UNIT II:

Advanced Encryption Standard (AES), Introduction to Public Key Cryptosystem, Discrete Logarithmic Problem, Diffie-Hellman Key Exchange Computational & Decisional Diffie-Hellman Problem, RSA Assumptions & Cryptosystem,RSA Signatures & Schnorr Identification Schemes, Primarily Testing,Elliptic Curve over the Reals, Elliptic curve Modulo a Prime., Chinese Remainder Theorem.

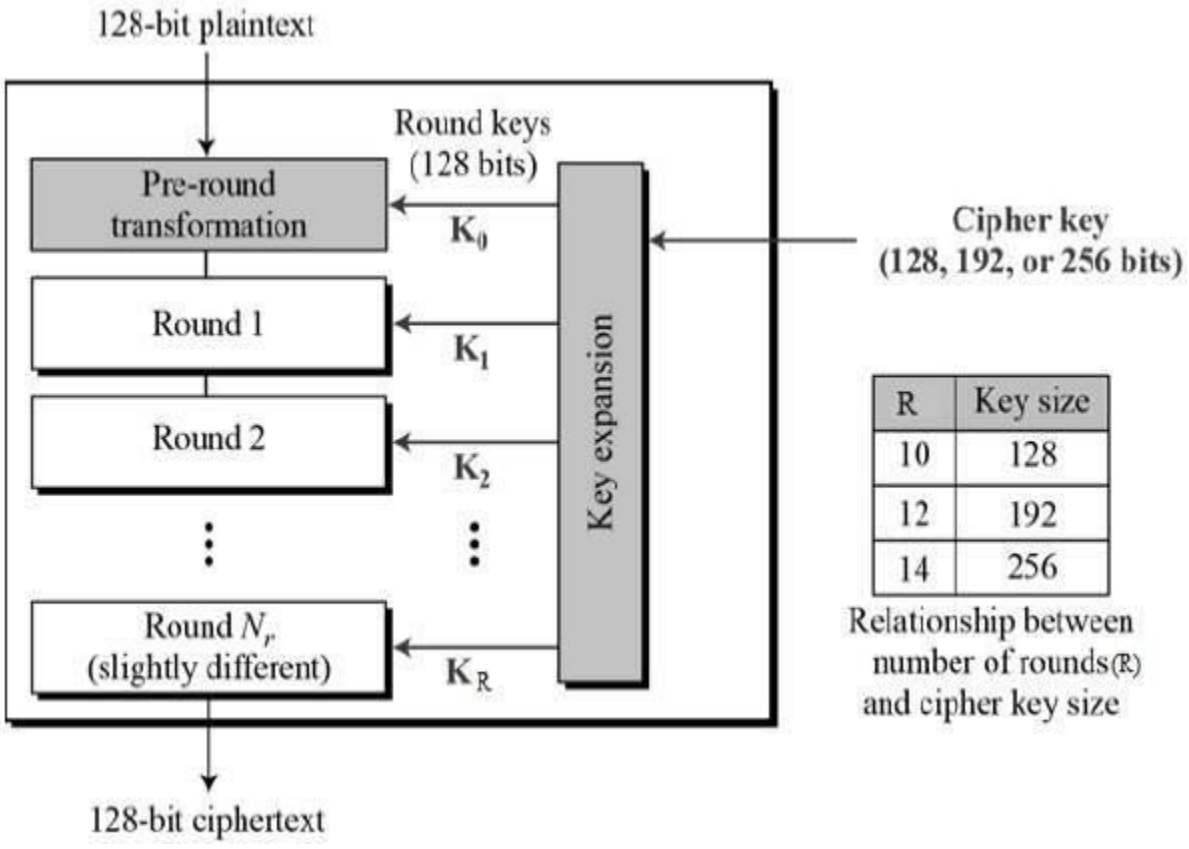
Operation of AES

AES is an iterative rather than Feistel cipher. It is based on 'substitution–permutation network'. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix −

Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

The schematic of AES structure is given in the following illustration −

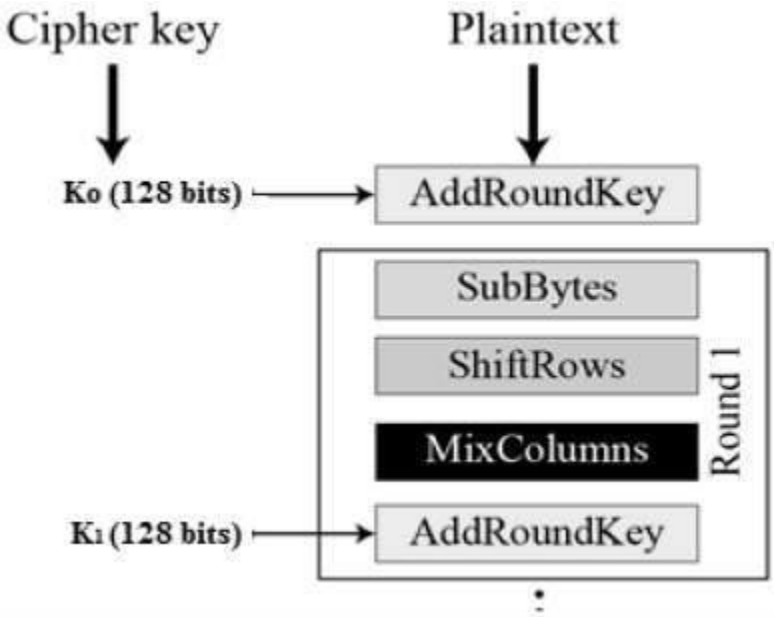


R	Key size
10	128
12	192
14	256

Relationship between number of rounds(R) and cipher key size

Encryption Process

Here, we restrict to description of a typical round of AES encryption. Each round comprise of four sub-processes. The first round process is depicted below −



Byte Substitution (SubBytes)

The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.

Circular Shift rows

Each of the four rows of the matrix is shifted to the left. Any entries that 'fall off' are re-inserted on the right side of row. Shift is carried out as follows −

- First row is not shifted.
- Second row is shifted one (byte) position to the left.

87	F2	AD	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

87	F2	AD	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

- Third row is shifted two positions to the left.
- Fourth row is shifted three positions to the left.
- The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

MixColumns

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

87	F2	AD	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

EXAMPLE

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

Addroundkey

The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

XOR

AC	19	28	57
77	FA	D1	5C
66	DC	29	00
F3	21	41	6A

=

EB	59	8B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D2

Decryption Process

The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order −

- Add round key
- Mix columns
- Shift rows
- Byte substitution

Since sub-processes in each round are in reverse manner, unlike for a Feistel Cipher, the encryption and decryption algorithms needs to be separately implemented, although they are very closely related.

AES Analysis

In present day cryptography, AES is widely adopted and supported in both hardware and software. Till date, no practical cryptanalytic attacks against AES has been discovered. Additionally, AES has built-in flexibility of key length, which allows a degree of 'future-proofing' against progress in the ability to perform exhaustive key searches.

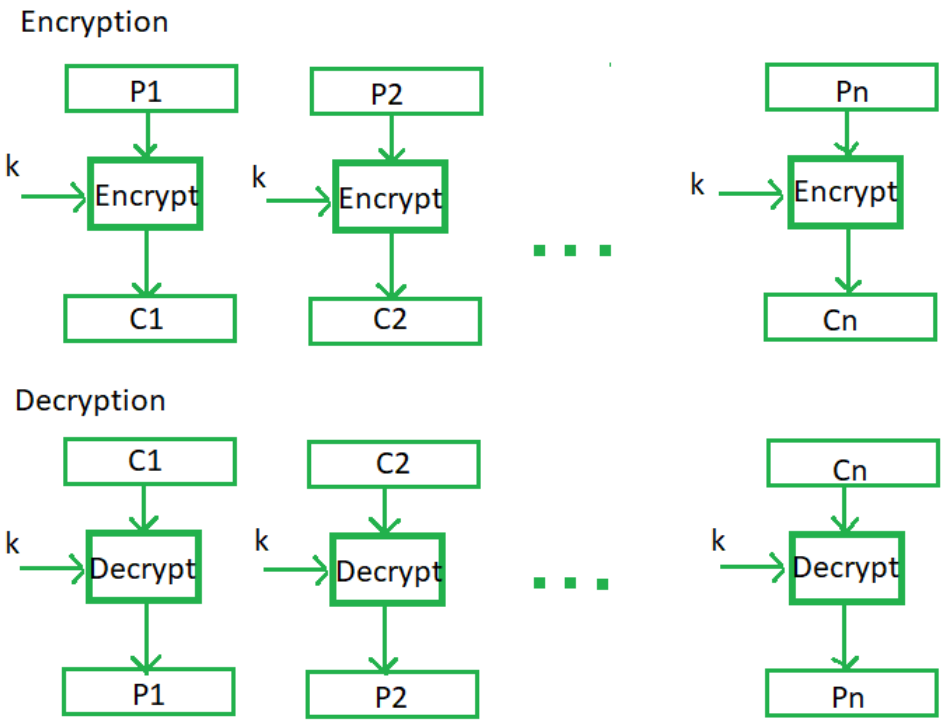
However, just as for DES, the AES security is assured only if it is correctly implemented and good key management is employed.

Block Cipher modes of Operation

Encryption algorithms are divided into two categories based on input type, as block cipher and stream cipher. **Block cipher** is an encryption algorithm which takes fixed size of input say *b* bits and produces a ciphertext of *b* bits again. If input is larger than *b* bits it can be divided further. For different applications and uses, there are several modes of operations for a block cipher.

Electronic Code Book (ECB) –

Electronic code book is the easiest block cipher mode of functioning. It is easier because of direct encryption of each block of input plaintext and output is in form of blocks of encrypted ciphertext. Generally, if a message is larger than *b* bits in size, it can be broken down into bunch of blocks and the procedure is repeated.
Procedure of ECB is illustrated below:



Advantages of using ECB –

- Parallel encryption of blocks of bits is possible, thus it is a faster way of encryption.

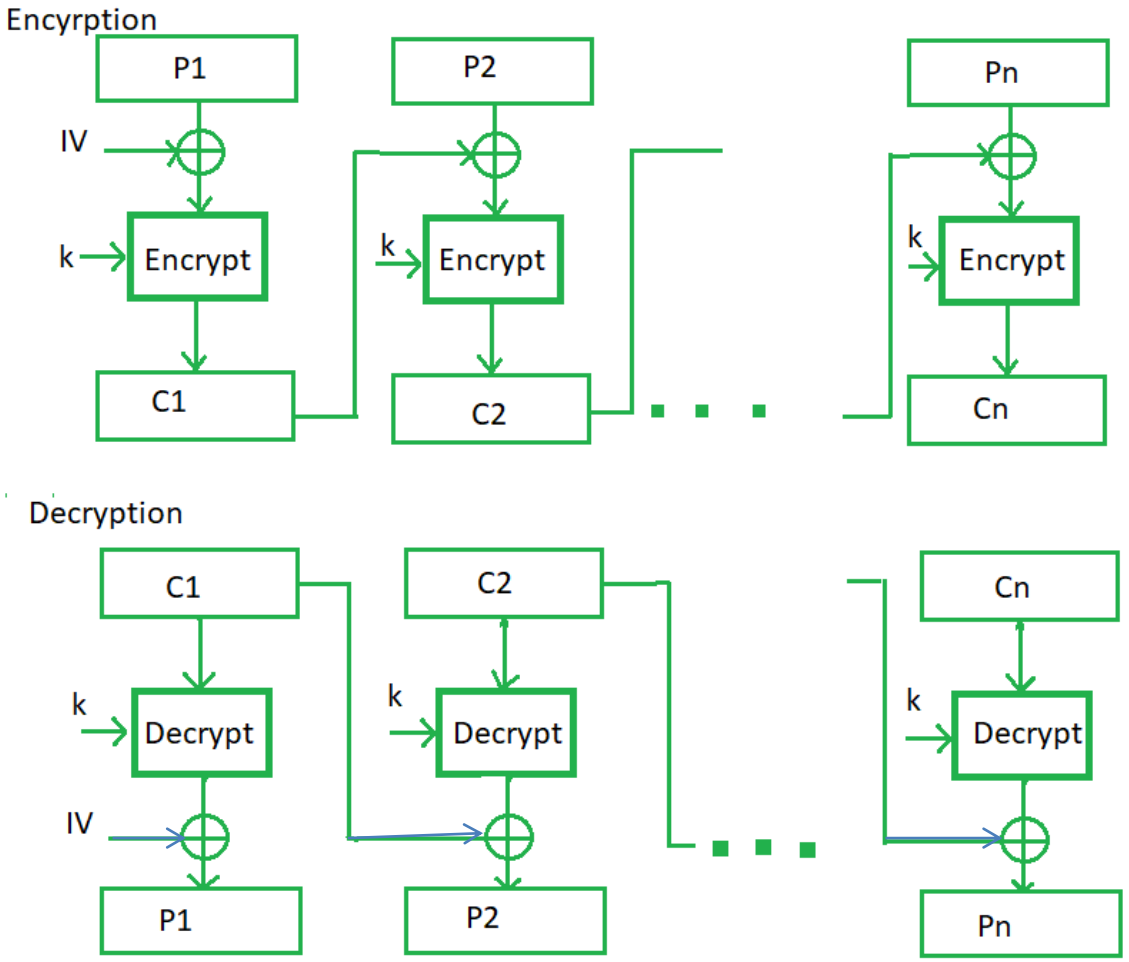
- Simple way of block cipher.
- Secure transmission for single value like password and encryption key.

Disadvantages of using ECB –

- Prone to cryptanalysis since there is a direct relationship between plaintext and ciphertext.

