Deciphering the Mysteries

SoS 2022(Cryptography)

Arya Vishe
21D070018, IIT-B

Mentor- Param Rathour

Photos/Pic_1.png

12/05/2022

Contents

Week 1

Introduction to Cryptography

1.1.1 Some terminologies

- *Plaintext,* The message, usually alphabetic in this context, that you wish to send.

- *Ciphertext,* The message that will be transmitted after enciphering.
  Depicted using captial letters A, B, C, $\cdots$
- *Key,* The word/ phrase/ alphabetic order using which we encrypted our *Plaintext.*
- *Monoalphabetic Substitution Cipher,* A cipher key which uses only one set of alphabet to encrypt the *Plaintext*
- *Polyalphabetic Substitution Cipher,* A key that consists of multiple alphabetic orders/ keywords to encrypt the message.
- *Alice, Bob and Eve,* The quintessential trio studied in Cryptology examples wherein *Alice* and *Bob*, are trying to communicate secretly which *Eve* tries to intercept and decrypt the messages.

1.1.2 Early Cryptography

The earliest forms of monoalphabetic[1] ciphers include the *Caesarean/Shift Cipher.* This involves shifting of the alphabet by a few (historically 3) letter to encrypt the *Plain text.*
This yields us with only 26 possible cipher keys which can be easily cycled through by Eve, once she is sure that Caeser Cipher has been used, hence providing us with next none security.

[1]Consisting of only 1 key

Of course, there exist a very large number of random monoalphabetic cipher keys ($26! \approx 4 \times 10^{26}$ to be precise) but monoalphabetic ciphers can be easily broken into using statistical analysis of the cipher text. Tools employed by cryptanalysts include-

- Frequency of letters
- Frequency of Bigrams (pairs of letters that occur together)
- Presence of vowels

Another classical cipher, which used polyalphabetic cipher keys, *Vigenère* cipher will be analysed in later weeks.

Some Mathematical Pre-requisites for Modular Arithematic

1.2.1 GCD

Definition. GCD *of any 2 integers is the largest positive integer that divides each of the integers, provided both integers aren't 0.*

The standard procedure to find $\gcd(a, b)$ is to list out factors of a and b, and choose the largest factor common to both. But an even faster algorithm exists called the *Euclid's algorithm.* It uses the fact that $\gcd(a, b) = \gcd(a - k \cdot b, b), \forall a, b, k \in \mathbb{Z}$ such that $a > b$

Thus to find $\gcd(A, B)$, we can use this algorithm

1. Take $a = A$, $b = B$
2. If $b > a$, swap them.
3. $\gcd(a, b) = \gcd(a - k \cdot b)$ so replace a with $a - k \cdot b$ such that $(k+1) \cdot b > a$ and $k \cdot b < a$
4. If $a = 0$, $\gcd(A, B) = b$
   else swap a and b, and repeat from Step 3.

Some interesting results-

- $r_{n+2} < \frac{r_n}{2}$ where r_n is the n^{th} remainder[a].
- it takes at most $2 \log_2 b + 2$ steps[b].
- We can prove that $\gcd(a, b) = u \cdot a + v \cdot b$ where $u, v \in \mathbb{Z}$. These u, v values can be determined using a simple algorithm if we know the quotients at each step.
  We can even find the original a and b if we have all the quotients.

- The above statement also means that if $\gcd(a, b) = 1$, we can write any number using linear combinations of a and b.

[a]Proof- Assuming 2 cases (1. $r_{n+1} < \frac{r_n}{2}$ is obvious,2. $r_{n+1} > \frac{r_n}{2}$ ensures that $r_n = 1.r_{n+1} + r_{n+2} \Rightarrow r_{n+2} = r_n - r_{n+1} < \frac{r_n}{2}$)

[b]We can use the fact that if $b \in [2^{n-1}, 2^n)$ and $r_{2k+1} < \frac{b}{2^k}$, we can easily see that this is true since $2n + 1 = 2(n - 1) + 2 < 2\log_2 b + 2$

Modular Arithematic

1.3.1 Format

$$a \equiv b \bmod (m) \text{ where } a - b \text{ is divisible by } m$$

1.3.2 Properties

- If $a \cdot b \equiv 1 \bmod (m)\ \exists b \in \mathbb{Z} \iff \gcd(a, m) = 1$, we say that b is the *modular inverse* of a modulo m.
  <u>*Note-*</u> *Although multiple values of b will exist if $\gcd(a, m) = 1$, all will have the same value of $b \bmod (m)$. Hence we used the term **the inverse**.*
- If $a_1 \equiv b_1 \bmod (m)$ and $a_2 \equiv b_2 \bmod (m)$, then, $a_1 \cdot a_2 \equiv b_1 \cdot b_2 \bmod (m)$
- We can denote *inverse* of a as a^{-1} such that, $a^{-1} \equiv b \bmod (m)$.
  Hence $7^{-1} \equiv 8 \bmod (11)$. This means if we can write $\frac{4}{7} \equiv 7 \cdot 8 \bmod (11) \equiv 1 \bmod (11)$ as $4 \equiv 7 \bmod (11)$.
  The interesting thing to note here is that it isn't easy to find the modular inverses of numbers as m becomes larger.
- Since we can write $a \cdot b \equiv 1 \bmod (m)$ as $a \cdot b = q \cdot m + 1$, from the result obtained in last section, we can be sure that finding a value of b will take no. of steps $\sim \log_2 m$.

