Modern Cryptography

Private Key Encryption Scheme

Shashank Singh

IISER Bhopal

August 7, 2025

Setting of Private-key Cryptography..

Classical Cryptography

- Before sending the message (plaintext) m, Alice transforms (encrypts) it into a message c (ciphertext), using an algorithm `ENC` and a key k.
- Bob, on receiving c, decrypts it to get m, using a corresponding algorithm `DEC` and the same key k.

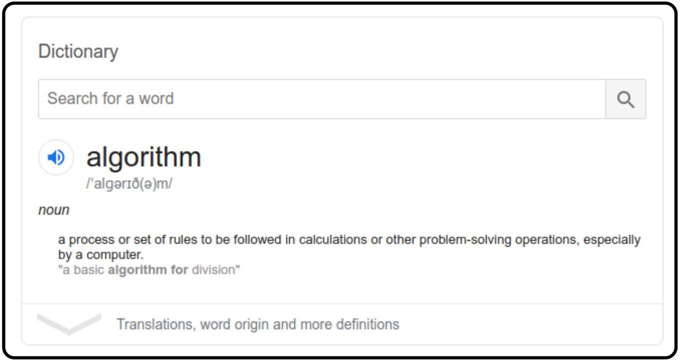
Setting of Private-key Cryptography..

Classical Cryptography

- The key k, needs to be (somehow) shared between the two communicating parties in advance and it is not known to the adversary.
- Alice and Bob could be same. Recall the disk encryption, where the same party encrypts the data on a disk and later decrypts it to get back the data.

ALGORITHMS

-a step by step solution to a problem.



ALGORITHMS

Important features of an algorithm

1. Finiteness: An algorithm must always terminate after a finite number of steps.
2. Definiteness: Each step of an algorithm must be precisely defined.
3. Input: An algorithm has zero or more inputs: quantities that are given to it initially before the algorithm begins, or dynamically as the algorithm runs.
4. Output: An algorithm has one or more outputs: quantities that have a specified relation to the inputs.
5. Effectiveness: Its operations must all be sufficiently basic that they can in principle be done exactly and in a finite length of time by someone using pencil and paper.

(TIME) COMPLEXITY OF AN ALGORITHM

```
def is_prime(n):
    for a in primes_upto(sqrt(n)):
        if (a divides n):
            return False
    return True
```

- The time complexity deals with how fast or slow a particular algorithm performs.
- We define it as a numerical function $T(n)$, which represent the running time of the algorithm as a function of input size (in bits) n.
- But $T(n)$ depends on the implementation! A given algorithm will take different amounts of time on the same inputs depending factors as: processor speed; instruction set, disk speed, brand of compiler and etc. So we want to define $T(n)$, which do not depend on the above factors.

(Time) Complexity of an Algorithm

```
def is_prime(n):
     for a in primes_upto(sqrt(n)):
          if (a divides n):
               return False
     return True
```

- The time complexity deals with how fast or slow a particular algorithm performs.
- We define it as a numerical function $T(n)$, which represent the running time of the algorithm as a function of input size (in bits) n.

- The way around is to estimate efficiency of each algorithm asymptotically. We will measure time $T(n)$ as the number of elementary "steps" (in a model of computation), provided each such step takes constant time.

Asymptotic Notations

In this section we will consider functions that have $\mathbb{N}$ as their domain and $\mathbb{R}_{\geq 0}$ as the range.

Definition (O-notation)

We say that a function $f(n)$ is *big-oh* of $g(n)$, written as $f(x) = O(g(n))$, if there exists positive constants c and n_0 such that

$$0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0.$$

In other words $O(g(n))$ denotes a set of functions that satisfy the above.

Remark

If $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ *exists and* $\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq \infty$ *then* $f(n) = O(g(n))$.

Asymptotic Notations..

Example

- Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$.
- Show that n^2 is not $O(n)$.

Asymptotic Notations..

Definition (o-notation)

We say that a function $f(n)$ is *small-oh* or *little-oh* of $g(n)$, written as $f(x) = o(g(n))$, if for any positive non zero constant c, there exist a positive constant n_0 such that

$$0 \leq f(n) < c \cdot g(n) \text{ for all } n \geq n_0.$$

In other words $o(g(n))$ denotes a set of functions that satisfy the above.

