UNIT 3 NETWORK SECURITY-I

3.0 INTRODUCTION

Cryptography plays an important role in network security. The purpose of cryptography is to protect transmitted information from being read and understood by anyone except the intended recipient. In the ideal sense, unauthorised individuals can never read an enciphered message.

Secret writing can be traced back to 3,000 B.C. when it was used by the Egyptians. They used hieroglyphics to conceal writing from unintended recipients. Hieroglyphics is derived from the Greek word hieroglyphica, which means “sacred carvings”. Hieroglyphics evolved into hieratic, which was easier to use script. Around 400 B.C., military cryptography was employed by the Spartans in the form of strip of papyrus or parchment wrapped around a wooden rod. This system is called a Scytale. The message to be encoded was written lengthwise down the rod on the wrapped material. Then, the material was unwrapped and carried to the recipient. In unwrapped form, the writing appeared to random characters, and to read again the material was rewound on a rod of the same diameter and length.

During 50 B.C., Julius Caesar, the emperor of Rome, used a substitution cipher to transmit messages to Marcus Tullius Cicero. In this, letters of the alphabets are substituted for other letters of the same alphabet. Since, only one alphabet was used, this is also called monoalphabetic substitution. This particular cipher involved shifting the alphabet by three letters and substituting those letters. This is sometimes known as C3 (for Caesar shifting three places).

In this unit, we provide details of cryptography and its needs. In section 3.2 we define cryptography, Section 3.3 presents a brief review of symmetric cryptography. In section 3.4 we discuss the Asymmetric Cryptography. We summarise the unit in section 3.6 followed by the Solutions/Answers.

3.1 OBJECTIVES

After going through this unit, you should be able to:

- learn about the principles and practice of cryptography, and
- various symmetric and public key algorithms.

3.2 CRYPTOGRAPHY

Cryptography is the science of writing in secret code and is an ancient art; the first documented use of cryptography in writing dates back to circa 1900 B.C. when an Egyptian scribe used non-standard hieroglyphics in an inscription. Some experts argue that cryptography appeared simultaneously sometime after writing was invented, with applications ranging from diplomatic missives to war-time battle plans. The new form of cryptography emerges with the widespread development of computer and communications technologies. Cryptography is a key technology when communicating over any un-trusted medium, particularly the Internet. Rivest defines cryptography as simply "Communication in the presence of adversaries". Modern cryptographic techniques have many uses, such as access control, digital signatures, e-mail security, password authentication, data integrity check, and digital evidence collection and copyright protection.

Cryptographic systems are generically classified among three independent dimensions:

- **Types of Operation:** All encryption algorithms are based on two general principles: substitution and transposition. The fundamental requirements are that no information is lost and all operations are reversible.

- **Key Used:** If both sender and the receiver use the same key, then it is referred to as symmetric, single-key, secret-key, or conventional encryption. And if the encryption and decryption key are different, the system is asymmetric key, two-key, or public key encryption.

Within the context of any application-to-application communication, there are some specific security requirements: (1) Authentication, (2) Privacy/confidentiality, (3) Integrity, and (4) Non-repudiation. There are, in general, three types of cryptographic schemes typically used to accomplish these goals: (1) secret key (or symmetric) cryptography, (2) public key (or asymmetric) cryptography, and (3) hash functions, each of which is described in the subsequent subsections.

☞ Check Your Progress 1

1) What is cryptography?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

2) List the various requirements of application-to-application communication.

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

3) List various types of cryptographic techniques.

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

3.3 SYMMETRIC KEY CRYPTOGRAPHY

About twenty years ago, cryptography was the art of making and breaking codes. The codes were used to transfer messages over an unsecured channel. The channel should be protected from intruders who read, insert, delete or modify messages, as depicted in *Figure 1*. The transmission of a message is done using an encryption function E that converts the message or plaintext using the key into a cipher text. The receiver does the reverse of this operation, using the decryption function D and a decryption key to recover the plaintext from the cipher text. The key is distributed in advance over a secure channel, for example by courier.

Normally, the encryption and decryption function, but not the key, are considered known to the adversary, so the protection of the information depends on the key only. If the enemy knows the key, the whole system is useless until a new key is distributed. In secret Key Cryptography, a single key is used for both encryption and decryption, as shown in *Figure 1.* The main difficulty with this approach is the distribution of the key.

Secret key cryptography schemes are generally categorised as being either stream ciphers or block ciphers. Stream ciphers operate on a single byte (or computer word) at a time, and implement some form of feedback mechanism so that the key is constantly changing. Block cipher scheme encrypts one block of data at a time using the same key on each block.

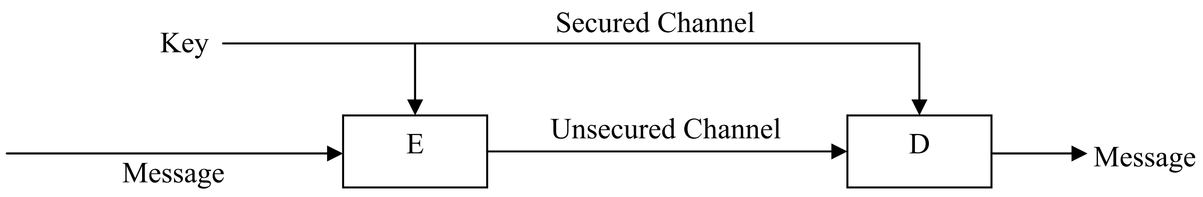


Figure 1: Secret Key Cryptography

The most common secret key cryptography scheme is Data Encryption Standard (DES), designed by IBM in 1970s and adopted by the National Bureau of Standard (NBS) of USA [now the National Institute for Standards and Technology (NIST)] in 1977 for commercial and unclassified government applications. DES is a block-cipher employing a 56-bit key operating on 64 bit blocks of data. DES has a complex set of rules and transformations designed basically for fast hardware implementation and slow software implementation. The latter point is significant in today’s context, as the speed of CPU is several orders of magnitude faster than twenty years ago. IBM also proposed a 112-bit key for DES in the 1990s, which was never implemented seriously.

In 1997, NIST initiated the task of developing a new secure cryptosystem for U.S government applications. This effort has resulted in AES (Advanced Encryption Standard). AES became the official successor to DES in December 2001. AES is based on Rjndael Algorithm and employs a 128-, 192-, or 256-bit key.