1.3.3 Ring theory

As we can see in the above definitions, it is obvious that in $a \equiv r \bmod (m)$, $r \in [0, m-1] \cap \mathbb{Z}$. We can then say that $r \in \mathbb{Z}/m\mathbb{Z}$ where $\mathbb{Z}/m\mathbb{Z}$ is the ring of integers with modulo m.

For addition and multiplication in $\mathbb{Z}/m\mathbb{Z}$ space, we just divide the answer with m and use the remainder as final value.
Units modulo m- If $\gcd(a, m) = 1$, OR a^{-1} exists for modulo m, we say that a is units modulo m.

- If a_1, a_2 are units modulo m, $a_1 \cdot a_2$ will also be a units modulo m.
- If a_1, a_2 are units modulo m, $a_1 + a_2$ need not be a units modulo m.

The set $*(\mathbb{Z}/m\mathbb{Z})$ is called the group of units modulo m and can also be defined as-
$*(\mathbb{Z}/m\mathbb{Z}) = a \in \mathbb{Z}/m\mathbb{Z}, \gcd(a, m) = 1$ If we took a subset of units modulo $m = \mathbf{P}$, then-

- If $k_{i,j} = a_i \cdot a_j$, where $a_i, a_j \in \mathbf{P}$, then $k_{i,j}$ also belongs to $\mathbf{P}$.
- If $k_{i,j} = a_i + a_j$, where $a_i, a_j \in \mathbf{P}$, then $k_{i,j}$ need not belong to $\mathbf{P}$.

Euler's Totient/ Phi Function- $\phi(m) = \# * (\mathbb{Z}/m\mathbb{Z})$ OR $\phi(m)$ counts the number of $a \in [1, m-1]$, such that, $\gcd(a, m) = 1$ i.e. number of values of $a < m$ such that a and m are co-prime
e.g. $\phi(24) = 8, \phi(7) = 6$
Note- *Notice how $\phi(p)$ where p is a prime number, will always be $p - 1$*

1.3.4 Shift Ciphers and Modulo Arithematic

We can assign numbers from 0 through 25 to all the letters in the alphabet and then use modulo arithematic for encryption and decryption-

- For a shift of k, we can encrypt any letter whose number is denotes by p to a letter $(p + k) \bmod (26)$
- To decrypt the cipher text (p'), you subtract in the ring via $(p' - k) \bmod (26)$.

1.3.5 Fast Powering algorithm

To calculate $g^A \bmod (m)$, where $g, A \in \mathbb{N}$, it can take $\sim 2^n)$ multiplications if $A \in [2^n, 2^{(n+1)})$. We can, however, use the fact that g can we written as a linear combination of powers of 2 to reduce the number of steps we take.

- Express $A = \sum_{n=0}^{k} a_r 2^r$ where $2^k \leq A < 2^{(k+1)}$
- Let $g_r = g^{2^r}$. We can find g_1 using $g \cdot g$ and for subsequent values of r, we can just use $g_{r+1} = g_r \cdot g_r$
- Now we have $g^A = \sum_{r=0}^{k} g^{a_0 + a_1 \cdot 2 + a_2 \cdot 2^2 + \cdots} = \sum_{r=0}^{k} g_0^{a_0} + g_1^{a_1} + \cdots$
- $x^2 \bmod (N) = (x \bmod (N))^2 \bmod (N)$

Keep in mind that a_r can only take the values 0 and 1 thus we can say that this method takes $2r$ multiplication steps at most. (Including the exponential expansion of A).

Prime numbers and Finite Fields

1.4.1 Division modulo m

We can usually do multiplication, subtraction, addiiton modulo m but division modulo m is only possible if the divisior(k) is coprime with m.

Divison modulo m means- If $a = p \bmod (m)$ and $b = q \bmod (m)$,

$$c = \tfrac{a}{b} = \tfrac{p'}{q'}$$

where $p \bmod (m) = p' \bmod (m)$ and $q \bmod (m) = q' \bmod (m)$ and $\frac{p'}{q'} \in \mathbb{Z}$. It is only possible if $\gcd(b, m) = 1$ as otherwise you cannot get a unique value.
Example- $a = 11 \bmod (5)$, $b = 4 \bmod (5)$ then $\frac{a}{b} = \frac{11 \bmod (5)}{4 \bmod (5)} = \frac{16 \bmod (5)}{4 \bmod (5)}$=4
Note- In the above example, 4 is the only integral value possible of $\frac{a}{b}$. Values such as $\frac{11 \bmod (5)}{4 \bmod (5)} = \frac{1}{4}$ are ignored.

The reason for $\gcd(4, 5) = 1$ being necessary in the above examples is that we can write $\frac{11 \bmod (5)}{4 \bmod (5)} = 11 \cdot 4^{-1} \bmod (5)$ and 4^{-1} can only exist if $\gcd(4, 5) = 1$. (Here $4^{-1} = 4$ thus our answer $= 44 \bmod (5) = 4$)

1.4.2 Division in rings

As seen earlier, if $\gcd(a, m) = 1$, we can easily divide modulo m by a any integer. If we now studied rings of form $\mathbb{Z}/p\mathbb{Z}$, where $p \in prime$ we will get that any integer $b \in \mathbb{Z}/p\mathbb{Z}$ can be used in division as obviously $b < p$ AND $p \in prime$ thus $\gcd(b, p) = 1$

Note- If $a \in *\mathbb{Z}/m\mathbb{Z}$, a can be used as a divisor modulo m. ALSO $\mathbb{Z}/p\mathbb{Z}= *\mathbb{Z}/p\mathbb{Z}$.