Cipher Block Chaining –

Cipher block chaining or CBC is an advancement made on ECB since ECB compromises some security requirements. In CBC, previous cipher block is given as input to next encryption algorithm after XOR with original plaintext block. In a nutshell here, a cipher block is produced by encrypting a XOR output of previous cipher block and present plaintext block.
The process is illustrated here:



Advantages of CBC –

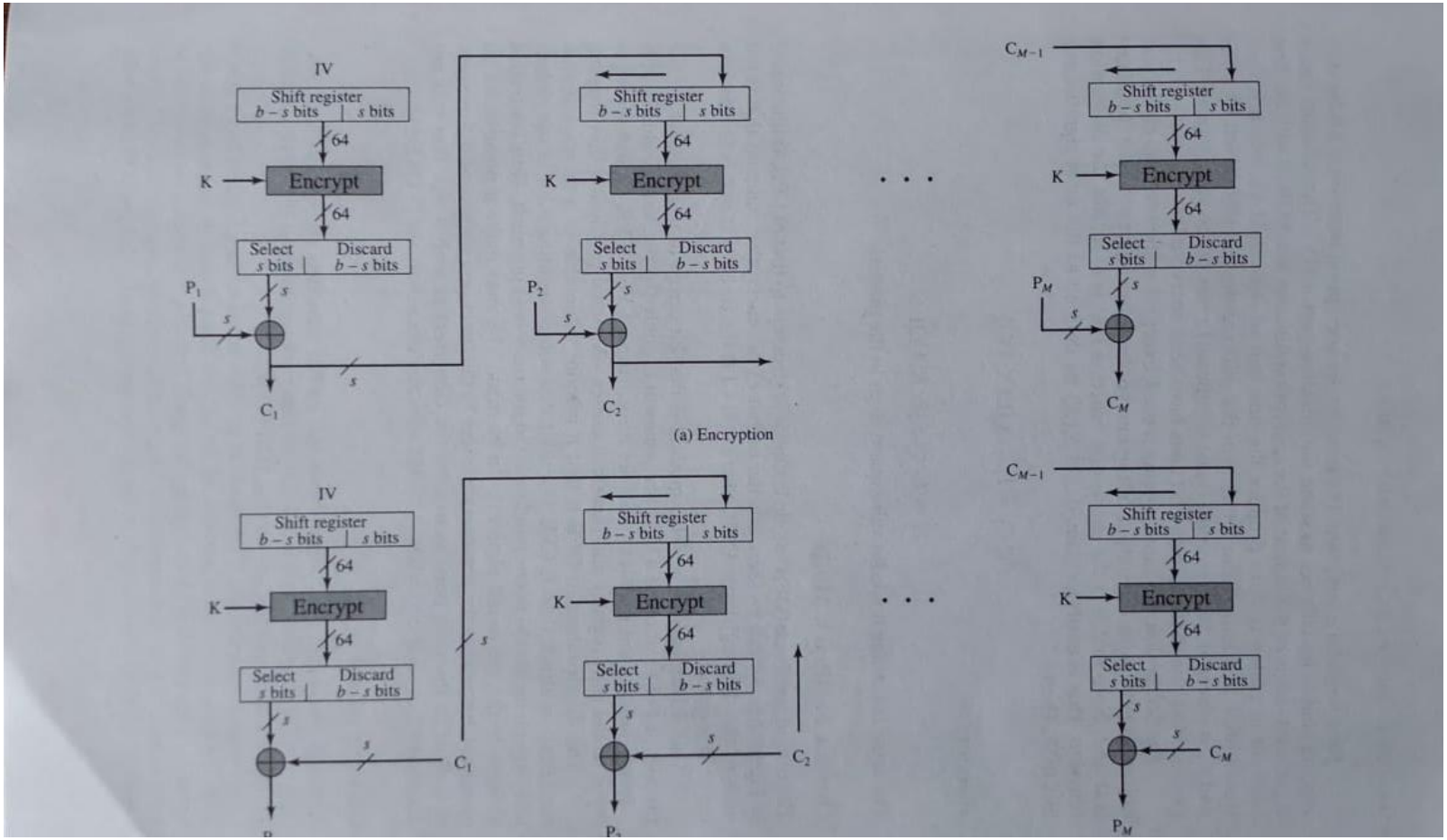
- CBC works well for input greater than *b* bits.
- CBC is a good authentication mechanism.
- Better resistive nature towards cryptanalysis than ECB.
- General purpose block oriented transmission i.e. authentication

Disadvantages of CBC –

- Parallel encryption is not possible since every encryption requires previous cipher.

Cipher Feedback Mode (CFB) –

In this mode the cipher is given as feedback to the next block of encryption with some new specifications: first an initial vector IV is used for first encryption and output bits are divided as set of *s* and *b-s* bits the left hand side *s* bits are selected and are applied an XOR operation with plaintext bits. The result given as input to a shift register and the process continues. The encryption and decryption process for the same is shown below, both of them use encryption algorithm. Generally s bit size is 8 bits.


IV
Shift register
b – s bits
s bits
64
K
Encrypt
Select
s bits
Discard
b – s bits
P1
P2
PM
C1
C2
CM
CM–1
(a) Encryption

Advantages of CFB –

- Since, there is some data loss due to use of shift register, thus it is difficult for applying cryptanalysis.
- General purpose stream oriented transmission i.e. authentication

Output Feedback Block –

The output feedback mode follows nearly same process as the Cipher Feedback mode except that it sends the encrypted output as feedback instead of the actual cipher which is XOR output. In this output feedback mode, all bits of the block are sending instead of sending selected *s* bits. The Output Feedback mode of block cipher holds great resistance towards bit transmission errors. It also decreases dependency or relationship of cipher on plaintext.

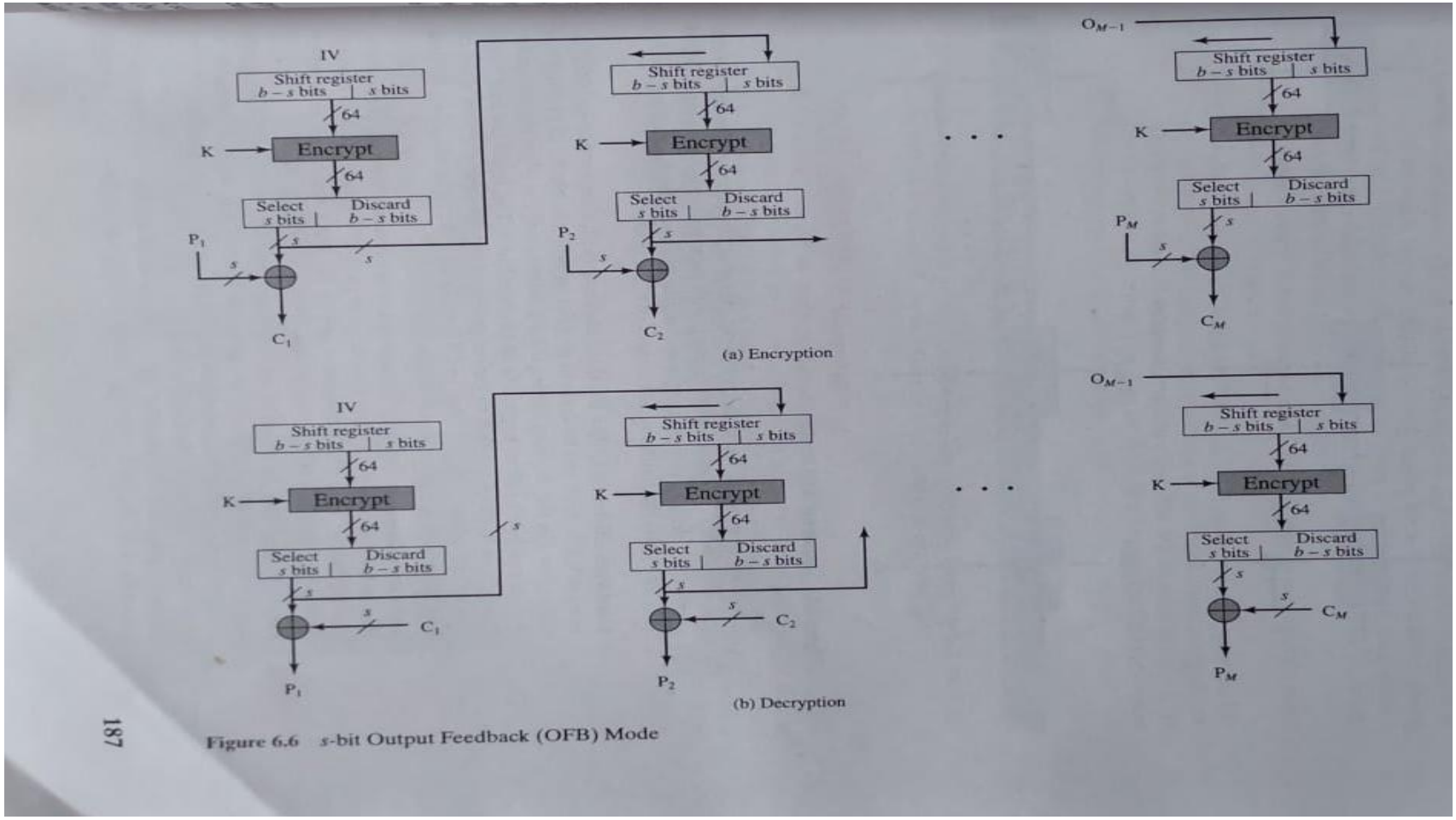


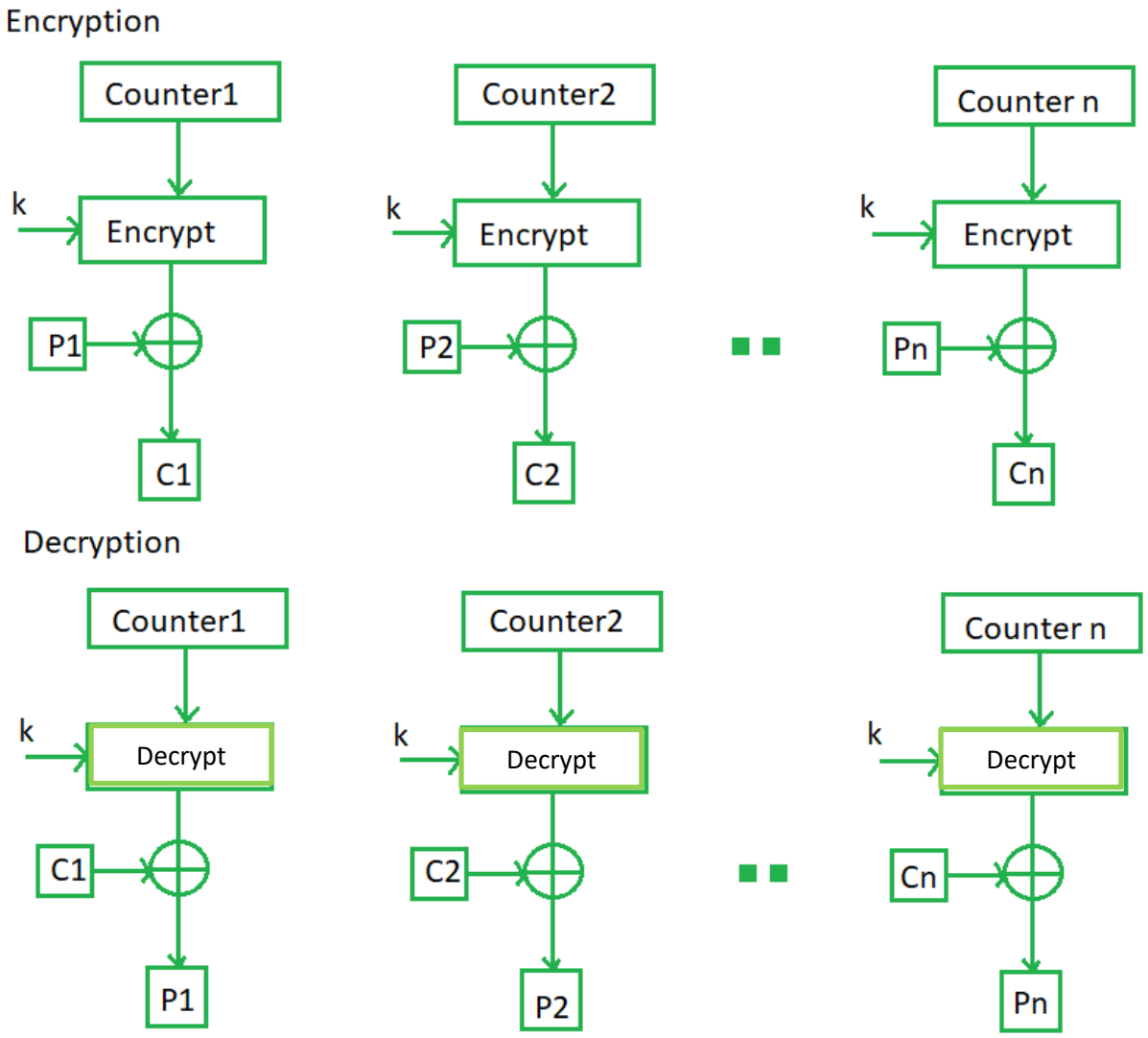
Figure 6.6 *s*-bit Output Feedback (OFB) Mode

187

Counter Mode –

The Counter Mode or CTR is a simple counter based block cipher implementation. Every time a counter initiated value is encrypted and given as

input to XOR with plaintext which results in ciphertext block. The CTR mode is independent of feedback use and thus can be implemented in parallel.
Its simple implementation is shown below:



Stream Ciphers

- Difficulty Level : Medium

In stream cipher, one byte is encrypted at a time while in block cipher ~128 bits are encrypted at a time.

Initially, a key(k) will be supplied as input to pseudorandom bit generator and then it produces a random 8-bit output which is treated as keystream.

The resulted keystream will be of size 1 byte, i.e., 8 bits.

1. Stream Cipher follows the sequence of pseudorandom number stream.
2. One of the benefits of following stream cipher is to make cryptanalysis more difficult, so the number of bits chosen in the Keystream must be long in order to make cryptanalysis more difficult.
3. By making the key longer it is also safe against brute force attacks.
4. The longer the key the stronger security is achieved, preventing any attack.
5. Keystream can be designed more efficiently by including more number of 1s and 0s, for making cryptanalysis more difficult.
6. Considerable benefit of a stream cipher is, it requires few lines of code compared to block cipher.