The following terminology is often used when symmetric ciphers are examined:

Block Ciphers

A block of cipher transforms *n*-bit plaintext blocks to *n*-bit ciphertext blocks on the application of a cipher key *k*. The key is generated at random using various mathematical functions as shown in *Figure 2*.

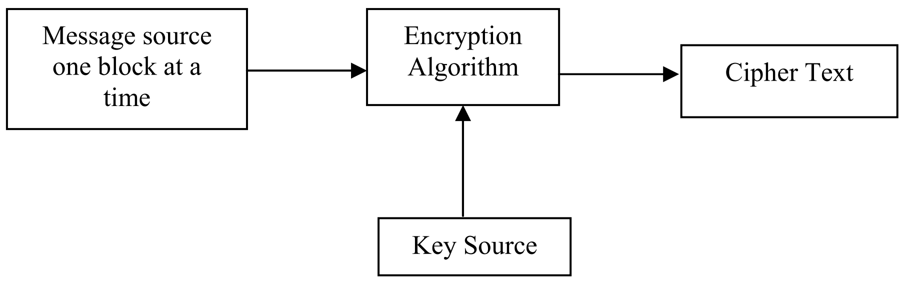


Figure 2: Block cipher

Stream Ciphers

A stream cipher consists of a state machine that outputs at each state transition one bit of information. This stream of output bits is called the *running key*. Just XORring the running key to the plaintext message can implement the encryption. The state machine is a pseudo-random number generator. For example, we can build one from a block cipher by encrypting repeatedly its own output. Typically, more elaborate constructions (for high speed) are used for stream ciphers. Some of the more well-known stream ciphers are RC4 and SEAL. Several stream ciphers are based on linear-feedback shift registers (LFSR) (*Figure 3*), such as A5/1 used in the GSM. These have the benefit of being very fast (several times faster than usual block ciphers).

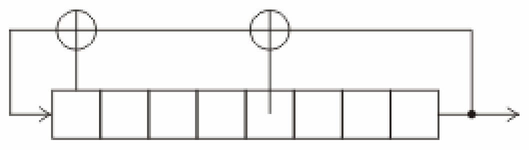


Figure 3: Stream cipher linear feedback shift register (LFSR)

SSSC (Self-Synchronising Stream Ciphers)

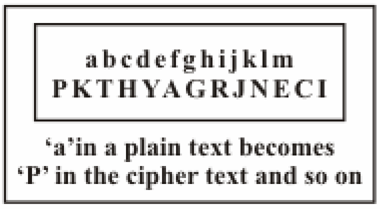
The class of *self-synchronsing stream ciphers* has property that it corrects the output stream after the bit flips or even drops bits after a short time.
SSSCs can be constructed using block ciphers in a CFB-related way. Assume, that we have already encrypted *n* bits of the message and know that much of the ciphertext (where *n* denotes the block length of the cipher). Then we produce a new running key bit by encrypting the *n* ciphertext bits. Take one bit of the output of the cipher to be the new running key bit. Now moving one bit further we can iterate this procedure for the whole length of the message.

One bit error in a ciphertext cannot affect the decrypted plaintext after *n* bits. This makes the cipher self-synchronising.

The block cipher used should have sufficiently large block size to avoid substitution attacks.

Substitution and Transposition

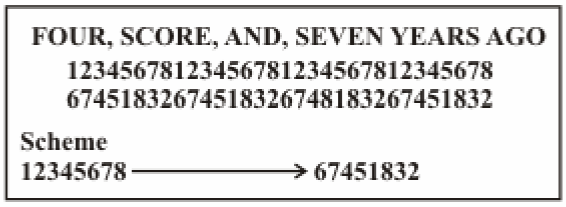
A substitution (*Figure 4*) means replacing symbols or group of symbols by other symbols or groups of symbols. Transposition (*Figure 5*), on the other hand, means permuting the symbols in a block. Neither of these operations alone provide sufficient security, but strong ciphers can be built by combining them. A Substitution-Permutation Network (SPN) means a cipher composed of a number of stages, each involving substitutions and permutations. The prominent examples of SPNs are DES and CAST-128.



a b c d e f g h i j k l m
P K T H Y A G R J N E C I

'a'in a plain text becomes
'P' in the cipher text and so on

Figure 4: Substitution



```
FOUR, SCORE, AND, SEVEN YEARS AGO
  12345678123456781234567812345678
  67451832674518326748183267451832

Scheme
12345678 ——————————> 67451832
```

Figure 5: Transposition

S-Boxes

Lookup tables that map *n* bits to *m* bits, where *n* and *m* are often equal. There are several ways of constructing and measuring good S-boxes for ciphers.

a) S-boxes, which are resistant against well known attacks, can be created using rigorous mathematical approach by applying bent functions (or related).

b) Other designers apply heuristic approaches that result in S-boxes that are more difficult to handle in mathematical proofs, but can have additional benefits.

The S-box may even be the only non-linear part of the cipher (e.g., DES) and thus, may be considered as the single most important part of the cipher. In fact, DES's S-boxes are so good that it is used in many other cipher design (for example, Serpent).

Feistel Networks

The original idea was used in the block cipher, Lucifer, invented by **Horst Feistel.** Several variations have been devised from the original version. A **Feistel** network (*Figure 6*) is a general way of constructing block ciphers from simple functions. The standard **Feistel** network takes a function from *n* bits to *n* bits and produces an invertible function from *2n* bits to *2n* bits. The structure of **Fiestel** network is based on round function. The essential property of **Feistel** networks that makes them so useful in cipher design is that the round function need not be invertible, but always is the resulting function. If the round function depends on, say, *k* bits of a key, then the **Feistel** cipher requires *rk* bits of the key where *r* is the number of rounds used. The security of the **Feistel** structure is not obvious. It is compulsory that a **Feistel cipher** has enough number of rounds, but just adding more rounds does not always guarantee security.

The process of taking the user key and expanding it into *rk* bits for the **Feistel** rounds is called **key scheduling**. This is often a non-linear operation and hence does not directly provide any information about the actual user key. There are many ciphers that have this basic structure; Lucifer, DES, and Twofish etc.