We can calculate a^{-1} using- $au + pv = 1$ (as $\gcd(a, p) = 1$), wherein solving for u yields us $a^{-1} \bmod (p)$.

1.4.3 Finite Fields

Since we can compute division, addition, multiplication and subtraction in $\mathbb{Z}/p\mathbb{Z}$, we can call it a field just like $(R)/\mathbb{Z}/$ etc$\cdots$
Since $\mathbb{Z}/p\mathbb{Z}$ only has a finite number of elements, we call it a finite field and can even denote it by $(F)_p$ to show the finitedness.

Fermat's "Little" Theorem

The theorem states that-

$$a^{p-1} \equiv 1 \bmod (p) \text{ for } a \neq k \cdot p$$

Proof:
Since $a \neq k \cdot p$, $a, 2a, 3a, 4a, \cdots (p-1)a \bmod (p) \neq 0$. Now if we considered $k \in [1, p-1]$,

$$k_1 \cdot a \equiv j_1 \bmod (p) \text{ AND } k_2 \cdot a \equiv j_2 \bmod (p)$$

$$\Rightarrow (k_1 - k_2) \cdot a \equiv (j_1 - j_2) \bmod (p)$$

Since $k_1 - k_2$ cannot be a multiple of p, $j_1 - j_2 \neq 0$ and so for every $k \in [1, p-1]$, we get a distinct $j \in [1, p-1]$. Hence we get a bijective function of $[1, p-1] \rightarrow [1, p-1]$. So-

$$\prod_{k=1}^{p-1} k \cdot a \equiv \prod_{k=1}^{p-1} k \bmod (p) \Rightarrow (p-1)! a^{p-1} \equiv (p-1)! \bmod (p)$$

$$\Rightarrow a^{p-1} = 1 \bmod (p)$$

We can use this to find the inverse of any number modulo p-

$$a \cdot a^{p-2} \equiv 1 \bmod (p) \Rightarrow a^{-1} \equiv a^{p-2} \bmod (p)$$

Voila!

We just found an inutitive, new way to check if a number is composite or not

If we wanted to check if a number n was prime or not-

- If $n \in even$, obviously composite
- If $n \in odd$, take $a = 2$. Now if $2^{n-1} \not\equiv 1 \bmod (n)$, we can be sure that it is definitely composite. [2]

Definition. We define *order* as the smallest number k such that

$$a^k \equiv 1 \bmod (p)$$

[2]As is with all good things in life, this too has an asterisk- you cannot judge if a number is prime or not solely based on this. There is a subset of composite numbers called the "Carmicheal Numbers" which satisfy $b^{(n-1)} \equiv 1 \bmod (n)$ despite their lack of primality. Just random gifts of joy Mathematics bestows upon you.

We can also be sure that $p-1$ is divisible by k.

1.5.1 Primitive Roots Theorem

The theorem states that *for a prime number p, there always exists a number $g \in \mathbb{F}_p$, such that every element of $\mathbb{F}_p^*$ can be expressed as a power of g*

Number of primitive roots of $p = \phi(p-1)$

Symmetric Ciphers

1.6.1 Notations

If you have a message $m \in \mathcal{M}$ (where $\mathcal{M}$ is the list of all possible messages), you can choose a key $k \in \mathcal{K}$ (again, where $\mathcal{K}$ is the space of all keys), to get a cipher $c \in \mathcal{C}$. (At this point if you cannot guess what $\mathcal{C}$ stands for, I don't think you should read further).

We thus need invertible functions $e_k(m)$ and $d_k(m)$ as -

$$e_k(m_i) = c_i \text{ AND } d_k(e_k(m)) = m$$

1.6.2 Principles of Succesful Ciphers

Kerckhoff's principle says that the security of a cryptosystem should depend only on the secrecy of the key, and not on the secrecy of the encryption algorithm itself. So keeping that in mind, we design a few principles-

1. For any plaintext m and key k, it must be easy to calculate $e_k(m)$.
2. For any ciphertext c and key k, it must be easy to calculate $d_k(m)$.
3. If you have ciphertexts $c_1, c_2, c_3 \cdots c_\alpha$ encrypted using the key k, it should be really difficult to find $d_k(c_1), d_k(c_2), \cdots d_k(c_\alpha)$ without knowing k.
4. If you have $c_1, c_2, c_3, \cdots c_\beta$ and the corresponding messages $m_1, m_2, m_3, \ldots m_\beta$, it should still be difficult to decipher a c not in this set.

1.6.3 Encoding of Cipher Blocks

We can represent any plaintext in the form of bits $m_{B-1}m_{B-2}\cdots m_2m_1m_0$ where $m_i \in 0, 1 \forall i \in [0, B)$. Then we can break it into bytes or blocks of some other size and encode each block separately. We prefer to choose a $\mathcal{K}$ such that $k \in [0, 2^{B_k})$ and $B_k \geq 80$ bits or 160 bits based on the message type. (This number is chosen based on the computing capacity of a modern computer since you can brute-force check each and every key $k \in \mathcal{K}$, if B_k is not large enough.)