Encryption:

For Encryption,

- Plain Text and Keystream produces Cipher Text (Same keystream will be used for decryption.).
- The Plaintext will undergo XOR operation with keystream bit-by-bit and produces the Cipher Text.

Example –

Plain Text : 10011001

Keystream : 11000011

``````````````````

Cipher Text : 01011010

**Decryption :**

For Decryption,

- Cipher Text and Keystream gives the original Plain Text (Same keystream will be used for encryption.).
- The Ciphertext will undergo XOR operation with keystream bit-by-bit and produces the actual Plain Text.

**Example –**

Cipher Text : 01011010

Keystream : 11000011

``````````````````

Plain Text : 10011001

Decryption is just the reverse process of Encryption i.e. performing XOR with Cipher Text.

come

a=97 b=98 c=99 d=100 e=101

99120122101

99 -1100011

120-1111000

122-1111010

101-1100101

Plain text in bits- 11000111111000111110101100101

11000011110000111110000111100

00000100001000000000101011001-cipher text

Cipher - 01000010000100000000101011001

Key 11000011110000111110000111100

Plain text 10 00001110

2	99	
	49	1

RC4 Encryption Algorithm

- Difficulty Level : Easy

RC4 is a stream cipher and variable length key algorithm. This algorithm encrypts one byte at a time (or larger units on a time).
A key input is pseudorandom bit generator that produces a stream 8-bit number that is unpredictable without knowledge of input key, The output of the generator is called key-stream, is combined one byte at a time with the plaintext stream cipher using X-OR operation.
Example:
RC4 Encryption

10011000 ? 01010000 = 11001000

RC4 Decryption

11001000 ? 01010000 = 10011000

Co me mo on so on th en wh en

Key-Generation Algorithm –
A variable-length key from 1 to 256 byte is used to initialize a 256-byte state vector S, with elements S[0] to S[255]. For encryption and decryption, a byte k is generated from S by selecting one of the 255 entries in a systematic fashion, then the entries in S are permuted again.

1. **Key-Scheduling Algorithm:**
   Initialization: The entries of S are set equal to the values from 0 to 255 in ascending order, a temporary vector T, is created.
   If the length of the key k is 256 bytes, then k is assigned to T. Otherwise, for a key with length(k-len) bytes, the first k-len elements of T as copied from K and then K is repeated as many times as necessary to fill T. The idea is illustrated as follow:

   for

   i = 0 to 255 do S[i] = i;

   T[i] = K[i mod k - len];

2. we use T to produce the initial permutation of S. Starting with S[0] to S[255], and for each S[i] algorithm swap it with another byte in S according to a scheme dictated by T[i], but S will still contain values from 0 to 255 :

   j = 0;

   for

   i = 0 to 255 do

   {

   j = (j + S[i] + T[i])mod 256;

   Swap(S[i], S[j]);

}

1. **Pseudo random generation algorithm (Stream Generation):**
   Once the vector S is initialized, the input key will not be used. In this step, for each S[i] algorithm swap it with another byte in S according to a scheme dictated by the current configuration of S. After reaching S[255] the process continues, starting from S[0] again

```
i, j = 0;

while (true)

   i = (i + 1)mod 256;

j = (j + S[i])mod 256;

Swap(S[i], S[j]);

t = (S[i] + S[j])mod 256;

k = S[t];
```

2. **Encrypt using X-Or():**

Simplified RC4 Example

Example

Steven Gordon

1 Simplified RC4 Example

Lets consider the stream cipher RC4, but instead of the full 256 bytes, we will use 8 x 3-bits. That is, the state vector **S** is 8 x 3-bits. We will operate on 3-bits of plaintext at a time since S can take the values 0 to 7, which can be represented as 3 bits.

Assume we use a 4 x 3-bit key of **K** = [1 2 3 6]. And a plaintext **P** = [1 2 2 2]

The first step is to generate the stream.

Initialise the state vector **S** and temporary vector **T**. **S** is initialised so the S[i] = i, and **T** is initialised so it is the key **K** (repeated as necessary).

S = [0 1 2 3 4 5 6 7]
T = [1 2 3 6 1 2 3 6]

Now perform the initial permutation on S.

```
j = 0;
for i = 0 to 7 do
            j = (j + S[i] + T[i]) mod 8
            Swap(S[i],S[j]);
end
```

For i = 0:
j = (0 + 0 + 1) mod 8
= 1
Swap(S[0],S[1]);

S = [1 0 2 3 4 5 6 7]

For i = 1:
j = 3
Swap(S[1],S[3])
S = [1 3 2 0 4 5 6 7];

For i = 2:
j = 0
Swap(S[2],S[0]);
S = [2 3 1 0 4 5 6 7];

For i = 3:
j = 6;
Swap(S[3],S[6])
S = [2 3 1 6 4 5 0 7];

For i = 4:
j = 3
Swap(S[4],S[3])
S = [2 3 1 4 6 5 0 7];

For i = 5:
j = 2
Swap(S[5],S[2]);
S = [2 3 5 4 6 1 0 7];

For i = 6:
j = 5;
Swap(S[6],S[4])
S = [2 3 5 4 0 1 6 7];

For i = 7:
j = 2;
Swap(S[7],S[2])
S = [2 3 7 4 0 1 6 5];

Hence, our initial permutation of S = [2 3 7 4 0 1 6 5];

Now we generate 3-bits at a time, k, that we XOR with each 3-bits of plaintext to produce the ciphertext. The 3-bits k is generated by:

```
i, j = 0;
while (true) {
      i = (i + 1) mod 8;
      j = (j + S[i]) mod 8;
      Swap (S[i], S[j]);
      t = (S[i] + S[j]) mod 8;
      k = S[t]; }
```

The first iteration:
S = [2 3 7 4 0 1 6 5]
i = (0 + 1) mod 8 = 1
j = (0 + S[1]) mod 8 = 3
Swap(S[1],S[3])
S = [2 4 7 3 0 1 6 5]
t = (S[1] + S[3]) mod 8 = 7
k = S[7] = 5

Remember, P = [1 2 2 2]

So our first 3-bits of ciphertext is obtained by: k XOR P
5 XOR 1 = 101 XOR 001 = 100 = 4

The second iteration:
S = [2 4 7 3 0 1 6 5]
i = (1 + 1) mod 8 = 2
j = (2 + S[2]) mod 8 = 1
Swap(S[2],S[1])
S = [2 7 4 3 0 1 6 5]

t = (S[2] + S[1]) mod 8 = 3
k = S[3] = 3

Second 3-bits of ciphertext are:
3 XOR 2 = 011 XOR 010 = 001 = 1

The third iteration:
S = [2 7 4 3 0 1 6 5]
i = (2 + 1) mod 8 = 3
j = (1 + S[3]) mod 8 = 4
Swap(S[3],S[4])
S = [2 7 4 0 3 1 6 5]
t = (S[3] + S[4]) mod 8 = 3
k = S[3] = 0

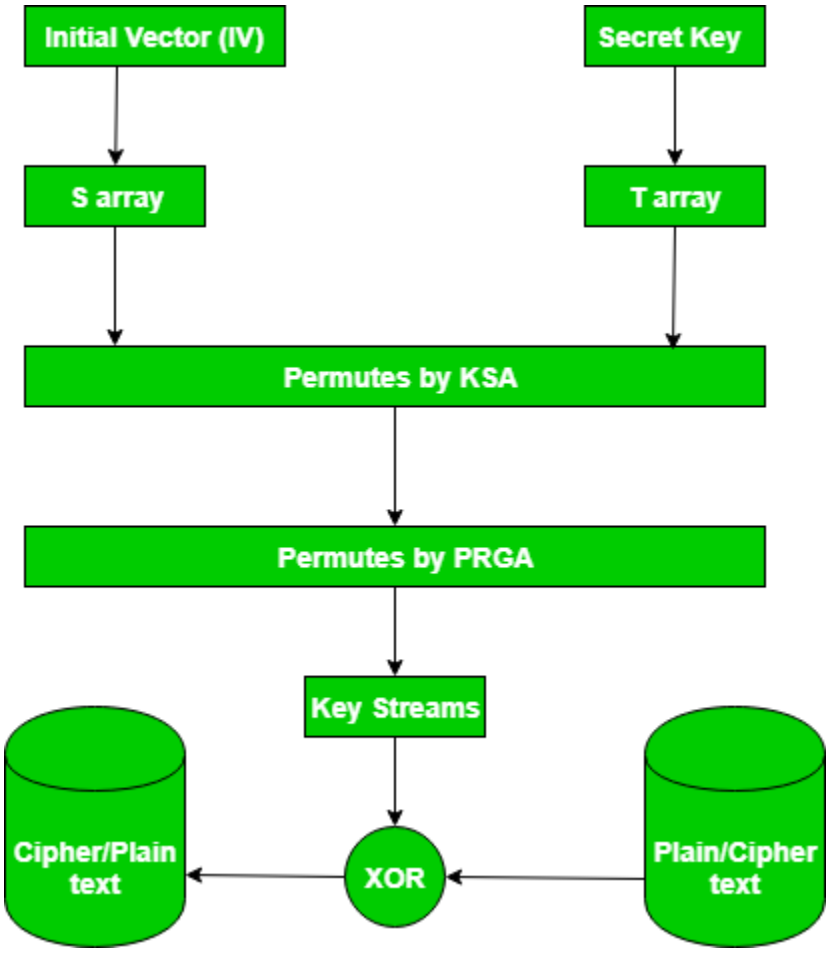
Third 3-bits of ciphertext are:
0 XOR 2 = 000 XOR 010 = 010 = 2

The final iteration:
S = [2 7 4 0 3 1 6 5]
i = (1 + 3) mod 8 = 4
j = (4 + S[4]) mod 8 = 7
Swap(S[4],S[7])
S = [2 7 4 0 5 1 6 3]
t = (S[4] + S[7]) mod 8 = 0
k = S[0] = 2

Last 3-bits of ciphertext are:
2 XOR 2 = 010 XOR 010 = 000 = 0

So to encrypt the plaintext stream **P** = [1 2 2 2] with key **K** = [1 2 3 6] using our simplified RC4 stream cipher we get **C** = [4 1 2 0].

(or in binary: **P** = 001010010010, **K** = 001010011110 and **C** = 100001010000)



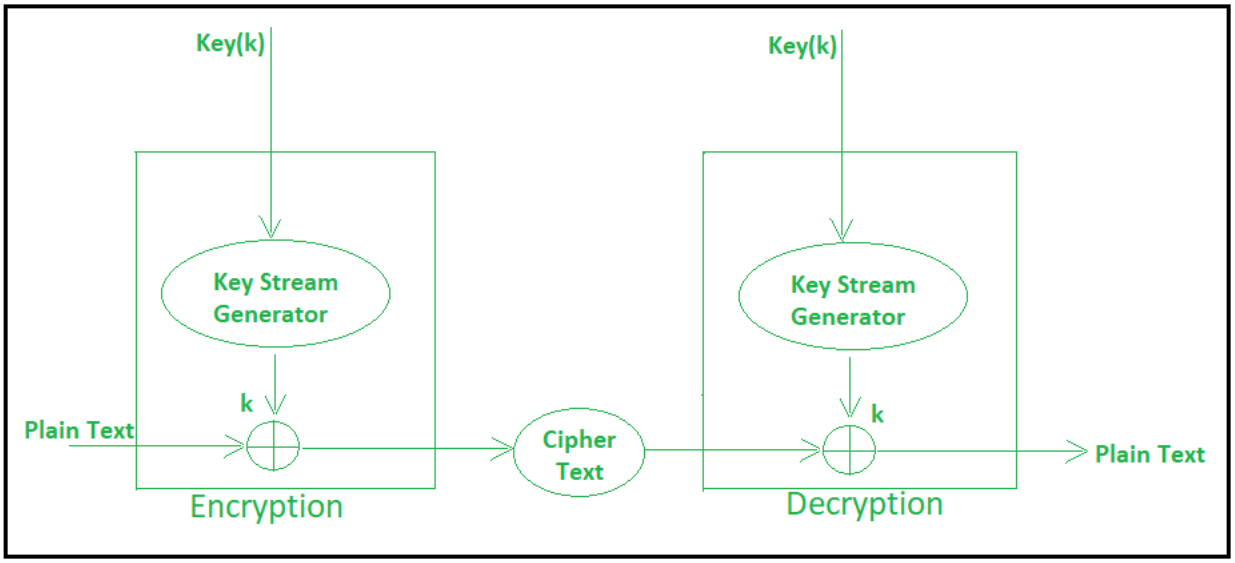


Diagram of Stream Cipher

INTRODUCTION TO PUBLIC-KEY CRYPTOGRAPHY

Public-key cryptography and related standards underlie the security features of many products such as signed and encrypted email, single sign-on, and Transport Layer Security/Secure Sockets Layer (SSL/TLS) communications. This chapter covers the basic concepts of public-key cryptography.

Internet traffic, which passes information through intermediate computers, can be intercepted by a third party:

Eavesdropping

Information remains intact, but its privacy is compromised. For example, someone could gather credit card numbers, record a sensitive conversation, or intercept classified information.

Tampering

Information in transit is changed or replaced and then sent to the recipient. For example, someone could alter an order for goods or change a person's resume.

Impersonation

Information passes to a person who poses as the intended recipient. Impersonation can take two forms:

- *Spoofing.* A person can pretend to be someone else. For example, a person can pretend to have the email address jdoe@example.net or a computer can falsely identify itself as a site called www.example.net.
- *Misrepresentation.* A person or organization can misrepresent itself. For example, a site called www.example.net can purport to be an on-line furniture store when it really receives credit-card payments but never sends any goods.

Public-key cryptography provides protection against Internet-based attacks through:

Encryption and decryption

Encryption and decryption allow two communicating parties to disguise information they send to each other. The sender encrypts, or scrambles, information before sending it. The receiver decrypts, or unscrambles, the information after receiving it. While in transit, the encrypted information is unintelligible to an intruder.