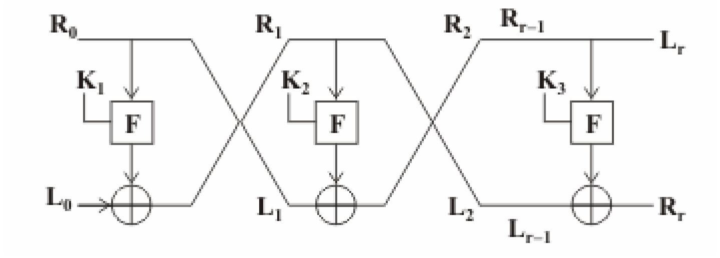


Figure 6: Feistal Cipher

Expansion, Permutation

They are linear operations, and thus, not sufficient to guarantee security. They are common tools in mixing bits in a round function and when used with good non-linear S-boxes (as in DES) they are vital for the security because they propagate the non-linearity uniformly over all bits.

Bitslice Operations (Bitwise Logic Operations XOR, AND, OR, NOT and Bit Permutations)

The idea of bitslice implementations of block ciphers is due to **Eli Biham.** It is common in vector machines to achieve parallel operation. However, **Biham** applied it on serial machines by using large registers as available in modern computers. The term "bitslice" has been the contribution of Matthew Kwan.

All block ciphers can be designed in bitslice manner, but this could affect the speed of operations such as addition and multiplication they may become very slow. On the other hand, permutations are almost free as they only require a *renaming* of the registers and this can be done at the coding level. Thus, for example, in DES exhaustive key search using bitslice techniques, one can increment the current key in, a fraction of a time than is usually needed for key scheduling.

The AES finalist *Serpent* is designed to be implemented using bitslice operations only. This makes it particularly efficient on modern architectures with many registers.

Modes of Operation

Block ciphers are the most commonly used cipher. Block ciphers transform a fixed-size block of data (usually 64 bits) into another fixed-size block (possibly *64* bits wide again) using a function selected by the key. If the key, input block and output block have all *n* bits, a block cipher basically defines a one-to-one mapping from *n*-bit integers to permutations of *n*-bit integers.

If the same block is encrypted twice with the same key, the resulting ciphertext blocks are also the same (this *mode* of encryption is called *electronic code book*, or **ECB**). This information could be useful for an attacker. For identical plaintext blocks being encrypted to different ciphertext blocks, three standard modes are generally used:

1) **CBC (Cipher Block Chaining):** First XORing the plaintext block with the previous ciphertext block obtains a ciphertext block, and then the resulting value needs to be encrypted. In this, leading blocks influence all trailing blocks, which increases the number of plaintext bits one ciphertext bit depends on, but this may also leads to synchronisation problems if one block is lost. The Process of Cipher Block Chaining shown in *Figure 7.* In this Figure m_1, m_2, m_3 are messages and C_1, C_2 and C_3 are ciphertexts.

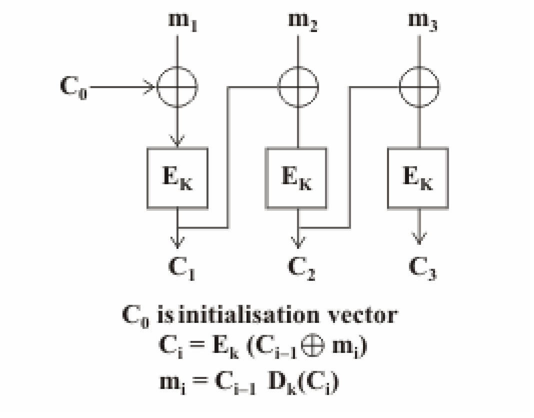


Figure 7: Cipher block chaining

2) **CFB (Cipher Feedback):** In this the *k*th ciphertext block is obtained by encrypting the *(k-1)*th ciphertext block and XORing the result onto the plaintext. The CFB mode possess self-synchronising property: A CFB feedback loop can also be used as a pseudo-random number generator if one simply feeds one block of true random data with trailing blocks of zeroes into the encryption routine (although the expected period of this PRNG would be only about $2^{n/2}$ where *n* is the block size of the cipher). CFB is shown in *Figure 8.*

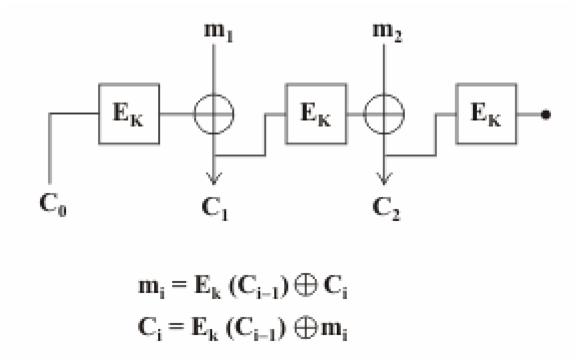


Figure 8: Cipher feedback model

3) **OFB (Output Feedback Block):**It is used as a synchronous key-stream generator, whose output is XORed with the plaintext to obtain ciphertext, block by block. The key-stream is generated by riterative encryption, starting with an initialisation vector (IV) which, along with the key, is agreed upon beforehand. OFB is shown in *Figure 9*.

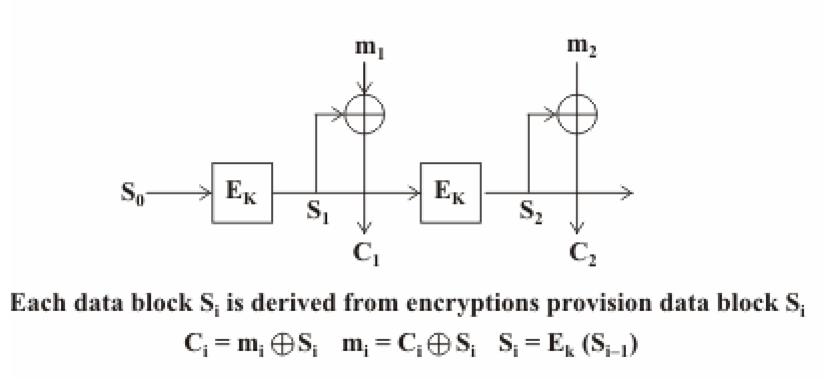

m1
m2
S0
EK
S1
EK
S2
C1
C2
Each data block Si is derived from encryptions provision data block Si
Ci = mi ⊕Si   mi = Ci ⊕ Si   Si = Ek (Si–1)

Figure 9: Output feedback block

The One-Time Pad

The one-time pad (OTP) is the only cipher that has been proven to be unconditionally secure, i.e., unbreakable in practice. Further, it has been proven that any unbreakable, unconditionally secure cipher must be a one-time pad.