Remark

If $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ *exists and* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ *then* $f(n) = o(g(n))$.

Example

Let $f(n) = n^2$, then $f(n) \neq o(n^2)$ but $f(n) = o(n^2 \log n)$.

Asymptotic Notations..

Definition (o-notation)

We say that a function $f(n)$ is *small-oh* or *little-oh* of $g(n)$, written as $f(x) = o(g(n))$, if for any positive non zero constant c, there exist a positive constant n_0 such that

$$0 \leq f(n) < c \cdot g(n) \text{ for all } n \geq n_0.$$

In other words $o(g(n))$ denotes a set of functions that satisfy the above.

Remark

If $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ *exists and* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ *then* $f(n) = o(g(n))$.

Example

Let $f(n) = n^2$, then $f(n) \neq o(n^2)$ but $f(n) = o(n^2 \log n)$.

Asymptotic Notations..

Definition (o-notation)

We say that a function $f(n)$ is *small-oh* or *little-oh* of $g(n)$, written as $f(x) = o(g(n))$, if for any positive non zero constant c, there exist a positive constant n_0 such that

$$0 \leq f(n) < c \cdot g(n) \text{ for all } n \geq n_0.$$

In other words $o(g(n))$ denotes a set of functions that satisfy the above.

Remark

If $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ *exists and* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ *then* $f(n) = o(g(n))$.

Example

Let $f(n) = n^2$, then $f(n) \neq o(n^2)$ but $f(n) = o(n^2 \log n)$.

Asymptotic Notations..

Definition (Ω-notation)

We say that a function $f(n)$ is *big-omega* of $g(n)$, written as $f(x) = \Omega(g(n))$, if there exists positive constants c and n_0 such that

$$0 \le c \cdot g(n) \le f(n) \text{ for all } n \ge n_0.$$

In other words $\Omega(g(n))$ denotes a set of functions that satisfy the above.

Remark

If $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ *exists and* $\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0$ *then* $f(n) = \Omega(g(n))$.

Asymptotic Notations..

Definition (ω-notation)

We say that a function $f(n)$ is *little-omega* of $g(n)$, written as $f(x) = \omega(g(n))$, if for any positive non zero constant c, there exist a positive nonzero constant n_0 such that

$$0 \leq c \cdot g(n) < f(n) \text{ for all } n \geq n_0.$$

In other words $\Omega(g(n))$ denotes a set of functions that satisfy the above.

Remark

If $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ *exists and* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ *then* $f(n) = \omega(g(n))$.

Θ-NOTATION

Definition (Θ-Notation)

We say that a function $f(n)$ is theta of $g(n)$, written as $f(n) = \Theta(g(n))$, if there exists positive constants c_1, c_2 and n_0 such that

$$0 \leq c_2 \cdot g(n) \leq f(n) \leq c_1 \cdot g(n) \text{ for all } n \geq n_0.$$

Remark

If $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ *exists and* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$*, where* c *is a non-zero positive constant, then* $f(n) = \Theta(g(n))$*.*

TERMINOLOGY FOR COMPLEXITY OF ALGORITHMS

Complexity (input size is n)	Terminology
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	Linearithmic complexity
$\Theta(n^b)$, for positive integer b	Polynomial complexity
$\Theta(b^n)$, where $b > 1$	Exponential complexity
$\Theta(n!)$	Factorial Complexity

