and encryption are indeed inverse operations:

$$de = 1 \mod \phi(n) \quad \Longrightarrow \quad x^{de} = x \mod n.$$

RSA algorithm

5. Attacking without the private key is computationally complex.

Integer factorization is computationally difficult for large numbers. This is not a proven fact; it's just that no polynomial time algorithm is known. (Note that despite this, primality testing is computationally simple! That is, we can easily tell if a number is a prime or not, but if it is not a prime, we cannot factor it.)

RSA algorithm

5. Attacking without the private key is computationally complex.

Integer factorization is computationally difficult for large numbers. This is not a proven fact; it's just that no polynomial time algorithm is known. (Note that despite this, primality testing is computationally simple! That is, we can easily tell if a number is a prime or not, but if it is not a prime, we cannot factor it.)

So even though n is public, p and q cannot be computed efficiently, and without p and q, $m = \phi(n)$ and d cannot be computed either. Overall, if p and q are sufficiently large, attacking RSA is computationally infeasible.

RSA algorithm

5. Attacking without the private key is computationally complex.

Integer factorization is computationally difficult for large numbers. This is not a proven fact; it's just that no polynomial time algorithm is known. (Note that despite this, primality testing is computationally simple! That is, we can easily tell if a number is a prime or not, but if it is not a prime, we cannot factor it.)

So even though n is public, p and q cannot be computed efficiently, and without p and q, $m = \phi(n)$ and d cannot be computed either. Overall, if p and q are sufficiently large, attacking RSA is computationally infeasible.

Currently, sufficiently large means prime numbers with 150+ digits (in base 10).

RSA algorithm

5. Attacking without the private key is computationally complex.

Integer factorization is computationally difficult for large numbers. This is not a proven fact; it's just that no polynomial time algorithm is known. (Note that despite this, primality testing is computationally simple! That is, we can easily tell if a number is a prime or not, but if it is not a prime, we cannot factor it.)

So even though n is public, p and q cannot be computed efficiently, and without p and q, $m = \phi(n)$ and d cannot be computed either. Overall, if p and q are sufficiently large, attacking RSA is computationally infeasible.

Currently, sufficiently large means prime numbers with 150+ digits (in base 10).

(1991, RSA challenge to factorize 54 numbers; currently 23 has been factorized, the largest has 250 digits.)