The Vernam cipher (invented by **G. Vernam** in 1917) in one time pad and is very simple: take a stream of bits that contain the plaintext message, and a key, which is the secret random bit-stream of the same length as the plaintext. [To encrypt the plaintext with the key, sequentially exclusive-or each pair of key bit and plaintext bit to obtain the ciphertext bit]. If the key is truly random, the attacker or adversary has no means of deciding whether some guessed plaintext is more likely than any other when, only the ciphertext and no knowledge of the plaintext is available.

The main practical problem is that the key does not have a small constant length, but the same length as the message, and one part of a key should never be used twice (or the cipher can be broken). However, this cipher has allegedly been in widespread use since its invention, and even more since the security proof by **C. Shannon** in 1949. Although, admittedly the security of this cipher had been conjectured earlier, it was **Shannon** who actually found a formal proof for it.

DES

The Data Encryption Standard (**DES**) is an algorithm developed in the mid-1970s and turned into a standard by the US National Institute of Standards and Technology (NIST), and was also adopted by several other governments worldwide. It was and still is widely used, especially in the financial industry.

DES is a block cipher with a 64-bit block size. It uses 56-bit keys. This makes it suspectible to exhaustive key search with modern computing powers and special-purpose hardware. DES is still strong enough to keep most random hackers, adversaries and individuals out, DES is easily breakable with special hardware by, government, criminal organizations etc. DES is getting too weak, and should not be used in new applications. NIST proposed in 2004 to withdraw the DES standard.

A variant of DES, Triple-DES (also **3DES**) is based on using DES three times (normally in an encrypt-decrypt-encrypt sequence with three different, unrelated keys). The Triple-DES is arguably much stronger than (single) DES, however, it is rather slow compared to some new block ciphers.

Although, at the time of DES's introduction, its design philosophy was held secret. Some information has been published about its design, and one of the original designers, **Don Coppersmith**, has said that they discovered ideas similar to differential crypt analysis while designing DES in 1974. However, it was just a matter of time that these fundamental ideas were re-discovered.

Although DES is no longer considered a practical solution, it is many a time used to describe new crypt analytical methods. Even today, there are no crypt analytical techniques that would completely break DES in a structural way; indeed, the only real weakness known is the short key size (and perhaps the small block size).

DES uses the:

- Data Encryption Algorithm (DEA),
- a secret key block-cipher employing a 56-bit key operating on 64-bit blocks.
- FIPS 81 describes four modes of DES operation: Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), and Output Feedback (OFB). Despite all these options, ECB is the most commonly deployed mode of operation.

DES uses a 56-bit key, which is divided into eight 7-bit blocks and an 8th odd parity bit is added to each block (i.e., a "0" or "1" is added to the block so that there are an odd number of 1 bit in each 8-bit block). By using the 8 parity bits for rudimentary error detection, a DES key is actually 64 bits in length for computational purposes (although it only has 56 bits worth of randomness, or *entropy*).

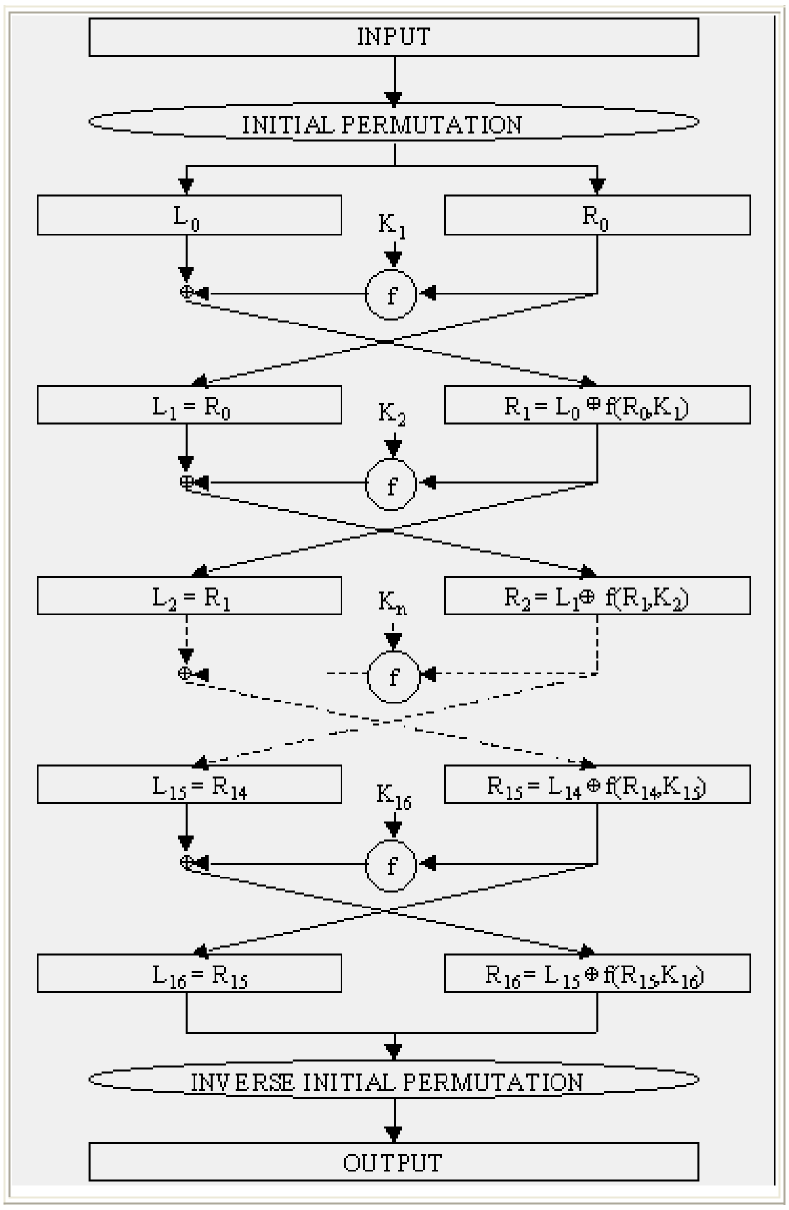

INPUT
INITIAL PERMUTATION
L_0
K_1
R_0
f
$L_1 = R_0$
K_2
$R_1 = L_0 \oplus f(R_0, K_1)$
f
$L_2 = R_1$
K_n
$R_2 = L_1 \oplus f(R_1, K_2)$
f
$L_{15} = R_{14}$
K_{16}
$R_{15} = L_{14} \oplus f(R_{14}, K_{15})$
f
$L_{16} = R_{15}$
$R_{16} = L_{15} \oplus f(R_{15}, K_{16})$
INVERSE INITIAL PERMUTATION
OUTPUT