1.6.4 Examples of Symmetric Ciphers

Choose $\mathcal{K} = \mathcal{C} = \mathcal{M} = \mathbb{F}_p^*$ for some prime number p and then you can define-

$$e_k(m) \equiv k \cdot m \bmod (p)$$

Using that you can also find k' (in $2\log_2(p) + 2$ steps) such that-

$$d_k(c) \equiv k' \cdot c \bmod (p) \text{ to yield us } d_k(c) \equiv m$$

Although this method follows the first 3 principles, it fails the fourth principle of successful ciphering as-

$$k \equiv m^{-1} \cdot c \bmod (p)$$

The above formula gives you the exact key as soon as you find one pair of ciphertext and plaintext thus is not preferable at all.

Simple multiplicative cipher

If instead of using a modulo p, if you just used $e_k(m) = k \cdot m$, you can easily find $\gcd(c_1, c_2, c_3 \cdots c_\alpha)$ which can give you the value of k outright. (Calculating GCD of multiple numbers is not labour intensive for a computer)

Affine Cipher

Here we define ciphertext using both multiplication and addition modulo p-

$$e_k \equiv k_1 \cdot m + k_2 \bmod (p)$$

AND

$$d_k(c) \equiv k_1' \cdot (c - k_2) \bmod (p)$$

Note- Caesarean Shift Cipher is just a special case of this cipher with $k_1 = 1$

Hill Cipher

In this cipher, we first denote every message of length n as a $n \times 1$ column matrix. Then we can define a key $k = k_{1,1}k_{1,2}k_{1,3}\cdots k_{n,n}$ as-

$$k = \begin{bmatrix} k_{1,1} & k_{1,2} & k_{1,3} & \dots & k_{1,n} \\ k_{2,1} & k_{2,2} & k_{2,3} & \dots & k_{1,n} \\ k_{3,1} & k_{3,2} & k_{3,3} & \dots & k_{1,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ k_{n,1} & k_{n,2} & k_{n,3} & \dots & k_{n,n} \end{bmatrix}$$

We can calculate the inverse of $k = k^{-1}$ and using the both of them, our cipher looks like-

$$\begin{bmatrix} k_{1,1} & k_{1,2} & k_{1,3} & \dots & k_{1,n} \\ k_{2,1} & k_{2,2} & k_{2,3} & \dots & k_{1,n} \\ k_{3,1} & k_{3,2} & k_{3,3} & \dots & k_{1,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ k_{n,1} & k_{n,2} & k_{n,3} & \dots & k_{n,n} \end{bmatrix} . \begin{bmatrix} m_{1,1} \\ m_{2,1} \\ m_{3,1} \\ \vdots \\ m_{n,1} \end{bmatrix} = \begin{bmatrix} c_{1,1} \\ c_{2,1} \\ c_{3,1} \\ \vdots \\ c_{n,1} \end{bmatrix}$$

Unfortunately both Affine and Hill Ciphers are both susceptible to plain text attacks. Hence we cannot use either one of these despite how dapper they look.

XOR Cipher

XOR $\oplus$ of key and message $c \equiv m \oplus k$ seems like a good option but it has the following flaw if you have access to 2 ciphers made using the same key-

$$c' \oplus c \equiv (m' \oplus k) \oplus (m \oplus k) \equiv m' \oplus m$$

$m \oplus m'$	Possible m	Possible m'
0	0	0
0	1	1
1	0	1
1	1	0

Thus we can easily find information about m from just 2 ciphers. (Although still m exactly isn't visible)

1.6.5 Random bit sequences

To circumvent the above issue, we can use different "keys" for different messages that we encrypt. One way to do that is to use a key k and a pseudo-random[3] number generator $\mathcal{R}$ such that-

$$\mathcal{R} : \mathcal{K} \times \mathbb{Z} \to 0, 1$$

Thus we can use this to construct a sequence of n-bits (the XOR key) $j_1 j_2 j_3 \cdots j_n$ such that $j_i = \mathcal{R}(k, i)$ if message to be encrypted has n bits as well. Then we can just sequentially go through every bit of the message and apply $\oplus$ with bits of the XOR key.

Asymmetric Ciphers

One flaw with symmetric ciphers is that the key used for encryption is same as the one used for decryption. Hence for *Alice* to send a message to *Bob*, she has to first send him a copy of the key she used for encryption. However, in the information age, that key is equally susceptible to being intercepted by *Eve* hence undermining the privacy of *Alice* and *Bob*.

[3]Not called "random" as we are using some algorithm to generate it after all.

One effective way to combat this is having a public key used for encryption (k_{public}) and a private key for decryption ($k_{private}$). As the names suggest, the former is available freely and can be used by anyone while sending information to *Alice* but only *Alice* has the latter key.
Some important details of this method-

- $e_{k_{public}}(m_i) = c_i$ and $d_{k_{private}}(c_i) = d_{k_{private}}(e_{k_{public}}(m_i)) = m_i$.
- It should be infeasible to calculate $k_{private}$ using just knowledge of k_{public}.

Week 2

Probability Theory

2.1.1 Basics

Well the basics are luckily... pretty basic. All of the stuff we learnt already in JEE. Some of the

2.1.2 Monty Hall Problem

Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?