Efficient

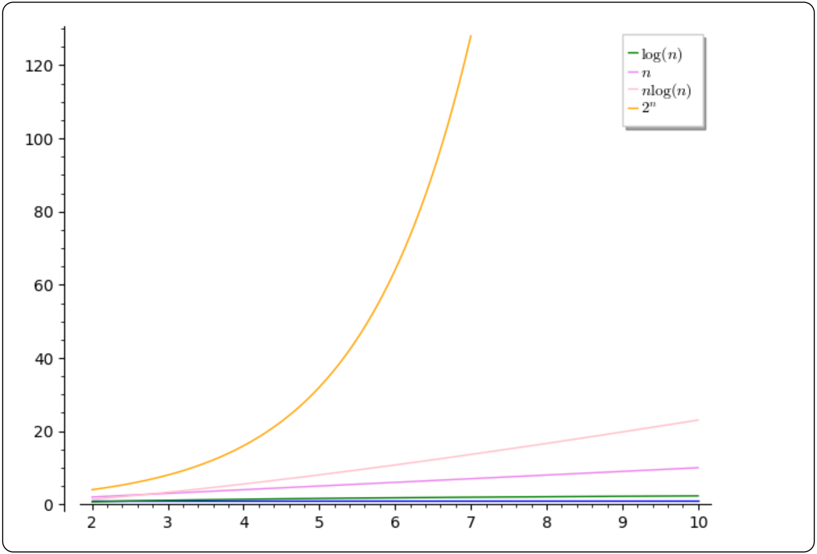
- Algorithms with time complexity $\Theta(n^b)$, where n is input size and b is a non-zero positive integer, are called polynomial time algorithms.

TERMINOLOGY FOR COMPLEXITY OF ALGORITHMS

Complexity (input size is n)	Terminology
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	Linearithmic complexity
$\Theta(n^b)$, for positive integer b	Polynomial complexity
$\Theta(b^n)$, where $b > 1$	Exponential complexity
$\Theta(n!)$	Factorial Complexity

Efficient (Constant, Logarithmic, Linear, Linearithmic, Polynomial complexity)

- Algorithms with time complexity $\Theta(n^b)$, where n is input size and b is a non-zero positive integer, are called polynomial time algorithms.



$\log(n)$
n
$n\log(n)$
2^n
120
100
80
60
40
20
0
2
3
4
5
6
7
8
9
10

PPT- Probabilistic Polynomial Time Algorithm

- The polynomial time algorithm is an algorithms with time complexity $O(n^b)$, where n is input size and b is a fixed non zero positive integer.
- A probabilistic algorithm is one that has the capability of "tossing coins", i.e. the algorithm has access to a random source of randomness that yields unbiased random bits that are independently equal to 1 with $\frac{1}{2}$ probability and to 0 with $\frac{1}{2}$ probability.
- A probabilistic polynomial-time algorithm is a probabilistic algorithm that may only perform a polynomial amount of operations including at most a polynomial number of coin-flips.

Private Key Encryption

Let $\mathcal{M}, \mathcal{K}$ and $\mathcal{C}$ represent the set of possible messages (plaintexts), the set of possible keys and the set of possible ciphertexts repectively.

Private Key Encryption..

A private key encryption algorithm is basically a set of three algorithms (we will be more specific later) $(\mathtt{GEN}, \mathtt{ENC}, \mathtt{DEC})$, which have the following functionalities:

- GEN It is a pobabilistic algorithm, called key generation algorithm. It outputs a key $k \in \mathcal{K}$ chosen according to some distribution.
- ENC It is called encryption algorithm. It takes as input a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$ and outputs a ciphertext $c \in \mathcal{C}$.
- DEC It is known as decryption algorithm. It take as input a key k and a ciphertext c and outputs a plaintext m.

Furthermore, it must satisfy the following correctness requirements:

$$\mathtt{DEC}_k\left(\mathtt{ENC}_k\left(m\right)\right) = m \forall m \in \mathcal{M}, \forall k \in \mathcal{K}. \quad (1)$$

KERCKHOFFS' PRINCIPLE

✔ The cipher method must not be required to be secret, and it must be able to fall into the hands of the enmy without inconvenience.

- The security of a cryptographic scheme relies solely on the secrecy of the key, not on the secrecy of the underlying algorithms.

Caeser's Cipher (Shift Cipher)

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Let $\mathcal{M} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26} := \{0, 1, \ldots, 25\}$.
For $0 \leq k \leq 25$, define

$$\texttt{ENC}_k(m) = (m + k) \mod 26$$

and

$$\texttt{DEC}_k(c) = (c - k) \mod 26$$

- Caeser's Cipher is the oldest recorded cipher, which is a Shift Cipher with the key $k = 3$.