Figure 10: DES algorithm

DES process on 64-bit blocks of the plaintext, invoking 16 rounds of permutations, swaps, and substitutes, as shown in *Figure 10*. The main DES steps are:

1) The 64-bit block undergoes an initial permutation (IP), where each bit is moved to a new bit position; e.g., the 1st, 2nd, and 3rd bits are moved to the 58th, 50th, and 42nd position, respectively.
2) The permuted 64-bit input is divided into two 32-bit blocks, called *left* and *right*, respectively. The initial values of the left and right blocks are denoted as L_0 and R_0.

3) There are then 16 rounds of operation on the L and R blocks.

During each iteration (where *n* ranges from 1 to 16), the following formula is used:

$$L_n = R_{n-1}$$
$$R_n = L_{n-1} \text{ XOR } f(R_{n-1},K_n)$$

The new L block value is taken from the prior R block value. The new R block is calculated by taking the bit-by-bit exclusive-OR (XOR) of the prior L block with the results of applying the DES cipher function, *f*, to the prior R block and K_n. (K_n is a 48-bit value derived from the 64-bit DES key). Each round uses a different 48 bits according to the standard's Key Schedule algorithm.

The cipher function, f, combines the 32-bit R block value and the 48-bit subkey in the following way:

- First, the 32 bits in the R block are expanded to 48 bits by an expansion function (E); the extra 16 bits are found by repeating the bits in 16 predefined positions.
- The 48-bit expanded R-block is then ORed with the 48-bit subkey.
- The result is a 48-bit value that is then divided into eight 6-bit blocks.
- These are fed as input into 8 selection (S) boxes, denoted $S_1,...,S_8$. Each 6-bit input yields a 4-bit output using a table lookup based on the 64 possible inputs; this results in a 32-bit output from the S-box. The 32 bits are then rearranged by a permutation function (P), producing the results from the cipher function.

The results from the final DES round — i.e., L_{16} and R_{16} — are recombined into a 64-bit value and fed into an inverse initial permutation (IP^{-1}). At this step, the bits are rearranged into their original positions, so that the 58th, 50th, and 42nd bits, for example, are moved back into the 1st, 2nd, and 3rd positions, respectively. The output from IP^{-1} is the 64-bit ciphertext block.

Breaking DES

DES's 56-bit key was too short to withstand a brute-force attack from computing power of modern computers. Remember Moore's Law: computer power doubles every 18 months. Keeping this law in mind, a key that could withstand a brute-force guessing attack in 2000 could hardly be expected to withstand the same attack a quarter of a century later.

DES is even more vulnerable to a brute-force attack as it is commonly used to encrypt words, meaning that the entropy of the 64-bit block is greatly reduced. That is, if we are encrypting random bit streams, then a given byte might contain any one of 2^8 (256) possible values and the entire 64-bit block has 2^{64}, or about 18.5 quintillion, possible values. If we are encrypting words, however, we are most likely to find a limited set of bit patterns; perhaps 70 or so if we account for upper and lower case letters, the numbers, space, and some punctuation. That is, only about ¼ of the bit combinations of a given byte are likely to occur. Despite this, the U.S. government insisted throughout the mid-1990s that 56-bit DES was secure and virtually unbreakable if appropriate precautions were employed. In response, RSA Laboratories sponsored a series of cryptographic challenges to prove that DES was no longer secure and appropriate for use.

- **DES Challenge I** was launched in March 1997. R. Verser completed it in 84 days in a collaborative effort to break DES using thousands of computers on the Internet. The first DES II challenge lasted 40 days during early 1998. This problem was solved by distributed.net, a worldwide distributed computing network using the spare CPU cycles of computers around the Internet. The participants in distributed.net's activities load a client program that runs in the background, conceptually similar to the SETI @Home "Search for Extraterrestrial Intelligence" project). By the end of the project, distributed.net systems were checking 28 *billion* keys per second.

- The second **DES II challenge** lasted just less than 3 days. On July 17, 1998, the Electronic Frontier Foundation (EFF) announced the construction of a hardware that could brute-force a DES key in an average of 4.5 days. The device called Deep Crack, could check 90 billion keys per second and cost only about $220,000 including design. As the design is scalable, this suggests that an organisation could develop a DES cracker that could break 56-bit keys in an average of a day for as little as $1,000,000.
- The **DES III challenge**, launched in January 1999, was broken in less than a day by the coordinated efforts of Deep Crack and distributed.net.

The Deep Crack algorithm is to launch a brute-force attack by guessing every possible key. A 56-bit key yields 2^{56}, or about 72 quadrillion, possible values. The DES cracker team initially assumed that *some* recognisable plaintext would appear in the decrypted string even though they didn't have a specific known plaintext block. They then applied all 2^{56} possible key values to the 64-bit block. The system checked to find if the decrypted value of the block was "interesting", which they defined as bytes containing one of the alphanumeric characters, space, or some punctuation. As the likelihood of a single byte being "interesting" is about ¼, then the likelihood of the entire 8-byte stream being "interesting" is about $¼^8$, or 1/65536 ($½^{16}$). This dropped the number of possible keys that might yield positive results to about 2^{40}, or about a trillion.

After the assumption that an "interesting" 8-byte block would be followed by another "interesting" block. Therefore, if the first block of ciphertext decrypted to something interesting, they decrypted the next block; otherwise, they abandoned this key. Only if the second block was also "interesting" did they examine the key closer. Looking for 16 consecutive bytes that were "interesting" meant that only 2^{24}, or 16 million, keys required to be examined further. This further examination was primarily to see if the text made any sense.