One important thing to keep in mind which makes the final solution seem non-intuitive $\rightarrow$ The host knows which door has the Car behind it. He will never open the door with the car. Let us try to rationalise the problem based on Bayes' Theorem.
Assume for now that whatever door we choose is called $Door_1$, the host opens either $Door_2$ or $Door_3$

Case 1: $Car \rightarrow Door_1$
In that case $P(keep \rightarrow Car) = 1$ and $P(swap \rightarrow Car) = 0$

Case 2: $Car \rightarrow Door_2$
In that case $P(keep \rightarrow Car) = 0$ and $P(swap \rightarrow Car) = 1$

Case 3: $Car \rightarrow Door_3$
In that case $P(keep \rightarrow Car) = 0$ and $P(swap \rightarrow Car) = 1$

Using Total probability theorem, it is obvious that $P(keep \rightarrow Car) = \frac{1}{3}$ and $P(swap \rightarrow Car) = \frac{2}{3}$

2.1.3 Monte-Carlo Algorithms

Information Theory

This section is yet incomplete.

Work in progress.

Week 3

Random Numbers

3.1.1 Why do we need random numbers

A lot of the encryption techniques used in the later weeks will require some randomness to

Random Bit Sequences but in specific-

- RSA cryptosystems, despite being completely deterministic, requires some randomness in the plaintext for better security.
- Public key cryptosystems depend almost completely on the using random prime numbers to design the keys.
- Probabilistic Encryption systems definitely require incorporation of randomness into the encryption process.

3.1.2 Is it even possible to construct an actually random number?

Quantum Theory states that the decay of specific atom is completely probabilistic. We can be sure that for a radioactive atom, there is a T time such that after T seconds, a undecayed atom has a *50%* chance of decaying but we cannot say with 100% surety if a particular atom will decay or not.

Thus to construct the first bit of a random number, we just study a radioactive atom for T seconds and if it decays, $bit = 1$ else $bit = 0$.

Unfortunately, this is a very impractical way to construct a large random string daily use. Hence we go for pseudo random bit generators.

3.1.3 Pseudo-Random Number Generator

PRNG is just a function that takes in an initial seed S which is truly random to algorithmically create a pseudo-random number[4].

[4]Called so because of the irony of having a well defined function that can create supposedly random numbers.

The two properties that a PRNG should have for it to be considered cryptographically secure are-

1. If *Eve* has the first k bits of the PRN, she should not be able to predict the next bit with a more than 50% chance.
2. If *Eve* has some the bits in the PRN such as $R_t, R_{t+1}, R_{t+2} \cdots$, she should not be able to predict any of $R_1, R_2, \cdots, R_{T-1}$ bits with a more than 50% surety.

To continue further, some knowledge of Hash Functions is necessary.

Hash Functions

Hash Algorithmns are like a summary of the original document. It is difficult to figure out the original document from the hash output but it acts as a signature that ensures accuracy. Due to the pigeonhole problem, it is obvious that Hash collisions would occur- but a good algorithm ensures that those are incredibly difficult to happen by happenstance and equally so to artificially bring about
- Tom Scott

Hash functions take in arbitrarily long documents D and output a shorter bit strong H. Their main objective is for authentication purposes. Some important characteristics are-

- Computation of $Hash(D)$ should be fast $\rightarrow$ *linear time*
- Inversion of Hash(D) should be difficult $\rightarrow$ *exponential time* i.e. finding **any** D given a hash output H of the form $Hash(D) = H$
- Ideally we want Hash functions to have collision resistance i.e. it should be difficult for 2 non-identical documents $D1$ and $D2$ to give the $Hash(D1) = Hash(D2)$. This property is important for the sake of unambiguity in differentiating between 2 possible inputs when the inteded recepient receives the Hash output.

Note-

a. There is a considerable difference between Authentication/ Verification and Inversion. Hash functions need to be easy to compute as well as verify but should be resistant from inversion.

b. Hash is not an encryption- it cannot (ideally) be decrypted back to the original message. It is just a signature. c. Hash outputs are “seemingly” random so even a small flipping of bits in the input should completely change the hash.

We ideally would base Hash functions on mathematically difficult problems to solve such as the Discrete Logaritms and Prime Factoring to ensure that inversion is difficult but that is impractical for even the encryption speeds today's world needs. So we prefer ad-hoc mixing methods such as the *Secure Hash Algorithm* SHA (or SHA-1) which is the universally used algorithm.

3.2.1 How does SHA-1 work?

SHA outputs 160 bits no matter what you input in; if the input is a multiple of 160 bits, we just xor the first result of the kth output with the transformation of the $(k+1)$th mixing to get the $k+1$th output. If the file isn't a multiple or is smaller than 160 bits, we just append 0s till we get a proper multiple.

1. Choose 5 32-bit numbers $h_0, h_1, h_2, \dots, h_4$ randomly.
2. Divide the file into chunks of 512 bits and then redivide those chunks individually into 16 blocks of 32-bits (or words) $w_0, w_1, w_2 \dots w_1 5$.
3. "Rotate" the words (i.e. shift the bits circularly arbitrarily **between** the words[a] (not inside the words individually)) to create 80 words $w_0, w_1, w_2, \dots, w_7 9$.

4. Start the loop with i as control variable incrementing from 0 to 79.
5. Now just create temporarily variables $a = h_0$, $b = h_1$, $\ldots e = h_4$ and construct f from some arbitrary combination of a, b, c, d, e using AND and XOR.
6. Arbitrarily mix a, b, c, d, e using some rotations, shifts and whatever you feel like adding on a lazy Wednesday afternoon.
7. Add f and w_i to a and then equate $h_0 = h_0 + a, h_1 = h_1 + b, \ldots, h_4 = h_4 + e$.
8. End the loop once you reach i=79. (i.e. complete 80 rounds)
9. Do the same for all chunks just making sure that the values of $h_0, h_1 \ldots h_4$ are carried forward after each chunk.
10. Concatenate $h_0, h_1 \ldots h_4$ to get the final 160-bit hash.