Tamper detection

Tamper detection allows the recipient of information to verify that it has not been modified in transit. Any attempts to modify or substitute data are detected.

Authentication

Authentication allows the recipient of information to determine its origin by confirming the sender's identity.

Nonrepudiation

Nonrepudiation prevents the sender of information from claiming at a later date that the information was never sent.

1.1. Encryption and Decryption

Encryption is the process of transforming information so it is unintelligible to anyone but the intended recipient. *Decryption* is the process of decoding encrypted information. A cryptographic algorithm, also called a *cipher*, is a mathematical function used for encryption or decryption. Usually, two related functions are used, one for encryption and the other for decryption.

With most modern cryptography, the ability to keep encrypted information secret is based not on the cryptographic algorithm, which is widely known, but on a number called a *key* that must be used with the algorithm to produce an encrypted result or to decrypt previously encrypted information. Decryption with the correct key is simple. Decryption without the correct key is very difficult, if not impossible.

1.1.1. Symmetric-Key Encryption

With symmetric-key encryption, the encryption key can be calculated from the decryption key and vice versa. With most symmetric algorithms, the same key is used for both encryption and decryption, as shown in Figure 1.1, "Symmetric-Key Encryption".

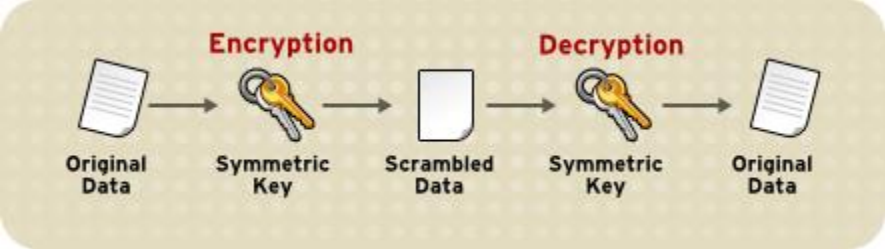


Figure 1.1. Symmetric-Key Encryption

Implementations of symmetric-key encryption can be highly efficient, so that users do not experience any significant time delay as a result of the encryption and decryption.

Symmetric-key encryption is effective only if the symmetric key is kept secret by the two parties involved. If anyone else discovers the key, it affects both confidentiality and authentication. A person with an unauthorized symmetric key not only can decrypt messages sent with that key, but can encrypt new messages and send them as if they came from one of the legitimate parties using the key.

Symmetric-key encryption plays an important role in SSL/TLS communication, which is widely used for authentication, tamper detection, and encryption over TCP/IP networks. SSL/TLS also uses techniques of public-key encryption, which is described in the next section.

1.1.2. Public-Key Encryption

Public-key encryption (also called asymmetric encryption) involves a pair of keys, a public key and a private key, associated with an entity. Each public key is published, and the corresponding private key is kept secret. (For more information about the way public keys are published, see Section 1.3, "Certificates and Authentication".) Data encrypted with a public key can be decrypted only with the corresponding private key. Figure 1.2, "Public-Key Encryption" shows a simplified view of the way public-key encryption works.

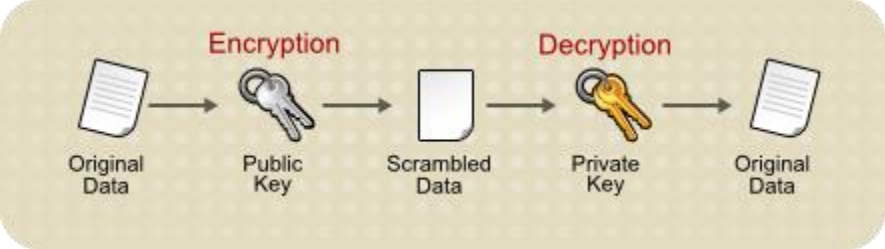


Figure 1.2. Public-Key Encryption

The scheme shown in Figure 1.2, "Public-Key Encryption" allows public keys to be freely distributed, while only authorized people are able to read data encrypted using this key. In general, to send encrypted data, the data is encrypted with that person's public key, and the person receiving the encrypted data decrypts it with the corresponding private key.

Compared with symmetric-key encryption, public-key encryption requires more processing and may not be feasible for encrypting and decrypting large amounts of data. However, it is possible to use public-key encryption to send a symmetric key, which can then be used to encrypt additional data. This is the approach used by the SSL/TLS protocols.

The reverse of the scheme shown in Figure 1.2, "Public-Key Encryption" also works: data encrypted with a private key can be decrypted only with the corresponding public key. This is not a recommended practice to encrypt sensitive data, however, because it means that anyone with the public key, which is by definition published, could decrypt the data. Nevertheless, private-key encryption is useful because it means the private key can be used to sign data with a digital signature, an important requirement for electronic commerce and other commercial applications of cryptography. Client software such as Mozilla Firefox can then use the public key to confirm that the message was signed with the appropriate private key and that it has not been tampered with since being signed. Section 1.2, "Digital Signatures" illustrates how this confirmation process works.

1.1.3. Key Length and Encryption Strength

Breaking an encryption algorithm is finding the key to the access the encrypted data in plain text. For symmetric algorithms, breaking the algorithm usually means trying to determine the key used to encrypt the text. For a public key algorithm, breaking the algorithm usually means acquiring the shared secret information between two recipients.

One method of breaking a symmetric algorithm is to simply try every key within the full algorithm until the right key is found. For public key algorithms, since half of the key pair is publicly known, the other half (private key) can be derived using published, though complex, mathematical calculations. Manually finding the key to break an algorithm is called a brute force attack.

Breaking an algorithm introduces the risk of intercepting, or even impersonating and fraudulently verifying, private information.

The *key strength* of an algorithm is determined by finding the fastest method to break the algorithm and comparing it to a brute force attack.

For symmetric keys, encryption strength is often described in terms of the size or *length* of the keys used to perform the encryption: longer keys generally provide stronger encryption. Key length is measured in bits.

An encryption key is considered full strength if the best known attack to break the key is no faster than a brute force attempt to test every key possibility.

Different types of algorithms — particularly public key algorithms — may require different key lengths to achieve the same level of encryption strength as a symmetric-key cipher. The RSA cipher can use only a subset of all possible values for a key of a given length, due to the nature of the mathematical problem on which it is based. Other ciphers, such as those used for symmetric-key encryption, can use all possible values for a key of a given length. More possible matching options means more security.

Because it is relatively trivial to break an RSA key, an RSA public-key encryption cipher must have a very long key — at least 2048 bits — to be considered cryptographically strong. On the other hand, symmetric-key ciphers are reckoned to be equivalently strong using a much shorter key length, as little as 80 bits for most algorithms. Similarly, public-key ciphers based on the elliptic curve cryptography (ECC), such as the Elliptic Curve Digital Signature Algorithm (ECDSA) ciphers, also require less bits than RSA ciphers.

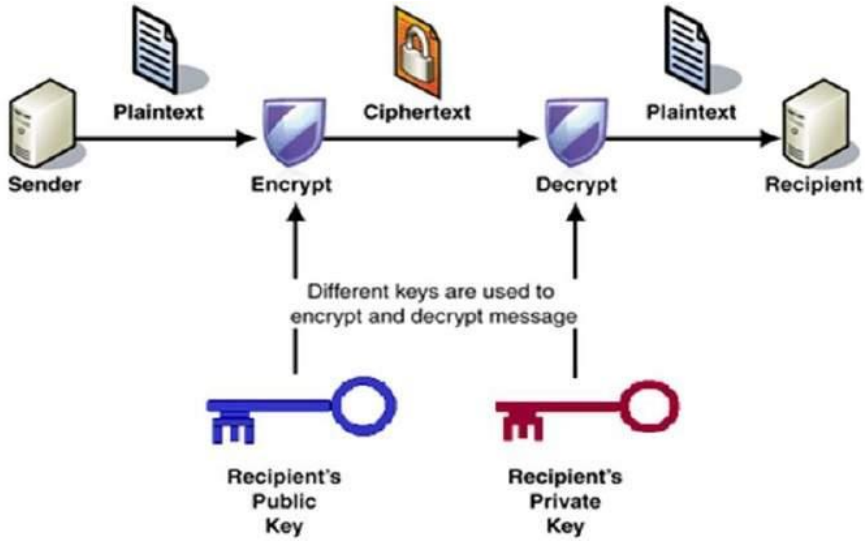
Public Key Cryptography

Unlike symmetric key cryptography, we do not find historical use of public-key cryptography. It is a relatively new concept.

Symmetric cryptography was well suited for organizations such as governments, military, and big financial corporations were involved in the classified communication.

With the spread of more unsecure computer networks in last few decades, a genuine need was felt to use cryptography at larger scale. The symmetric key was found to be non-practical due to challenges it faced for key management. This gave rise to the public key cryptosystems.

The process of encryption and decryption is depicted in the following illustration −



The most important properties of public key encryption scheme are −

- Different keys are used for encryption and decryption. This is a property which set this scheme different than symmetric encryption scheme.
- Each receiver possesses a unique decryption key, generally referred to as his private key.
- Receiver needs to publish an encryption key, referred to as his public key.
- Some assurance of the authenticity of a public key is needed in this scheme to avoid spoofing by adversary as the receiver. Generally, this type of cryptosystem involves trusted third party which certifies that a particular public key belongs to a specific person or entity only.
- Encryption algorithm is complex enough to prohibit attacker from deducing the plaintext from the ciphertext and the encryption (public) key.
- Though private and public keys are related mathematically, it is not be feasible to calculate the private key from the public key. In fact, intelligent part of any public-key cryptosystem is in designing a relationship between two keys.

There are three types of Public Key Encryption schemes. We discuss them in following sections −

Diffie-Hellman key exchange, also called exponential key exchange, is a method of digital encryption that uses numbers raised to specific powers to produce decryption keys on the basis of components that are never directly transmitted, making the task of a would-be code breaker mathematically overwhelming.

To implement Diffie-Hellman, the two end users Alice and Bob, while communicating over a channel they know to be private, mutually agree on positive whole numbers *p* and *q*, such that *p* is a prime number and *q* is a generator of *p*. The generator *q* is a number that, when raised to positive whole-number powers less than *p*, never produces the same result for any two such whole numbers. The value of *p* may be large but the value of *q* is usually small.

Once Alice and Bob have agreed on *p* and *q* in private, they choose positive whole-number personal keys *a* and *b*, both less than the prime-number modulus *p*. Neither user divulges their personal key to anyone; ideally they memorize these numbers and do not write them down or store them anywhere. Next, Alice and Bob compute public keys *a** and *b** based on their personal keys according to the formulas

$$a^* = q^a \bmod p$$

and

$$b^* = q^b \bmod p$$

The two users can share their public keys *a** and *b** over a communications medium assumed to be insecure, such as the Internet or a corporate wide area network (WAN). From these public keys, a number *x* can be generated by either user on the basis of their own personal keys. Alice computes *x* using the formula

$$x = (b^*)^a \bmod p$$

Bob computes *x* using the formula

$$x = (a^*)^b \bmod p$$

The value of *x* turns out to be the same according to either of the above two formulas. However, the personal keys *a* and *b*, which are critical in the calculation of *x*, have not been transmitted over a public medium. Because it is a large and apparently random number, a potential hacker has almost no chance of

correctly guessing *x*, even with the help of a powerful computer to conduct millions of trials. The two users can therefore, in theory, communicate privately over a public medium with an encryption method of their choice using the decryption key *x*.

The most serious limitation of Diffie-Hellman in its basic or "pure" form is the lack of authentication. Communications using Diffie-Hellman all by itself are vulnerable to man in the middle attacks. Ideally, Diffie-Hellman should be used in conjunction with a recognized authentication method such as digital signatures to verify the identities of the users over the public communications medium. Diffie-Hellman is well suited for use in data communication but is less often used for data stored or archived over long periods of time.

Step by Step Explanation

Alice	Bob
Public Keys available = P, G	Public Keys available = P, G
Private Key Selected = a	Private Key Selected = b
Key generated =	Key generated =