It is important to mention mentioning a couple of forms of crypt analysis that have been shown to be effective against DES. *Differential cryptanalysis*, invented in 1990 by **E. Biham** and **A. Shamir** (of RSA fame), is a chosen-plaintext attack. By selecting pairs of plaintext with particular differences, the crypt analyst examines the differences in the resultant ciphertext pairs. *Linear plaintext*, invented by **M. Matsui**, uses a linear approximation to analyse the actions of a block cipher (including DES). Both these attacks one be more efficient than brute force.

DES Variants

After DES algorithm was "officially" broken, several variants appeared. In the early 1990s, there was a proposal to increase the security of DES by effectively increasing the key length by using multiple keys with multiple passes etc. But for this scheme to work, it had to first be shown that, the DES function is not a *group*, as defined in mathematics. If DES was a group, then we could show that for two DES keys, X1 and X2, applied to some plaintext (P), we can get a single equivalent key, X3, that would provide the same result; i.e.,

$$E_{X2}(E_{X1}(P)) = E_{X3}(P)$$

where $E_X(P)$ represents DES encryption of some plaintext *P* using DES key *X*. If DES were a group, it wouldn't matter how many keys and passes we applied to some plaintext; we could always find a single 56-bit key that would provide the same result.

As DES was proven not to be a group, we apply additional keys and passes to increase the effective key length. One choice, then, might be to use two keys and two passes, yielding an effective key length of 112 bits. This can be called Double-DES. The two keys, Y1 and Y2, might be applied as follows:

$$C = E_{Y2}(E_{Y1}(P))$$
$$P = D_{Y1}(D_{Y2}(C))$$

where $E_Y(P)$ and $D_Y(C)$ represent DES encryption and decryption, respectively, of some plaintext *P* and ciphertext *C*, respectively, using DES key *Y*.

But an interesting attack can be launched against this "Double-DES" scheme. The applications of the formula above can be with the following individual steps (where C' and P' are intermediate results):

$$C' = E_{Y1}(P) \text{ and } C = E_{Y2}(C')$$
$$P' = D_{Y2}(C) \text{ and } P = D_{Y1}(P')$$

Unfortunately, C'=P', which leaves it vulnerable to a simple *known plaintext* attack (or "Meet-in-the-middle") where the attacker or adversary knows some plaintext (P) and its matching ciphertext (C). To obtain C', the attacker or adversary needs to try all 2^{56} possible values of Y1 applied to P; to obtain P', the attacker needs to try all 2^{56} possible values of Y2 applied to C. Since C'=P', the attacker knows when a match has been achieved — after only $2^{56} + 2^{56} = 2^{57}$ key searches, only twice the work of brute-forcing DES.

Triple-DES (3DES) is based upon the Triple Data Encryption Algorithm (TDEA), as described in FIPS 46-3. 3DES, which is not susceptible to a meet-in-the-middle attack, employs three DES passes and one, two, or three keys called K1, K2, and K3. Generation of the ciphertext (C) from a block of plaintext (P) is accomplished by:

$$C = E_{K3}(D_{K2}(E_{K1}(P)))$$

where $E_K(P)$ and $D_K(P)$ represent DES encryption and decryption, respectively, of some plaintext *P* using DES key *K*. This is also sometimes referred to as an *encrypt-decrypt-encrypt mode* operation.

Decryption of the ciphertext into plaintext is accomplished by:

$$P = D_{K1}(E_{K2}(D_{K3}(C)))$$

The use of three, independent 56-bit keys provides 3DES with an effective key length of 168 bits. The specification also defines use of two keys where, in the operations above, K3 = K1; this provides an effective key length of 112 bits. Finally, a third keying option is to use a single key, so that K3 = K2 = K1 (in this case, the effective key length is 56 bits and 3DES applied to some plaintext, P, will yield the same ciphertext, C, as normal DES would with that same key). With the relatively low cost of key storage and the modest increase in processing due to the use of longer keys, the best recommended practices are that 3DES are employed with three keys.

Another variant of DES, called DESX, is the contribution of Ron Rivest. Developed in 1996, DESX is a very simple algorithm that greatly increases DES's resistance to brute-force attacks without increasing its computational complexity. In DESX, the plaintext input is XORed with 64 additional key bits prior to encryption and the output is likewise XORed with the 64 key bits. By adding just two XOR operations, DESX has an effective keylength of 120 bits against an exhaustive key-search attack. It is pertinent to note that DESX is not immune to other types of more sophisticated attacks, such as differential or linear crypt analysis, but brute-force is the primary attack vector on DES.

AES

In response to cryptographic attacks on DES, search for a replacement to DES started in January 1997. In September 1997, a formal Call for Algorithms was initiated and in August 1998 announced that 15 candidate algorithms were being considered (Round 1). In April 1999, NIST announced that the 15 had been filtered down to five finalists (Round 2): *MARS* (multiplication, addition, rotation and substitution) from IBM; **Ronald Rivest's** *RC6*; *Rijndael* from a Belgian team; *Serpent*, developed jointly by a team from England, Israel, and Norway; and *Twofish*, developed by Bruce **Schneier**. In October 2000, NIST announced their selection: Rijndael.

In October 2000, NIST published the Report on the Development of the Advanced Encryption Standard (AES) that compared the five Round 2 algorithms in a number of categories. The table below summarises the relative scores of the five schemes (1=low, 3=high):

	Algorithm				
Category	**MARS**	**RC6**	**Rijndael**	**Serpent**	**Twofish**
General security	3	2	2	3	3
Implementation of security	1	1	3	3	2
Software performance	2	2	3	1	1
Smart card performance	1	1	3	3	2
Hardware performance	1	2	3	3	2
Design features	2	1	2	1	3

NIST selected Rijndael as its performance in hardware and software across a wide range of environments in all possible modes. It has excellent key setup time and has low memory requirements, in addition its operations are easy to defend against power and timing attacks.

In February 2001, NIST published the Draft Federal Information Processing Standard (FIPS) AES Specification for public review and comment. AES contains a subset of Rijndael's capabilities (e.g., AES only supports a 128-bit block size) and uses some slightly different nomenclature and terminology, but to understand one is to understand both. The 90-day comment period ended on May 29, 2001 and the U.S. Department of Commerce officially adopted AES in December 2001, published as FIPS PUB 197.