[a] Essentially if you have 1000, 1101, 0111, 1010 and you decided to rotate the 3rd bit in each 4-bit number, you can get a 1000, 1111, 0111, 1000 i.e. the 3rd bit of each number is extracted to form the array 0,0,1,1 and this is rotated as the algorithm is designed

Thus now to create a pseudo-random number R_i, we just select a specific $Hash$ function and input into it S (A truly random number) concatenated with i to get-

$$Hash(i \mid\mid S) = R_i$$

Public Standard for making PRNGs

Start of with a random seed S and a key k for the cryptosystem. Let E_k be the encryption function associated with that k.
Every time a random number is required, create a D from some CPU parameters (temperauture and/or date and time and/or cpu usage) to get-

$$C = E_k(D)$$

Thus your random number R will be-

$$R = E_k(C \text{ xor } S)$$

You can then update S to -

$$S = E_k(R \text{ xor } C)$$

Stream Ciphers

This is an encryption technique which heavily depends on the PRNGs we defined above.

If you wanted to encrypt a 10-bit plaintext using Stream Ciphers, you will require a 10-bit keystream as well. If k_i is the ith bit of the keystream and m_i is the ith bit of the plaintext,

$$c_i \equiv k_i \oplus m_i$$

This keystream should be as random as possible to prevent codebreakers from predicting properties about the keystream thus getting information about the plaintext.

But it is infeasible to make n-bit keystreams for large values of n hence we use PRNGs. We can thus have a k bit purely random seed S with which we can create $R_1, R_2, R_3 \ldots, R_n$ using the above given algorithm. (Initial seed S is updated after every use as was defined in the last section to strengthen the security of the system)

BTW the keystream used with Stream Ciphers is known as *One Time Pad*. A n-bit ciphertext made from a truly random n-bit one time pad is mathematically impossible to decipher.

3.4.1 Deciphering Stream Ciphers

xor ($\oplus$) has an interesting property $\rightarrow$

A	B	$A \oplus B$	$(A \oplus B) \oplus B$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

Hence we can say with surety that-

$$k_i \oplus c_i \equiv k_i \oplus (k_i \oplus m_i) \equiv m_i$$

Giving us a clear cut way to use this private key symmetric encryption system.

ALSO since the generation of R_n does not require any subsequent $R_{k>n}$, we can process each bit as it the CPU reads it instead of having to store it

Block Cipher

[5] Unlike Stream Cipher, Block Cipher is a lot simpler and less secure. It divides the input into chunks of bits and evaluates them separately.

[5]A lot of these concepts and pictures are sourced from “https://www.hypr.com/black-cipher/”

3.5.1 Electronic Code Book Method

Essentially if you have a k bit codebook and a n bit plain text, you divide the plain text into chunks of k bits (some padding scheme is employed in case $n \% k \neq 0$). Then you go through each chunk and find the corresponding ciphertext in the ECB (Electronic Code Book). Thus you encrypt the entire file.

Plaintext	*Ciphertext*
00	11
01	01
10	00
11	10

2-bit Electronic Code Book

Some minor itty bitty issues- It preserves large scale structures hence it is completely useless to the most degree in obfuscating files with a large size if there is a lot of repetition.

Imagine a 10×10 picture where the brightness of each pixel is defined by a 2-bit digit

Plaintext	*Ciphertext*
0000	1001
0001	1000
0010	0100
0011	0101
0100	0011
0101	0111
0110	1101
0111	1111
1000	0110
1001	1011
1010	0001
1011	1100
1100	0010
1101	1100
1110	1010
1111	0000

$$\begin{bmatrix} 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 01 & 01 & 01 & 01 & 01 & 01 & 00 & 00 \\ 00 & 00 & 01 & 10 & 10 & 10 & 10 & 01 & 00 & 00 \\ 00 & 00 & 01 & 10 & 11 & 11 & 10 & 01 & 00 & 00 \\ 00 & 00 & 01 & 10 & 11 & 11 & 10 & 01 & 00 & 00 \\ 00 & 00 & 01 & 10 & 10 & 10 & 10 & 01 & 00 & 00 \\ 00 & 00 & 01 & 01 & 01 & 01 & 01 & 01 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \end{bmatrix} \xrightarrow{\text{ECB}} \begin{bmatrix} 10 & 01 & 10 & 01 & 10 & 01 & 10 & 01 & 10 & 01 \\ 10 & 01 & 10 & 01 & 10 & 01 & 10 & 01 & 10 & 01 \\ 10 & 01 & 01 & 11 & 01 & 11 & 01 & 11 & 10 & 01 \\ 10 & 01 & 11 & 01 & 00 & 01 & 10 & 11 & 10 & 01 \\ 10 & 01 & 11 & 01 & 00 & 00 & 10 & 11 & 10 & 01 \\ 10 & 01 & 11 & 01 & 00 & 00 & 10 & 11 & 10 & 01 \\ 10 & 01 & 11 & 01 & 00 & 01 & 10 & 11 & 10 & 01 \\ 10 & 01 & 01 & 11 & 01 & 11 & 01 & 11 & 10 & 01 \\ 10 & 01 & 10 & 01 & 10 & 01 & 10 & 01 & 10 & 01 \\ 10 & 01 & 10 & 01 & 10 & 01 & 10 & 01 & 10 & 01 \end{bmatrix}$$

In the above illustration, despite the codebook using more bits than the individual pixels in the image, it still cannot obfuscate efficiently the initial file hence giving a lot of information away like-

1. The length of the codebook string.
2. Presence of repeated structures such as the brightspot in the middle and the darkspace in the periphery.
3. General lack of self respect for having used such a unsophisticated method.