Key generated =

Exchange of generated keys takes place

Key received = y key received = x

Generated Secret Key = Generated Secret Key =

Alice	Bob

Algebraically, it can be shown that

Users now have a symmetric secret key to encrypt

Example:
Step 1: Alice and Bob get public numbers P = 23, Q = 9

Step 2: Alice selected a private key a = 4 and

Bob selected a private key b = 3

Step 3: Alice and Bob compute public values

Alice: $x=Q^a \bmod P$

x =(9^4 mod 23) = (6561 mod 23) = 6

Bob: $y =Q^b \bmod P$

y = (9^3 mod 23) = (729 mod 23) = 16

Step 4: Alice and Bob exchange public numbers

Step 5: Alice receives public key y =16 and

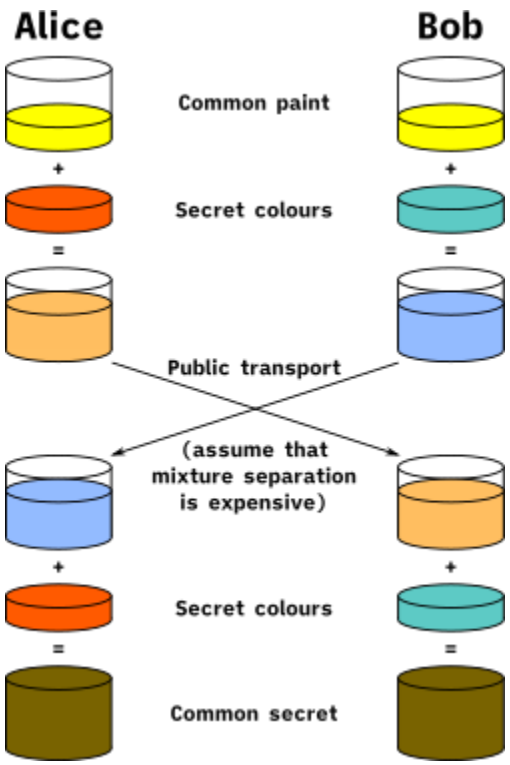
Bob receives public key x = 6

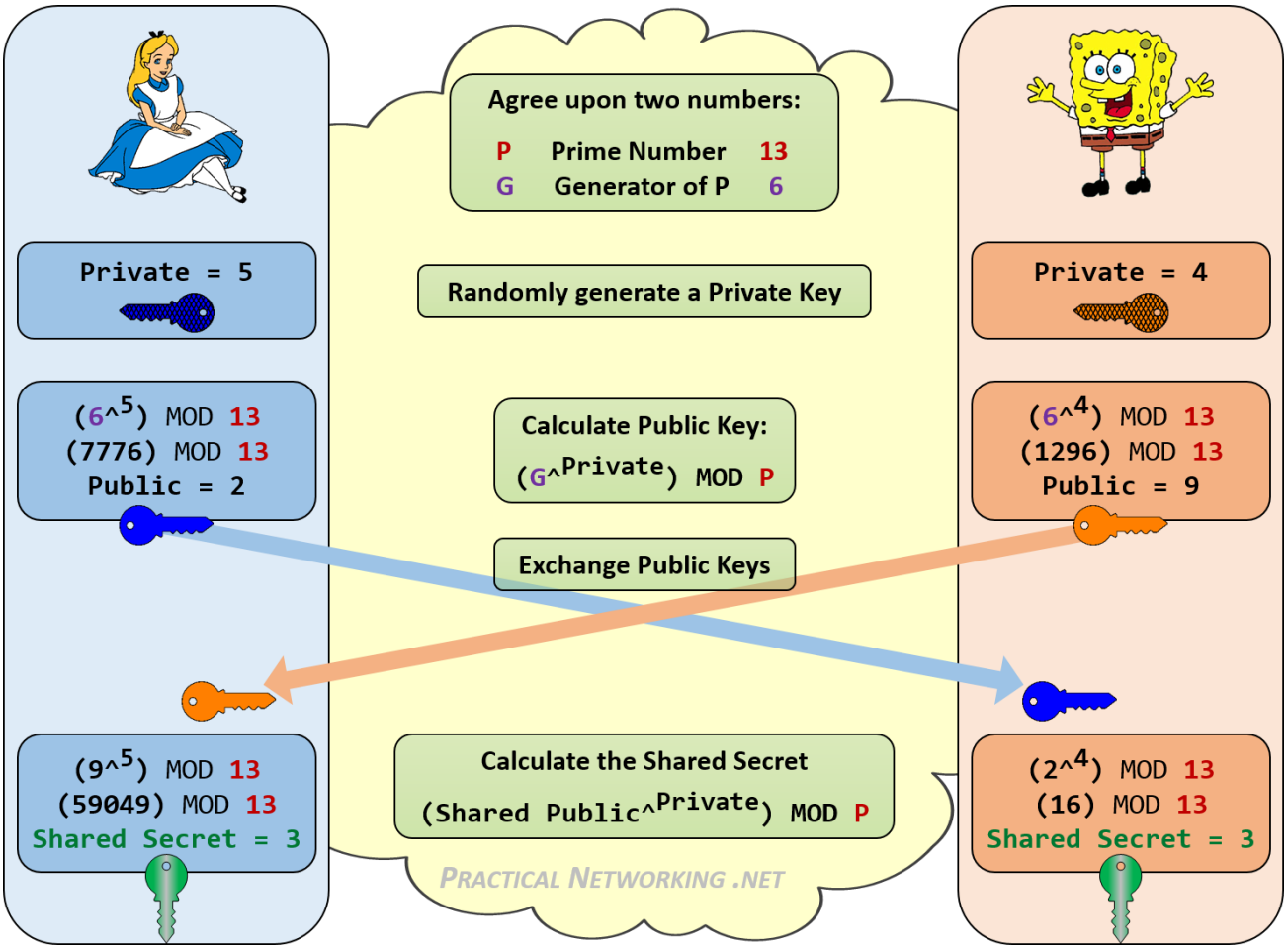
Step 6: Alice and Bob compute symmetric keys

Alice: ka = y^a mod p = 65536 mod 23 = 9

Bob: kb = x^b mod p = 216 mod 23 = 9

Step 7: 9 is the shared secret.


Alice
Bob
Common paint
+
+
Secret colours
=
=
Public transport
(assume that mixture separation is expensive)
+
+
Secret colours
=
=
Common secret



RSA Cryptosystem

This cryptosystem is one the initial system. It remains most employed cryptosystem even today. The system was invented by three scholars **Ron Rivest, Adi Shamir,** and **Len Adleman** and hence, it is termed as RSA cryptosystem.

We will see two aspects of the RSA cryptosystem, firstly generation of key pair and secondly encryption-decryption algorithms.

Generation of RSA Key Pair

Each person or a party who desires to participate in communication using encryption needs to generate a pair of keys, namely public key and private key. The process followed in the generation of keys is described below −

- **Generate the RSA modulus (n)**
  - Select two large primes, p and q.
  - Calculate n=p*q. For strong unbreakable encryption, let n be a large number, typically a minimum of 512 bits.

- **Find Derived Number (e)**
  - Number **e** must be greater than 1 and less than G=(p − 1)(q − 1).

    1>e>G
  - There must be no common factor for e and G=(p − 1)(q − 1) except for 1. In other words two numbers e and (p – 1)(q – 1) are coprime.
- **Form the public key**
  - The pair of numbers (n, e) form the RSA public key and is made public.
  - Interestingly, though n is part of the public key, difficulty in factorizing a large prime number ensures that attacker cannot find in finite time the two primes (p & q) used to obtain n. This is strength of RSA.
- **Generate the private key**
  - Private Key d is calculated from p, q, and e. For given n and e, there is unique number d.
  - Number d is the inverse of e modulo (p - 1)(q – 1). This means that d is the number less than (p - 1)(q - 1) such that when multiplied by e, it is equal to 1 modulo (p - 1)(q - 1).
  - This relationship is written mathematically as follows −

ed = 1 mod (p − 1)(q − 1)

The Extended Euclidean Algorithm takes p, q, and e as input and gives d as output.

Example

An example of generating RSA Key pair is given below. (For ease of understanding, the primes p & q taken here are small values. Practically, these values are very high).

- Let two primes be p = 7 and q = 13. Thus, modulus n = pq = 7 x 13 = 91.
- Select e = 5, which is a valid choice since there is no number that is common factor of 5 and G=(p − 1)(q − 1) = 6 × 12 = 72, except for 1.

  e should be a prime number and must be greater than 1 and less than(p-1)(q-1)

  e>1 and e<72

  2,3,4,5,6,7,8,9,10,11,12…….71

  Prime number between 2 and 71

  2,3,5,7,11,13,17,19,23,29,31,37,43,47,53,59,61,67,71

  e*d/(p-1)(q-1) remainder should be 1

  e=5 d=29

  e*d=1 mod G

  5*29=1 mod(72)

- The pair of numbers (n, e) = (91, 5) forms the public key and can be made available to anyone whom we wish to be able to send us encrypted messages.
- Input p = 7, q = 13, and e = 5 to the Extended Euclidean Algorithm. The output will be d = 29.
- Check that the d calculated is correct by computing −

de = 29 × 5 = 145 = 1 mod 72

- Hence, public key is (91, 5) and private keys is (91, 29).

Example 2

P=3 q=7

N=p*q=21

G =(p-1)*(q-1)=2*6=12

Find e

1>e>g or 1>e>12

2 3 4 5 6 7 8 9 10 11

gcd(e,n)=1

(2,21) (5,21) (11,21)

2 or 5 or 11

ed=1 mod (G)

12-13

24-25

36-37

48-49

60-61

72-73

84-85

96-97

108-109

120-121

E=5 d=5

5*5=1 mod(12)

Public kay(n,e)=(21,5)

Private key(n,d)=(21,5)

Example 3

P=5 q=11

N=P*q=55

G=(p-1)*(q-1)=4*10=40

1>e>40

2,3,4,…….40

GCD(n,e)=1

Eg=1 mod (g)

40-41

80-81

120-121

160-161

200-201

240-241

280-281

320-321

360-361

400-401

Example 4

P=7 q=13

Encryption and Decryption

Once the key pair has been generated, the process of encryption and decryption are relatively straightforward and computationally easy.

Interestingly, RSA does not directly operate on strings of bits as in case of symmetric key encryption. It operates on numbers modulo n. Hence, it is necessary to represent the plaintext as a series of numbers less than n.

RSA Encryption

- Suppose the sender wish to send some text message to someone whose public key is (n, e).
- The sender then represents the plaintext as a series of numbers less than n.

- To encrypt the first plaintext P, which is a number modulo n. The encryption process is simple mathematical step as −

$C = P^e \bmod n$

- In other words, the ciphertext C is equal to the plaintext P multiplied by itself e times and then reduced modulo n. This means that C is also a number less than n.
- Returning to our Key Generation example with plaintext P = 10, we get ciphertext C −

$C = 10^5 \bmod 91$

RSA Decryption

- The decryption process for RSA is also very straightforward. Suppose that the receiver of public-key pair (n, e) has received a ciphertext C.
- Receiver raises C to the power of his private key d. The result modulo n will be the plaintext P.

$\text{Plaintext} = C^d \bmod n$

- Returning again to our numerical example, the ciphertext C = 82 would get decrypted to number 10 using private key 29 −

$\text{Plaintext} = 82^{29} \bmod 91 = 10$

RSA Analysis

The security of RSA depends on the strengths of two separate functions. The RSA cryptosystem is most popular public-key cryptosystem strength of which is based on the practical difficulty of factoring the very large numbers.

- **Encryption Function** − It is considered as a one-way function of converting plaintext into ciphertext and it can be reversed only with the knowledge of private key d.
- **Key Generation** − The difficulty of determining a private key from an RSA public key is equivalent to factoring the modulus n. An attacker thus cannot use knowledge of an RSA public key to determine an RSA private key unless he can factor n. It is also a one way function, going from p & q values to modulus n is easy but reverse is not possible.

If either of these two functions are proved non one-way, then RSA will be broken. In fact, if a technique for factoring efficiently is developed then RSA will no longer be safe.

The strength of RSA encryption drastically goes down against attacks if the number p and q are not large primes and/ or chosen public key e is a small number.

ElGamal Cryptosystem

Along with RSA, there are other public-key cryptosystems proposed. Many of them are based on different versions of the Discrete Logarithm Problem.

ElGamal cryptosystem, called **Elliptic Curve Variant**, is based on the Discrete Logarithm Problem. It derives the strength from the assumption that the discrete logarithms cannot be found in practical time frame for a given number, while the inverse operation of the power can be computed efficiently.

Let us go through a simple version of ElGamal that works with numbers modulo p. In the case of elliptic curve variants, it is based on quite different number systems.

Generation of ElGamal Key Pair

Each user of ElGamal cryptosystem generates the key pair through as follows −

- **Choosing a large prime p.** Generally a prime number of 1024 to 2048 bits length is chosen.
- **Choosing a generator element g.**
  - This number must be between 1 and p − 1, but cannot be any number.
  - It is a generator of the multiplicative group of integers modulo p. This means for every integer m co-prime to p, there is an integer k such that g^k=a mod n.

    For example, 3 is generator of group 5 (Z_5 = {1, 2, 3, 4}).

N	3^n	3^n mod 5
1	3	3
2	9	4
3	27	2
4	81	1

- **Choosing the private key.** The private key x is any number bigger than 1 and smaller than p−1.
- **Computing part of the public key.** The value y is computed from the parameters p, g and the private key x as follows −

  $y = g^x \bmod p$
- **Obtaining Public key.** The ElGamal public key consists of the three parameters (p, g, y).

  For example, suppose that p = 17 and that g = 6 (It can be confirmed that 6 is a generator of group Z_{17}). The private key x can be any number bigger than 1 and smaller than 16, so we choose x = 5. The value y is then computed as follows −

  $y = 6^5 \bmod 17 = 7$
- Thus the private key is 62 and the public key is (17, 6, 7).

Encryption and Decryption

The generation of an ElGamal key pair is comparatively simpler than the equivalent process for RSA. But the encryption and decryption are slightly more complex than RSA.

ElGamal Encryption

Suppose sender wishes to send a plaintext to someone whose ElGamal public key is (p, g, y), then −

- Sender represents the plaintext as a series of numbers modulo p.
- To encrypt the first plaintext P, which is represented as a number modulo p. The encryption process to obtain the ciphertext C is as follows −
    - Randomly generate a number k;
    - Compute two values C1 and C2, where −

  $C1 = g^k \bmod p$
  $C2 = (P^* y^k) \bmod p$
- Send the ciphertext C, consisting of the two separate values (C1, C2), sent together.
- Referring to our ElGamal key generation example given above, the plaintext P = 13 is encrypted as follows −
    - Randomly generate a number, say k = 10
    - Compute the two values C1 and C2, where −

  $C1 = 6^{10} \bmod 17$
  $C2 = (13^* 7^{10}) \bmod 17 = 9$
- Send the ciphertext C = (C1, C2) = (15, 9).

ElGamal Decryption

- To decrypt the ciphertext (C1, C2) using private key x, the following two steps are taken −
    - Compute the modular inverse of $(C1)^x$ modulo p, which is $(C1)^{-x}$, generally referred to as decryption factor.
    - Obtain the plaintext by using the following formula −

  $C2 \times (C1)^{-x} \bmod p = \text{Plaintext}$
- In our example, to decrypt the ciphertext C = (C1, C2) = (15, 9) using private key x = 5, the decryption factor is

  $15^{-5} \bmod 17 = 9$
- Extract plaintext $P = (9 \times 9) \bmod 17 = 13$.

ElGamal Analysis

In ElGamal system, each user has a private key x. and has **three components** of public key − **prime modulus p, generator g, and public $Y = g^x \bmod p$**. The strength of the ElGamal is based on the difficulty of discrete logarithm problem.

The secure key size is generally > 1024 bits. Today even 2048 bits long key are used. On the processing speed front, Elgamal is quite slow, it is used mainly for key authentication protocols. Due to higher processing efficiency, Elliptic Curve variants of ElGamal are becoming increasingly popular.

Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC) is a term used to describe a suite of cryptographic tools and protocols whose security is based on special versions of the discrete logarithm problem. It does not use numbers modulo p.

ECC is based on sets of numbers that are associated with mathematical objects called elliptic curves. There are rules for adding and computing multiples of these numbers, just as there are for numbers modulo p.

ECC includes a variants of many cryptographic schemes that were initially designed for modular numbers such as ElGamal encryption and Digital Signature Algorithm.

It is believed that the discrete logarithm problem is much harder when applied to points on an elliptic curve. This prompts switching from numbers modulo p to points on an elliptic curve. Also an equivalent security level can be obtained with shorter keys if we use elliptic curve-based variants.

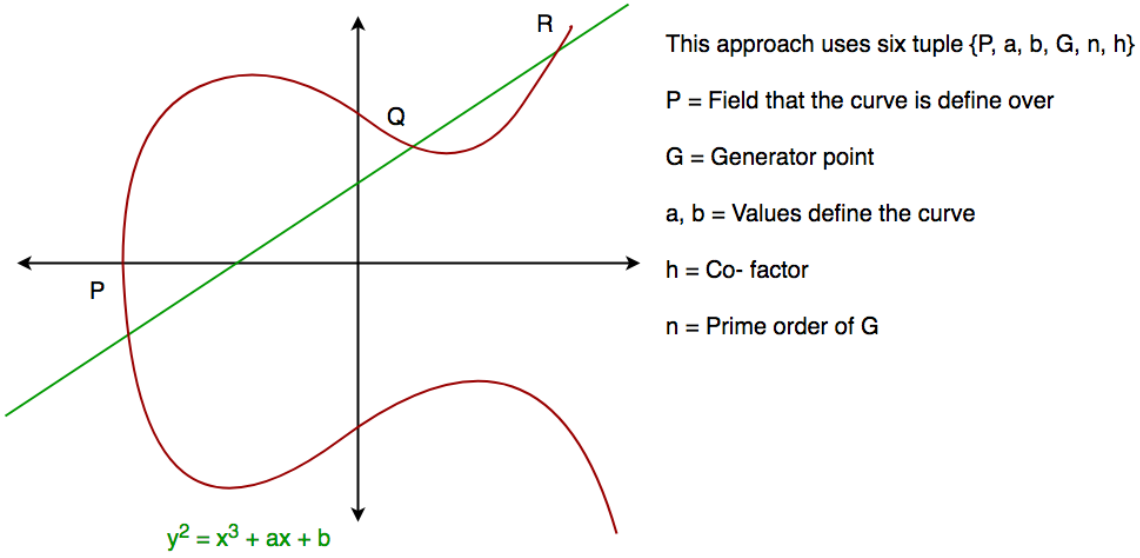
The shorter keys result in two benefits −

- Ease of key management
- Efficient computation

These benefits make elliptic-curve-based variants of encryption scheme highly attractive for application where computing resources are constrained.

Elliptic Curve Cryptography (ECC) is a key-based technique for encrypting data. ECC focuses on pairs of public and private keys for decryption and encryption of web traffic.

ECC is frequently discussed in the context of the Rivest–Shamir–Adleman (RSA) cryptographic algorithm. RSA achieves one-way encryption of things like emails, data, and software using prime factorization.



What is Elliptic Curve Cryptography?

ECC, an alternative technique to RSA, is a powerful cryptography approach. It generates security between key pairs for public key encryption by using the mathematics of elliptic curves.

RSA does something similar with prime numbers instead of elliptic curves, but ECC has gradually been growing in popularity recently due to its smaller key size and ability to maintain security. This trend will probably continue as the demand on devices to remain secure increases due to the size of keys growing, drawing on scarce mobile resources. This is why it is so important to understand elliptic curve cryptography in context.

In contrast to RSA, ECC bases its approach to public key cryptographic systems on how elliptic curves are structured algebraically over finite fields. Therefore, ECC creates keys that are more difficult, mathematically, to crack. For this reason, ECC is considered to be the next generation implementation of public key cryptography and more secure than RSA.

It also makes sense to adopt ECC to maintain high levels of both performance and security. That's because ECC is increasingly in wider use as websites strive for greater online security in customer data and greater mobile optimization, simultaneously. More sites using ECC to secure data means a greater need for this kind of quick guide to elliptic curve cryptography.

An elliptic curve for current ECC purposes is a plane curve over a finite field which is made up of the points satisfying the equation:
$y^2=x^3 + ax + b$.

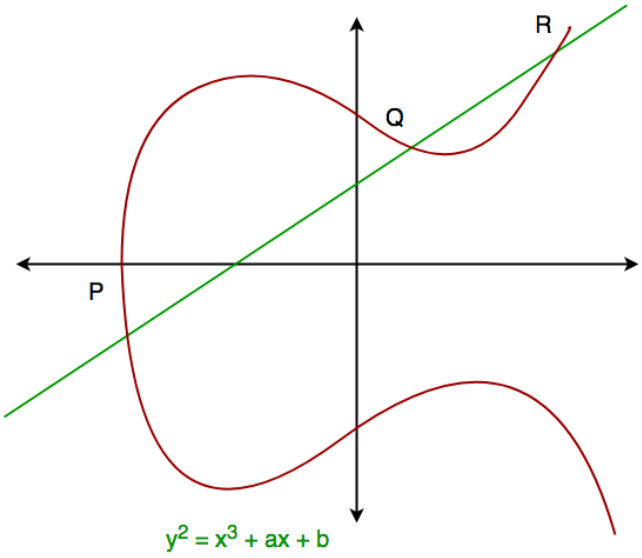
In this elliptic curve cryptography example, any point on the curve can be mirrored over the x-axis and the curve will stay the same. Any non-vertical line will intersect the curve in three places or fewer.

Elliptic Curve Cryptography (ECC) is an approach to public-key cryptography, based on the algebraic structure of elliptic curves over finite fields. ECC requires a smaller key as compared to non-ECC cryptography to provide equivalent security (a 256-bit ECC security has equivalent security attained by 3072-bit RSA cryptography).
For a better understanding of Elliptic Curve Cryptography, it is very important to understand the basics of the Elliptic Curve. An elliptic curve is a planar algebraic curve defined by an equation of the form

Where 'a' is the co-efficient of x and 'b' is the constant of the equation

The curve is non-singular; that is, its graph has no cusps or self-intersections (when the characteristic of the Co-efficient field is equal to 2 or 3).

In general, an elliptic curve looks like as shown below. Elliptic curves can intersect almost 3 points when a straight line is drawn intersecting the curve. As we can see, the elliptic curve is symmetric about the x-axis. This property plays a key role in the algorithm.



This approach uses six tuple {P, a, b, G, n, h}

P = Field that the curve is define over

G = Generator point

a, b = Values define the curve

h = Co- factor

n = Prime order of G

Diffie-Hellman algorithm

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

- For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime P and G (a primitive root of P) and two private values a and b.
- P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

Alice	**Bob**
Public Keys available = P, G	Public Keys available = P, G
Private Key Selected = a	Private Key Selected = b
Key generated = x = G^a mod P	Key generated = y = G^b mod P
Exchange of generated keys takes place	
Key received = y	key received = x
Generated Secret Key = k_a = y^a mod P	Generated Secret Key = k_b = x^b mod P
Algebraically, it can be shown that	
k_a = k_b	

Step 1: Alice and Bob get public numbers P = 23, G = 9

Step 2: Alice selected a private key a = 4 and

Bob selected a private key b = 3

Step 3: Alice and Bob compute public values

Alice: x =(9^4 mod 23) = (6561 mod 23) = 6

Bob: y = (9^3 mod 23) = (729 mod 23) = 16

Step 4: Alice and Bob exchange public numbers

Step 5: Alice receives public key y =16 and

Bob receives public key x = 6

Step 6: Alice and Bob compute symmetric keys

Alice: ka = y^a mod p = 65536 mod 23 = 9

Bob: kb = x^b mod p = 216 mod 23 = 9

Step 7: 9 is the shared secret.

Elliptic Curve Cryptography vs RSA

The difference in size to security yield between RSA and ECC encryption keys is notable. The table below shows the sizes of keys needed to provide the same level of security. In other words, an elliptic curve cryptography key of 384 bit achieves the same level of security as an RSA of 7680 bit.

RSA Key Length (bit)
1024
2048
3072
7680
15360

ECC Key Length (bit)
160
224
256
384
521

There is no linear relationship between the sizes of ECC keys and RSA keys. That is, an RSA key size that is twice as big does not translate into an ECC key size that's doubled. This compelling difference shows that ECC key generation and signing are substantially quicker than for RSA, and also that ECC uses less memory than does RSA.

Also, unlike in RSA, where both are integers, in ECC the private and public keys are not equally exchangeable. Instead, in ECC the public key is a point on the curve, while the private key is still an integer.

A quick comparison of the advantages and disadvantages of ECC and RSA algorithms looks like this:

ECC features smaller cipher texts, keys, and signatures, and faster generation of keys and signatures. Its decryption and encryption speeds are moderately fast. ECC enables lower latency than inverse throughout by computing signatures in two stages. ECC features strong protocols for authenticated key exchange and support for the tech is strong.

The main disadvantage of ECC is that it isn't easy to securely implement. Compared to RSA, which is much simpler on both the verification and encryption sides, ECC is a steeper learning curve and a bit slower for accumulating actionable results.

However, the disadvantages of RSA catch up with you soon. Key generation is slow with RSA, and so is decryption and signing, which aren't always that easy to implement securely.

Advantages of Elliptic Curve Cryptography

Public-key cryptography works using algorithms that are easy to process in one direction and difficult to process in the reverse direction. For example, RSA relies on the fact that multiplying prime numbers to get a larger number is easy, while factoring huge numbers back to the original primes is much more difficult.

However, to remain secure, RSA needs keys that are 2048 bits or longer. This makes the process slow, and it also means that key size is important.

Size is a serious advantage of elliptic curve cryptography, because it translates into more power for smaller, mobile devices. It's far simpler and requires less energy to factor than it is to solve for an elliptic curve discrete logarithm, so for two keys of the same size, RSA's factoring encryption is more vulnerable.

Using ECC, you can achieve the same security level using smaller keys. In a world where mobile devices must do more and more cryptography with less computational power, ECC offers high security with faster, shorter keys compared to RSA.

How Secure is Elliptic Curve Cryptography?

There are several potential vulnerabilities to elliptic curve cryptography, including side-channel attacks and twist-security attacks. Both types aim to invalidate the ECC's security for private keys.

Side-channel attacks including differential power attacks, fault analysis, simple power attacks, and simple timing attacks, typically result in information leaks. Simple countermeasures exist for all types of side-channel attacks.

An additional type of elliptic curve attack is the twist-security attack or fault attack. Such attacks may include invalid-curve attacks and small-subgroup attacks, and they may result in the private key of the victim leaking out. Twist-security attacks are typically simply mitigated with careful parameter validation and curve choices.

Although there are certain ways to attack ECC, the advantages of elliptic curve cryptography for wireless security mean it remains a more secure option.

What Is an Elliptic Curve Digital Signature?

An Elliptic Curve Digital Signature Algorithm (ECDSA) uses ECC keys to ensure each user is unique and every transaction is secure. Although this kind of digital signing algorithm (DSA) offers a functionally indistinguishable outcome as other DSAs, it uses the smaller keys you'd expect from ECC and therefore is more efficient.

What is Elliptic Curve Cryptography Used For?

ECC is among the most commonly used implementation techniques for digital signatures in cryptocurrencies. Both Bitcoin and Ethereum apply the Elliptic Curve Digital Signature Algorithm (ECDSA) specifically in signing transactions. However, ECC is not used only in cryptocurrencies. It is a standard for encryption that will be used by most web applications going forward due to its shorter key length and efficiency.

ECDH Key Exchange

The ECDH (Elliptic Curve Diffie–Hellman Key Exchange) is anonymous key agreement scheme, which allows two parties, each having an elliptic-curve public–private key pair, to establish a shared secret over an insecure channel. ECDH is very similar to the classical DHKE (Diffie–Hellman Key Exchange) algorithm, but it uses ECC point multiplication instead of modular exponentiations. ECDH is based on the following property of EC points:

(a * G) * b = (b * G) * a

If we have two secret numbers a and b (two private keys, belonging to Alice and Bob) and an ECC elliptic curve with generator point G, we can exchange over an insecure channel the values (a * G) and (b * G) (the public keys of Alice and Bob) and then we can derive a shared secret: secret = (a * G) * b = (b * G) * a. Pretty simple. The above equation takes the following form:

alicePubKey * bobPrivKey = bobPubKey * alicePrivKey = secret

The ECDH algorithm (Elliptic Curve Diffie–Hellman Key Exchange) is trivial:

Alice generates a random ECC key pair: {alicePrivKey, alicePubKey = alicePrivKey * G}

Bob generates a random ECC key pair: {bobPrivKey, bobPubKey = bobPrivKey * G}

Alice and Bob exchange their public keys through the insecure channel (e.g. over Internet)

Alice calculates sharedKey = bobPubKey * alicePrivKey

Bob calculates sharedKey = alicePubKey * bobPrivKey

Now both Alice and Bob have the same sharedKey == bobPubKey * alicePrivKey == alicePubKey * bobPrivKey

RSA and ElGamal Schemes – A Comparison

Let us briefly compare the RSA and ElGamal schemes on the various aspects.

RSA	ElGamal
It is more efficient for encryption.	It is more efficient for decryption.
It is less efficient for decryption.	It is more efficient for decryption.
For a particular security level, lengthy keys are required in RSA.	For the same level of security, very short keys are required.
It is widely accepted and used.	It is new and not very popular in market.