AES (Rijndael) Overview

Rijndael (pronounced as in "rain doll" or "rhine dahl") is a block cipher designed by **Joan Daemen** and **Vincent Rijmen**, both cryptographers in Belgium. Rijndael can operate over a variable-length block using variable-length keys; the version 2 specification submitted to NIST describes use of a 128-, 192-, or 256-bit key to encrypt data blocks that are 128, 192, or 256 bits long. The block length and/or key length can be extended in multiples of 32 bits and it is specifically designed for efficient implementation in hardware or software on a range of processors. The design of Rijndael was strongly influenced by the block cipher called *Square*, also designed by **Daemen** and **Rijmen.** Rijndael is an iterated block cipher, meaning that the initial input block and cipher key undergoes multiple rounds of transformation before producing the output. Each intermediate cipher result is called *State*.

For simplicity, the block and cipher key are often given as an array of columns where each array has 4 rows and each column represents a single byte (8 bits). The number of columns in an array representing the state or cipher key, then, can be calculated as the block or key length divided by 32 (32 bits = 4 bytes). An array representing a State will have Nb columns, where Nb values of 4, 6, and 8 correspond to a 128-, 192-, and 256-bit block, respectively. Similarly, an array representing a Cipher Key will have Nk columns, where Nk values of 4, 6, and 8 correspond to a 128-, 192-, and 256-bit key, respectively. An example of a 128-bit State (Nb = 4) and 192-bit Cipher Key (Nk = 6) is shown below:

$$\begin{matrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{matrix} \qquad \begin{matrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} & k_{0,4} & k_{0,5} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} & k_{1,4} & k_{1,5} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} & k_{2,4} & k_{2,5} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} & k_{3,4} & k_{3,5} \end{matrix}$$

The number of transformation rounds (Nr) in Rijndael is a function of the block length and key length, and is given in the table below:

Rounds Nr		**Block Size**		
		128 bits Nb = 4	**192 bits Nb = 6**	**256 bits Nb = 8**
Key Size	**128 bits Nk = 4**	10	12	14
	192 bits Nk = 6	12	12	14
	256 bits Nk = 8	14	14	14

The AES version of Rijndael does not support all nine combinations of block and key lengths, but only the subset using a 128-bit block size. NIST calls these supported variants AES-128, AES-192, and AES-256 where the number refers to the key size. The Nb, Nk, and Nr values supported in AES are:

	Parameters		
Variant	**Nb**	**Nk**	**Nr**
AES-128	4	4	10
AES-192	4	6	12
AES-256	4	8	14

The AES/Rijndael cipher itself has three operational stages:

- AddRound Key transformation
- **Nr**-1 Rounds comprising: SubBytes transformation; ShiftRows transformation; MixColumns transformation; AddRoundKey transformation
- A final Round comprising: SubBytes transformation; ShiftRows transformation; AddRoundKey transformation.

The nomenclature used below is taken from the AES specification, although, references to the Rijndael specification are made for completeness. The arrays *s* and *s'* refer to the State before and after a transformation, respectively (NOTE: The Rijndael specification uses the array nomenclature *a* and *b* to refer to the before and after States, respectively). The subscripts *i* and *j* are used to indicate byte locations within the State (or Cipher Key) array.

The SubBytes Transformation

The substitute bytes (called *ByteSub* in Rijndael) transformation operates on each of the State bytes independently and changes the byte value. An S-box, or *substitution table*, controls the transformation. The characteristics of the S-box transformation as well as a compliant S-box table are provided in the AES specification; as an example, an input State byte value of 107 (0x6b) will be replaced with a 127 (0x7f) in the output State and an input value of 8 (0x08) would be replaced with a 48 (0x30).

One simple way to think of the SubBytes transformation is that a given byte in State s is given a new value in State s' according to the S-box. The S-box, then, is a function on a byte in State s so that:

$$s'_{i,j} = \text{S-box}\ (s_{i,j})$$

The more general depiction of this transformation is shown by:

$$\begin{array}{cccc} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{array} \text{ ====> S-box ====> } \begin{array}{cccc} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{array}$$

The ShiftRows Transformation

The shift rows transformation cyclically shifts the bytes in the bottom three rows of the State array. According to the more general Rijndael specification, rows 2, 3, and 4 are cyclically left-shifted by C1, C2, and C3 bytes, respectively, per the table below:

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

The current version of AES, of course, only allows a block size of 128 bits (Nb = 4) so that C1=1, C2=2, and C3=3.

The diagram below shows the effect of the Shift Rows transformation on State s:

$$\begin{array}{cccccccccc} \textbf{State s} &&&& \textbf{----------- no shift ----------->} & \textbf{State s'} &&& \\ s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} & \text{----> left-shift by C1 (1) ---->} & s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} & \text{----> left-shift by C2 (2) ---->} & s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} & \text{----> left-shift by C3 (3) ---->} & s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} & & s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{array}$$

The MixColumns Transformation

The mix columns (called *MixColumn* in Rijndael) transformation uses a mathematical function to transform the values of a given column within a State, acting on the four values at one time as if they represented a four-term polynomial.

$$s'_{i,c} = \text{MixColumns}\ (s_{i,c})$$
$$\text{for } 0<=i<=3 \text{ for some column, c.}$$

The column position does not change, however, the values within the column change.

Round Key Generation and the AddRoundKey Transformation

The Cipher Key is used to derive a different key to be applied to the block during each round of the encryption operation. These keys are known as the Round Keys and each will be the same length as the block, i.e., Nb 32-bit words.

The AES defines a key schedule by which the original Cipher Key (of length Nk 32-bit words) is used to form an *Expanded Key*. The *Expanded Key* size is equal to the block size multiplied by the number of encryption rounds plus 1, which will provide Nr+1 different keys. (Note that there are Nr encipherment rounds but Nr+1

AddRoundKey transformations.). For example, AES uses a 128-bit block and either 10, 12, or 14 iterative rounds depending upon key length. With a 128-bit key, we would need 1408 bits of key material (128×11=1408), or an *Expanded Key* size of 44 32-bit words (44×32=1408). Similarly, a 192-bit key would require 1664 bits of key material (128×13), or 52 32-bit words, while a 256-bit key would require 1920 bits of key material (128×15), or 60 32-bit words. The key expansion mechanism, then, starts with the 128-, 192-, or 256-bit Cipher Key and produces a 1408-, 1664-, or 1920-bit *Expanded Key*, respectively. The original Cipher Key occupies the first portion of the *Expanded Key* and is used to produce the remaining new key material.