We can clearly see the issue here- inability of the encryption system to convert the same number to 2 different numbers if they occur at 2 different places. Hence we use *Cipher Block Chaining (CBC).*

3.5.2 Cipher Block Chaining

In this encryption method, the outcome of the previous encryption decides the outcome of the next encryption. Just like ECB mode, we do have a codebook we refer to for each chunk.

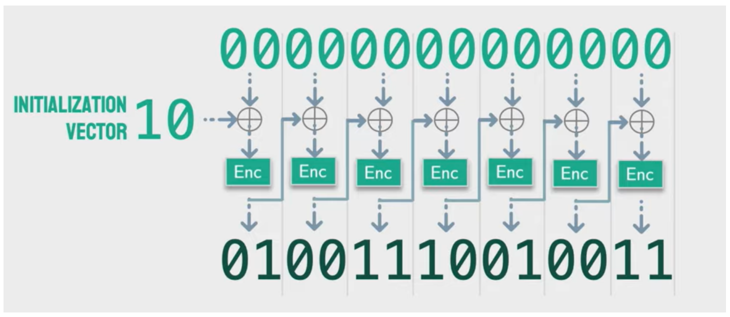

Figure 3.1: We can clearly see that the first IV (intialisation vector) is xor-ed with the first chunk. The cipher text of the first chunk is then used similarly. Notice how repeated structures are not visible anymore.

Now another strength of this method is the factor that the same codebook and plaintext can give a wildly different cipher if you just changed the initial seed (or initialisation vector).

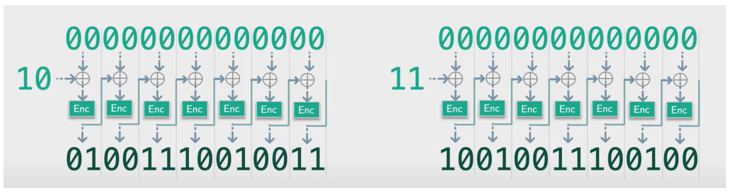

Figure 3.2: This is so nice to look at. Relish the randomness

An interesting point- The I.V. (or initial seed) is not at all a secret. It is in fact given along with the ecrypted message as it is necessary for decryption. Another property - it should not be predictable or used more than once (to ensure resistance from plaintext attacks)

3.5.2.1 How to decrypt CBC

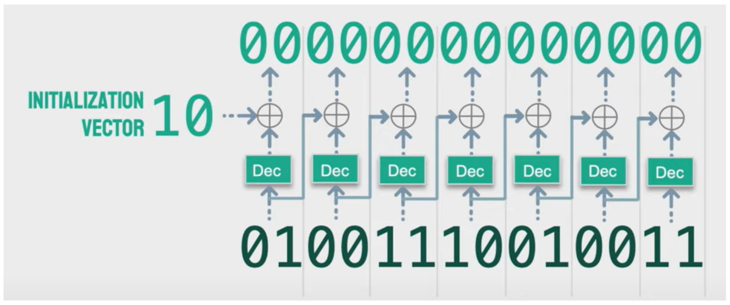

Figure 3.3: Decryption exploits the reversibility of $\oplus$

3.6 Counter Mode (CTR)

Another method of Block Ciphering is Counter Mode wherein you are actually encrypting (i.e. using the codebook) on the counter and not the plaintext. The plain text is merely xor-ed with the encryption of the counter. Initial seed decides the starting value of the counter which is incremented.

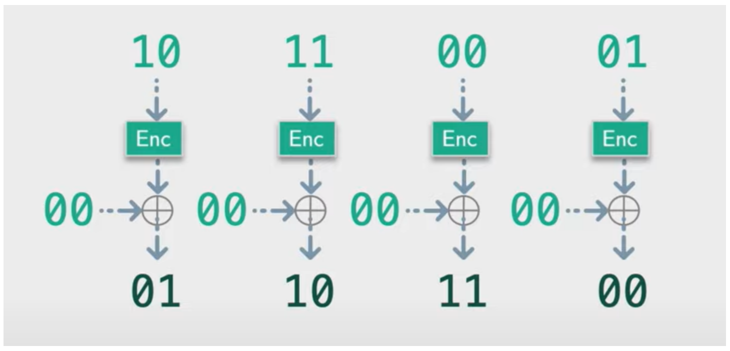

Figure 3.4: A very logical and simple encryption process. The counter is on the top and the plaintext is xor-ed in the middle