Chinese Remainder Theorem | Set 1 (Introduction)

We are given two arrays num[0..k-1] and rem[0..k-1]. In num[0..k-1], every pair is coprime (gcd for every pair is 1). We need to find minimum positive number x such that:

x % num[0] = rem[0],

x % num[1] = rem[1],

.....

x % num[k-1] = rem[k-1]

Basically, we are given k numbers which are pairwise coprime, and given remainders of these numbers when an unknown number x is divided by them. We need to find the minimum possible value of x that produces given remainders.

Examples :

Input: num[] = {5, 7}, rem[] = {1, 3}

Output: 31

Explanation:

31 is the smallest number such that:

(1) When we divide it by 5, we get remainder 1.

(2) When we divide it by 7, we get remainder 3.

Input: num[] = {3, 4, 5}, rem[] = {2, 3, 1}

Output: 11

Explanation:

11 is the smallest number such that:

(1) When we divide it by 3, we get remainder 2.

(2) When we divide it by 4, we get remainder 3.

(3) When we divide it by 5, we get remainder 1.

Chinese Remainder Theorem states that there always exists an x that satisfies given congruences. Below is theorem statement adapted from wikipedia.
Let num[0], num[1], …num[k-1] be positive integers that are pairwise coprime. Then, for any given sequence of integers rem[0], rem[1], … rem[k-1], there exists an integer x solving the following system of simultaneous congruences.

$$\begin{cases} x \equiv rem[0] & (\text{mod } num[0]) \\ \quad \dots & \\ x \equiv rem[k-1] & (\text{mod } num[k-1]) \end{cases}$$

Furthermore, all solutions x of this system are congruent modulo the product, prod = num[0] * num[1] * ... * nun[k-1]. Hence

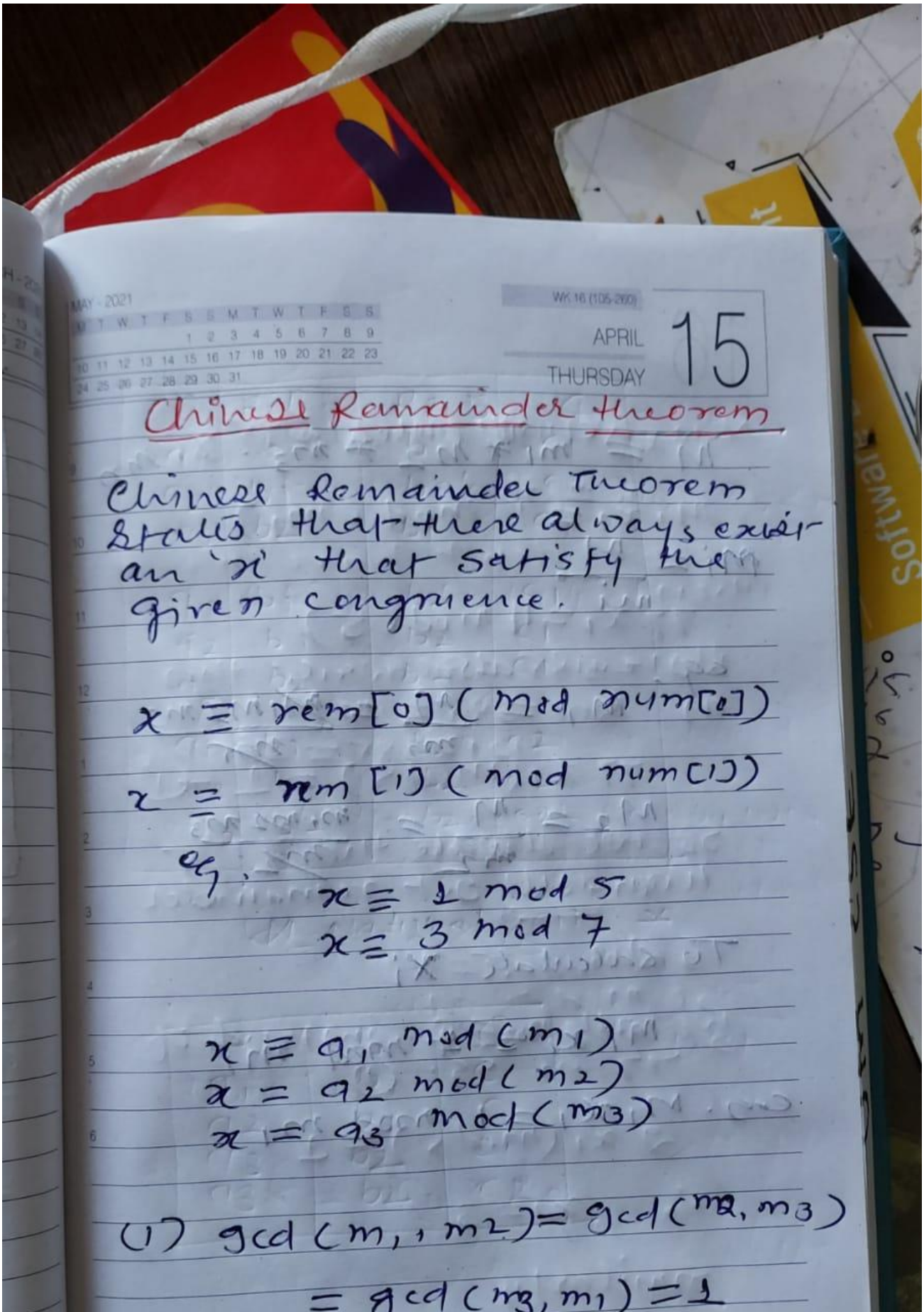
$$x \equiv y \ (\text{mod } num[i]), \quad 0 \leq i \leq k-1 \quad \Longleftrightarrow \quad x \equiv y \ (\text{mod } prod).$$

The first part is clear that there exists an x. The second part basically states that all solutions (including the minimum one) produce the same remainder when divided by-product of n[0], num[1], .. num[k-1]. In the above example, the product is 3*4*5 = 60. And 11 is one solution, other solutions are 71, 131, .. etc. All these solutions produce the same remainder when divided by 60, i.e., they are of form 11 + m*60 where m >= 0.

A **Naive Approach to find x** is to start with 1 and one by one increment it and check if dividing it with given elements in num[] produces corresponding remainders in rem[]. Once we find such an x, we return

it.

X mod 5=1 or x=1 mod(5)



MAY - 2021

APRIL 15 THURSDAY

Chinese Remainder theorem

Chinese Remainder Theorem states that there always exist an 'x' that satisfy the given congruence.

$x \equiv rem[0] \pmod{num[0]}$

$x \equiv rem[1] \pmod{num[1]}$

eg.

$x \equiv 1 \bmod 5$

$x \equiv 3 \bmod 7$

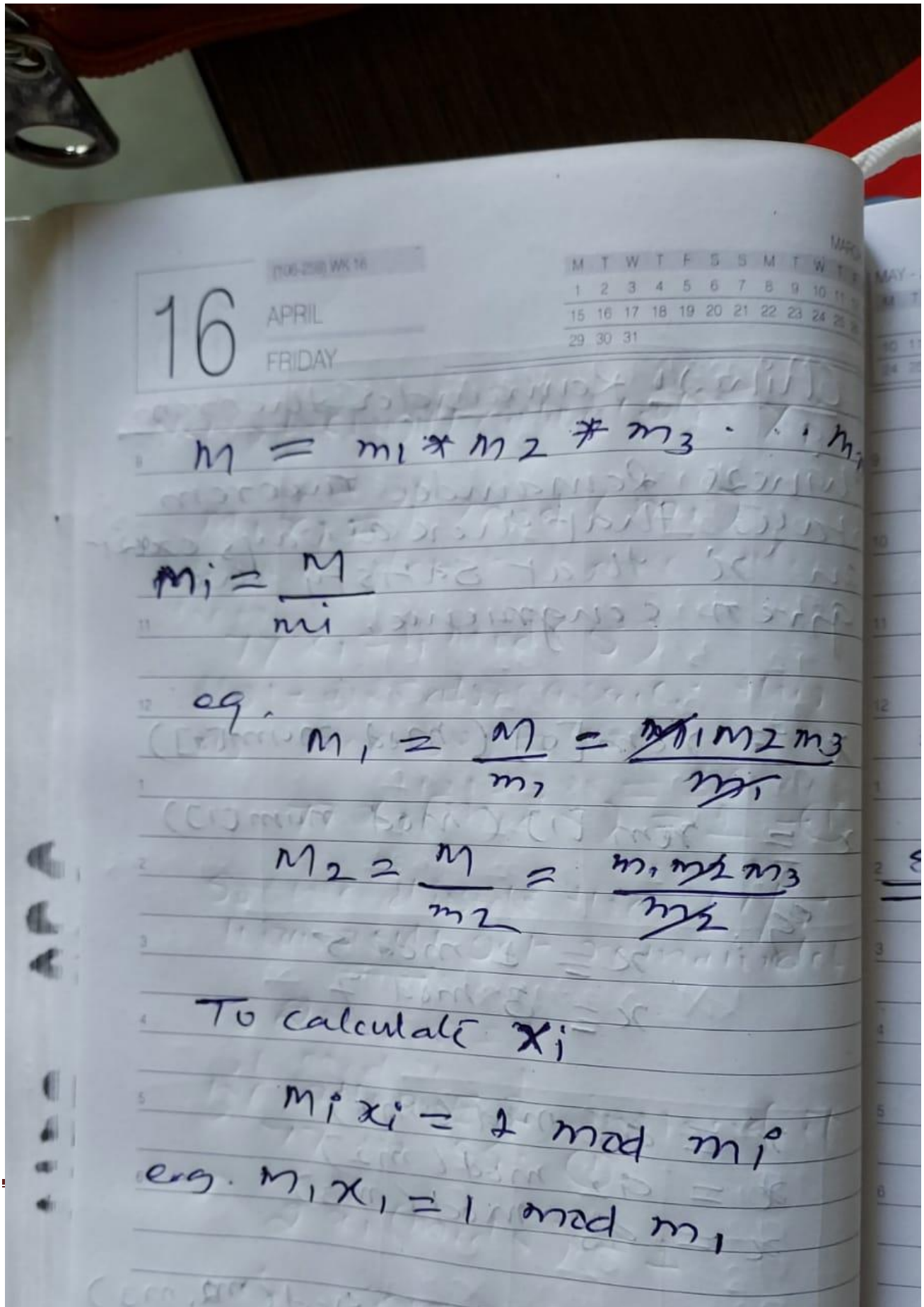
$x \equiv a_1 \bmod (m_1)$

$x = a_2 \bmod (m_2)$

$x \equiv a_3 \bmod (m_3)$

(1) $\gcd(m_1, m_2) = \gcd(m_2, m_3)$

$= \gcd(m_3, m_1) = 1$



16 APRIL FRIDAY

$M = m_1 * m_2 * m_3 \ldots m_i$

$M_i = \frac{M}{m_i}$

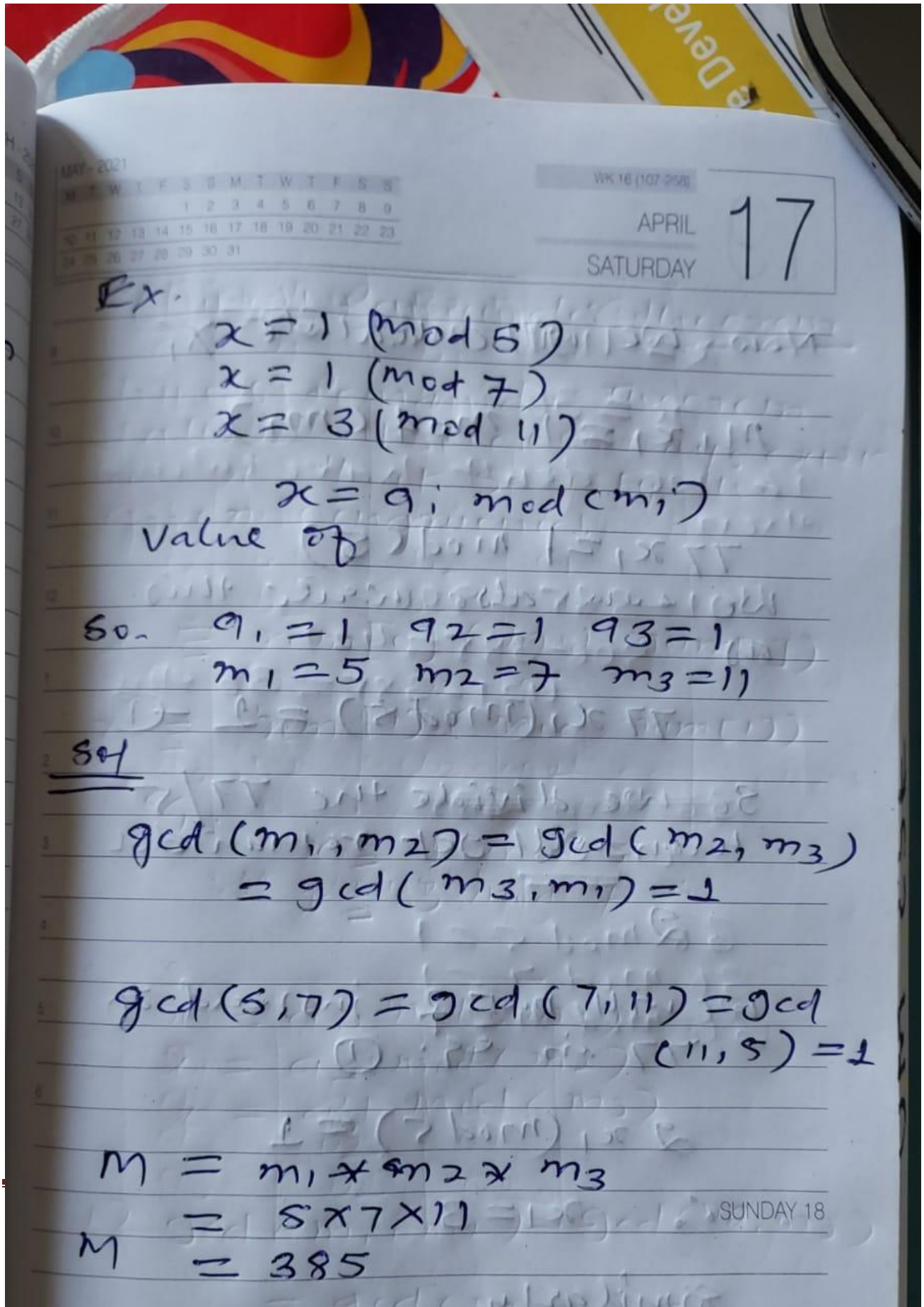
eg. $M_1 = \frac{M}{m_1} = \frac{m_1 m_2 m_3}{m_1}$

$M_2 = \frac{M}{m_2} = \frac{m_1 m_2 m_3}{m_2}$

To calculate x_i

$M_i x_i = 1 \bmod m_i$

eg. $M_1 x_1 = 1 \bmod m_1$



APRIL 17
SATURDAY

Ex.

$x = 1 \pmod 5$
$x = 1 \pmod 7$
$x = 3 \pmod{11}$

$x = a_i \bmod (m_i)$

Value of

So. $a_1 = 1 \quad a_2 = 1 \quad a_3 = 1$
$m_1 = 5 \quad m_2 = 7 \quad m_3 = 11$

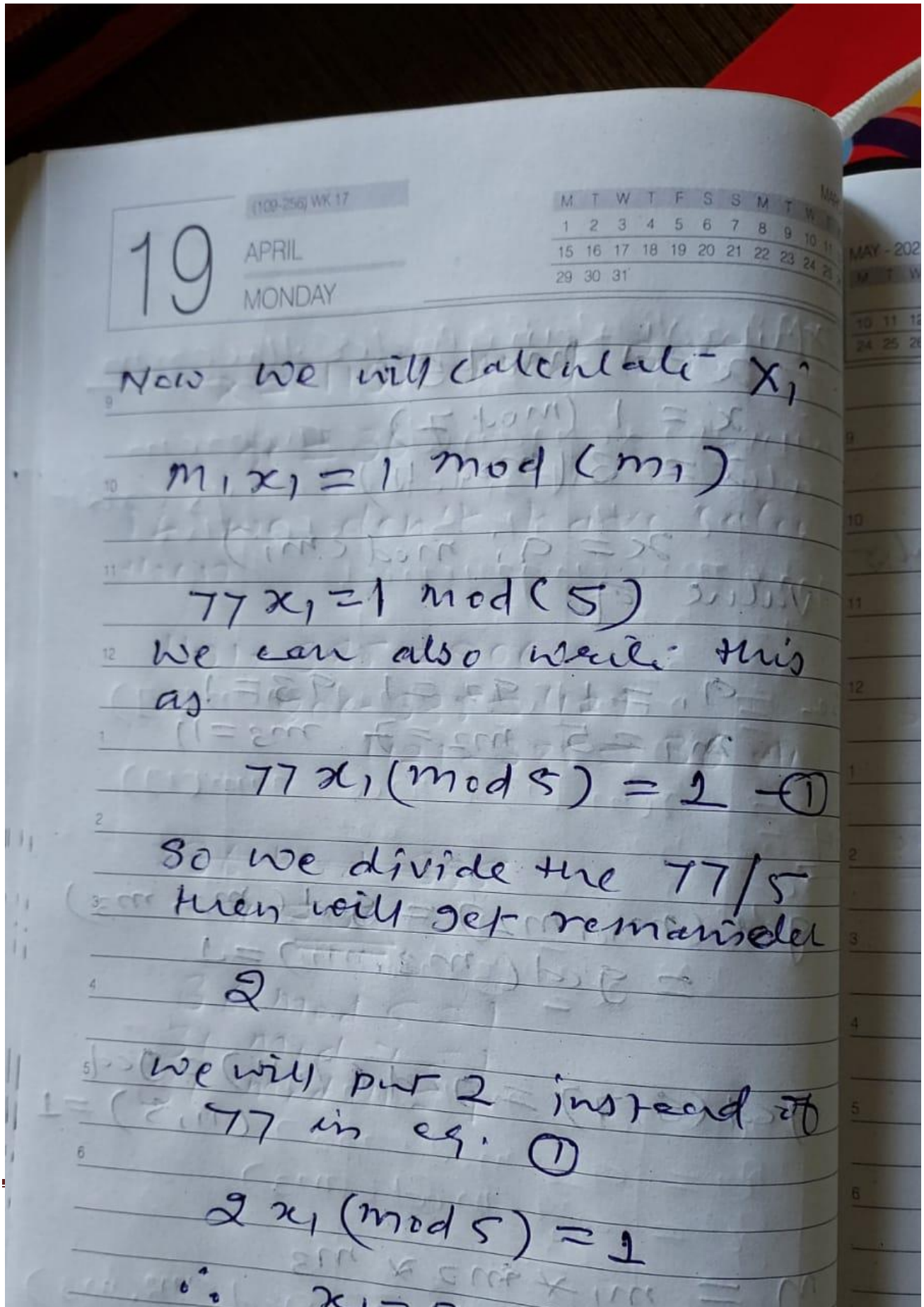
Sol

$\gcd(m_1, m_2) = \gcd(m_2, m_3)$
$= \gcd(m_3, m_1) = 1$

$\gcd(5,7) = \gcd(7,11) = \gcd(11,5) = 1$

$M = m_1 \times m_2 \times m_3$
$= 5 \times 7 \times 11$
$M = 385$

SUNDAY 18



19 APRIL MONDAY

Now we will calculate x_1

$m_1 x_1 = 1 \mod (m_1)$

$77 x_1 = 1 \mod (5)$

We can also write this as

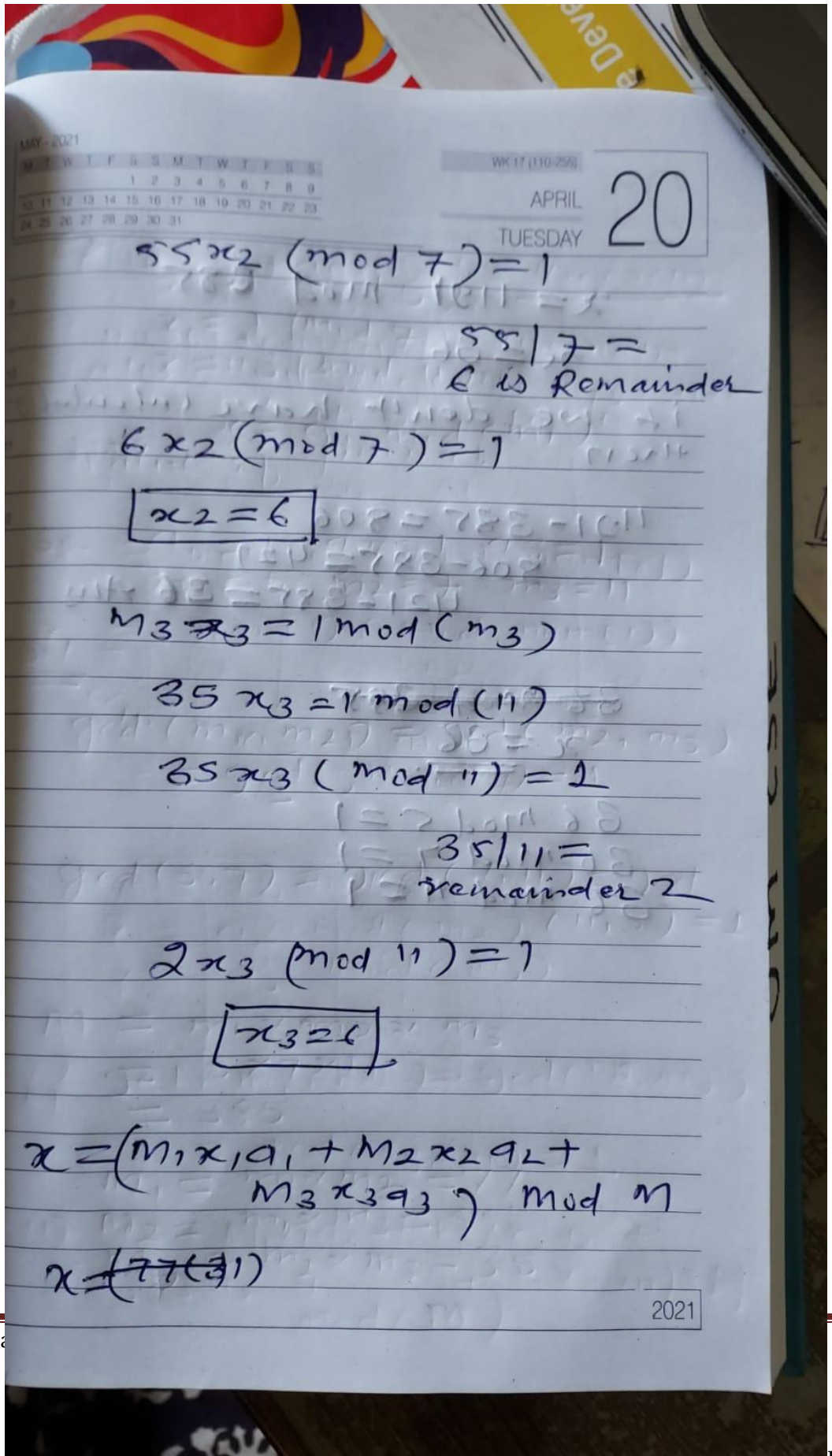
$77 x_1 \pmod 5 = 2$ — (1)

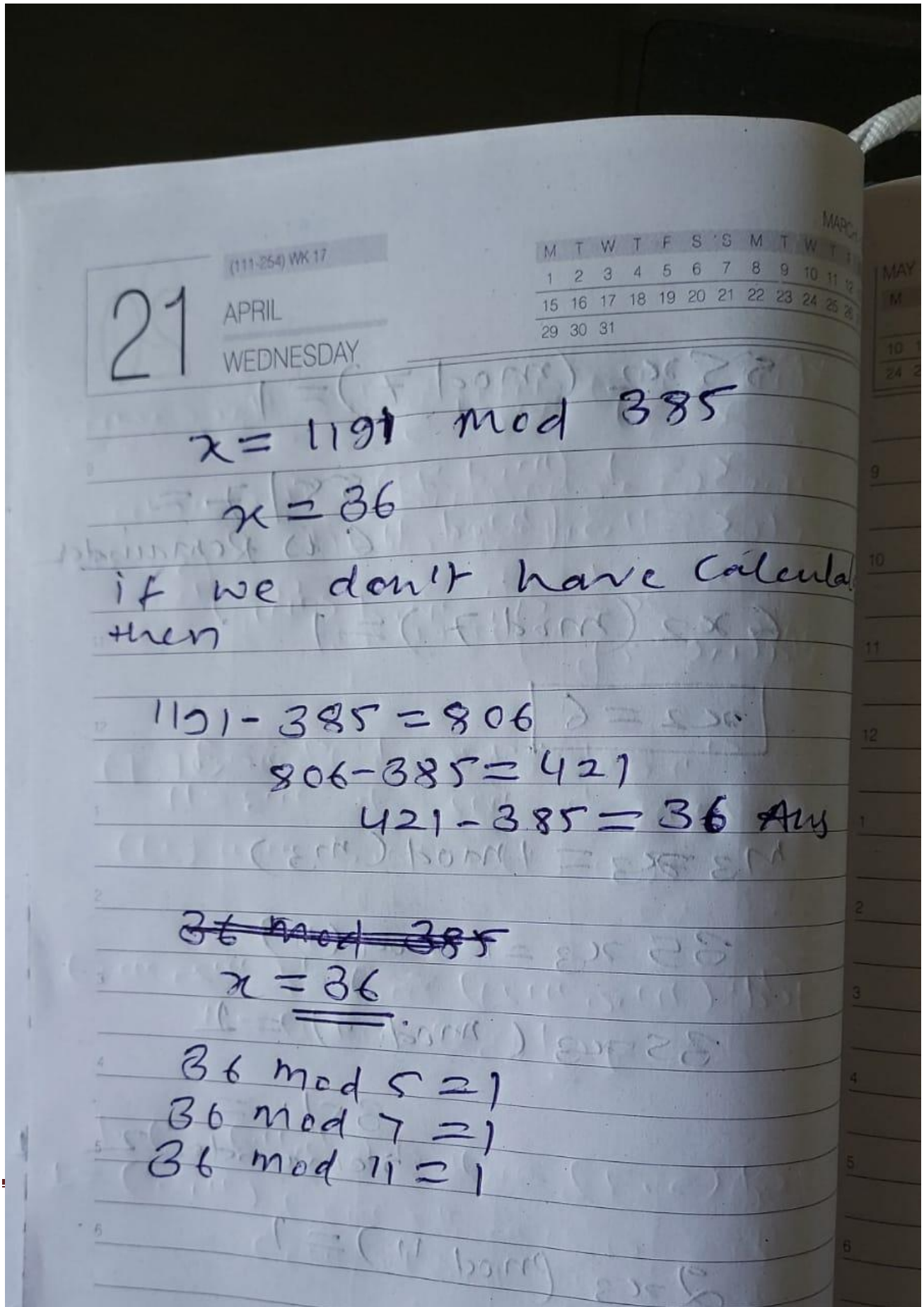
So we divide the 77/5 then will get remainder 2

We will put 2 instead of 77 in eq. (1)

$2 x_1 \pmod 5 = 1$

$\therefore$


APRIL
TUESDAY
20
55x2 (mod 7)=1
55|7 =
6 is Remainder
6x2 (mod 7)=1
x2=6
M3x3=1 mod(m3)
35x3=1 mod(11)
35x3 (mod 11)=1
35|11=
remainder 2
2x3 (mod 11)=1
x3=6
x=(M1x1a1+M2x2a2+
M3x3a3) mod M
2021


21
APRIL
WEDNESDAY
x = 1191 mod 385
x = 36
if we don't have calculator then
1191 - 385 = 806
806 - 385 = 421
421 - 385 = 36 Ans
36 mod 385
x = 36
36 mod 5 = 1
36 mod 7 = 1
36 mod 11 = 1