To put it algebraically, if we called the k-bit counter $C(i)$, and the code book as $ECB : \{0,1\}^k \rightarrow \{0,1\}^k$ and the ith k-bit chunk of plaintext as m_i then-

$$\boxed{cipher_i = m_i \oplus ECB(C(i))}$$

3.6.1 Decryption

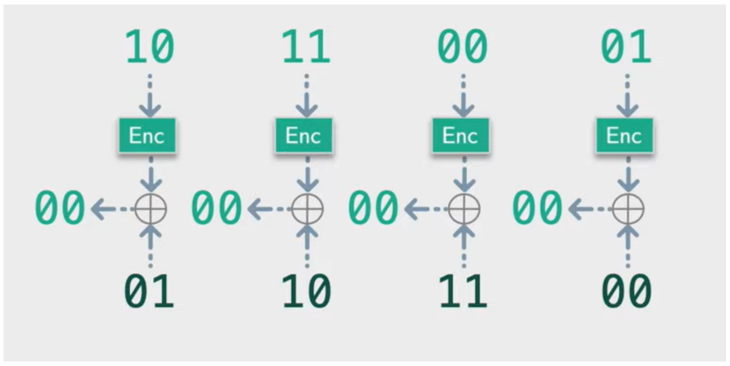

Figure 3.5: It is very similar in the sense that we interchange one

In algebraic terms, we can call it-

$$\boxed{m_i = cipher_i \oplus ECB(C(i))}$$

Advantages over CBC	Disadvantage over CBC
Simpler implementation Resistant to attacks terms as “padding oracle”	Not safe for small block lengths (¡128 bit)

Week 4

Introduction to Diffie Hellman

Finally now we will study a public key cryptosystem in detail. Diffie and Hellman were some of the earliest cryptographers working to create a one-way trapdoor function - easy to compute, difficult to compute unless you have some trapdoor information. These functions are of a particular importance. As was explained in Asymmetric Ciphers section, we make sure of $k_{public}, e_k()$ for encryption which is made public knowledge whereas $k_{private}, d_k()$ are kept private as they are used for decryption. Since k_{public} is constructed from $k_{private}$, it is important that the function that decides their relation is unidirectional else finding $k_{private}$

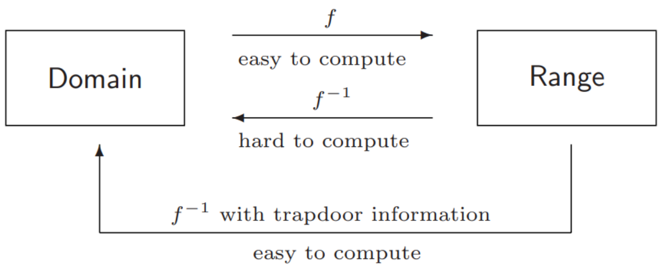

Figure 4.6: Schematic of a trapdoor function

will be possible for *Eve* from just the information that is publically available.

But another thing to keep in mind is that if *Bob* is receiving the message from *Alice*, he should be able to decrypt it since he has that extra information.

Although ultimately Diffie and Hellman failed to create a public key cryptosystem (PKC), they did manage to create a system that can securely transfer certain data by utilising the Discrete Logarithm problem.

Discrete Logarithm

Refer to Prime numbers and Finite Fields for the background information in this section. According to the Primitive Roots Theorem, in every $\mathbb{F}_p$, there exists a g such that $1, g^1, g^2, \ldots, g^{p-1}$ contains all the elements present in $\mathbb{F}_p*$ and $g^{p-1} \equiv 1 \bmod (p)$ (Fermat's Little Theorem).

Discrete Logarithm Problem (DLP) is the problem of finding x such that-

$$g^x \equiv h \bmod (p)$$

for some arbitrary h and x is called the *discrete logarithm of h to the base $g = \log_g h$.*

<u>Note-</u> We use the term "index" to sometimes refer to the discrete logarithm i.e. $x = \text{ind}_g(h)$. This is to ensure that the usual log and discrete log are not confused for each other.
$x' = x + (p-1)k$ would also satisfy our condition thus giving up ∞ly many solutions x. Hence x is only defined modulo (p-1). Hence we say that

$$\text{ind}_g : \mathbb{F}_p* \to (\mathbb{Z}/(p-1)\mathbb{Z})$$

Why is it called logarithm?
Well you can easily see that

$$\text{ind}_g(ab) = \text{ind}_g(a) + \text{ind}_g(b)$$

and that gives us enough of a reason to call it so.

58

This section is yet incomplete.
Work in progress.

Some content

text

Summary of project completion

As you can see in the previous few sections, Week 2 and Week 4 are incomplete with only a few sections filled- essentially a lag of 2 weeks. However, after the end semester exams, I am sure I can pick up the slack and work for the following modified plan of action. End-semester exams end for me at the end of June, 2022 so I will start most of the heavy work in July. I am also assuming a deadline extension till 22nd July, 2022 as was informally hinted at for fresher students.

5.1 Modified PoA

Week 6 (19th June- 25th June)	Diffie Hellman (Discrete Logarithms)
Week 7 (26th June- 2nd July)	Endsem week
Week 8 (3rd July- 9th July)	RSA (Factorisation), Data Encryption Standard (DES), Advanced Encryption Standard (AES)
Week 9 (10th July- 16th July)	Hash functions, Authentication problem, Digital signatures
Final Project submission[6] (17th July- 23rd July)	Elliptic curves

Bibliography

[1] Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman, *An Introduction to Mathematical Cryptography*

[2] Hypr.com (https://www.hypr.com/black-cipher/)

[2] Joshua Holden, *THE MATHEMATICS OF SECRETS*

[3] Jonathan Katz and Yehuda Lindell, *Introduction to Modern Cryptography*

[4] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, *HANDBOOK of APPLIED CRYPTOGRAPHY*

[5] Simon Singh, *"The Code Book- HOW TO MAKE IT, BREAK IT, HACK IT, CRACK IT"*