The result is an *Expanded Key* that can be 11, 13, or 15 separate keys, each used for one AddRoundKey operation. These, then, are the *Round Keys*. The diagram below shows an example using a 192-bit Cipher Key (Nk=6), shown in *magenta italics*:

Expanded Key:	*W_0*	*W_1*	*W_2*	*W_3*	*W_4*	*W_5*	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}	W_{14}	W_{15}	...	W_{44}	W_{45}	W_{46}	W_{47}	W_{48}	W_{49}	W_{50}	W_{51}
Round keys:	Round key 0				Round key 1				Round key 2				Round key 3				...	Round key 11				Round key 12			

The AddRoundKey (called *Round Key addition* in Rijndael) transformation merely applies each Round Key, in turn, to the State by a simple bit-wise exclusive OR operation.

MARS by Zunic et al., IBM.

This new design uses a special type of a Feistel network, which depends heavily on the instruction sets available on modern 32-bit processors. This has the benefit that on these target machines it is efficient, but it may lead to implementation problems/difficulties in cheaper architectures like smart cards.

RC6 by Rivest, Robshaw, and Yin, RSA Laboratories.

RC6 follows the ideas of RC5 with improvements. It attempts to avoid some of the differential attacks against RC5's data dependent rotations. However, there are some attacks that are not yet employed, and it is unclear whether RC6 is well enough analysed yet.

Serpent by Anderson, Biham, and Knudsen.

Serpent has a basically conservative but, in many ways innovative design. It may be implemented by bitslice (or vector) gate logic throughout. This makes it rather complicated to implement from scratch, and writing it in a non-bitslice way involves an efficiency penalty. The *32* rounds lead to probably the highest security margin on all AES candidates, while it is still fast enough for all practical purposes.

Twofish by Schneier et al., Counterpane Security.

Twofish is a block cipher designed by Counterpane, whose founder and CTO is **Bruce Schneier**. The design is highly delicate, supported with many alternative ways of implementation. It is crypt analysed in much detail, by the very authoritative "extended Twofish team". It is basically a Feistel cipher, but utilises many different ideas. This cipher has key dependent S-boxes like **Blowfish** (another cipher by Bruce Schneier).

Other Symmetric Ciphers

Blowfish

Bruce Schneier designed blowfish and it is a block cipher with a 64-bit block size and variable length keys (which may vary up to 448 bits). It has gained of acceptance in a number of applications, including Nautilus and PGPfone.

Blowfish utilises the idea of randomised S-boxes: while doing key scheduling, it generates large pseudo-random lookup tables through several encryptions. These tables depend on the user supplied key in a very complex manner. This makes blowfish highly resistant against many attacks such as differential and linear crypt analysis. But is not the algorithm of choice for environments where large memory space (something like *4096* bytes) is not available. The known attacks against Blowfish are based on its weak key classes.

CAST-128

CAST-128, Substitution-Permutation Network (SPN) cryptosystem, is similar to DES, which have good resistance to differential, linear and related-key crypt analysis. CAST-128 has Feistel structure and utilises eight fixed S-boxes. CAST-128 supports variable key lenghts between 40 and 128 bits (described in RFC2144).

IDEA

IDEA (International Data Encryption Algorithm), was developed at ETH Zurich in Switzerland by **Xuejia Lai** uses a 128-bit key, and is considered to be very secure. The best attacks against IDEA uses the impossible differential idea of **Biham, Shamir** and **Biryukov**. They can attack only *4.5* rounds of IDEA and this poses no threat to the total of *8.5* rounds used in IDEA. It is pertinent to note that IDEA is patented in the United States and in most European countries.

Rabbit

Rabbit is a stream cipher, based on iterating a set of coupled nonlinear function. It is characterised by a high performance in software.

RC4

RC4 is a stream cipher by Ron Rivest at RSA Data Security, Inc. It accepts keys of arbitrary length. It used to be a trade secret, until someone posted source code for an algorithm on the usenet, claiming it to be equivalent to RC4. The algorithm is very fast. Its security is unknown, but breaking it does not seem trivial either. Because of its speed, it may have uses in certain particular applications.

RC4 is essentially a pseudo random number generator, and the output of the generator is exclusive-ored with the data stream. For this reason, it is very important that the same RC4 key should not be used to encrypt two different data streams.

☞ Check Your Progress 2

1) What is the result of the Exclusive OR operation 1XOR0?

   (a) 1
   (b) 0
   (c) Indeterminate
   (d) 10

2) A block cipher:

   (a) Is an asymmetric key algorithm
   (b) Converts variable-length plaintext into fixed-length ciphertext
   (c) Breaks a message into fixed length units for encryption
   (d) Encrypts by operating on a continuous data stream.

3) What is the block length of the Rijndael Cipher?

   (a) 64 bits
   (b) 128 bits
   (c) Variable
   (d) 256 bits.

4) What is the key length of the Rijndael Block Cipher

   (a) 56 or 64 bits
   (b) 512 bits
   (c) 128, 192, or 256 bits
   (d) 512 or 1024 bits.

5) The NIST Advanced Encryption Standard uses the:

   (a) 3 DES algorithm
   (b) DES algorithm
   (c) Rijndael algorithm
   (d) IDEA algorithm.

6) The modes of DES do not include:

   (a) Electronic code book
   (b) Variable block feedback
   (c) Cipher block chaining
   (d) Output feedback.

3.4 PUBLIC KEY CRYPTOGRAPHY

Modern cryptography deals with communication between many different parties, without using a secret key for every pair of parties. In 1976, **Whitfield Diffie** and **Martin Hellman** published an invited paper in the ÏEEE Transactions on Information Theory titled "New Directions in Cryptography". This paper can be considered as the beginning of modern cryptography. Their paper described a two – key crypto system in which two parties can perform a secure communication over a non-secure channel without having to share a secret key. PKC depends upon the existence of so-called one-way function, or mathematical functions that are easy to calculate but the calculation of the inverse function is relatively difficult.

Generic PKC uses two keys that are mathematically related and knowledge of one key does not allow someone to easily determine the other key. One key is used to encrypt and the other key is used to decrypt. It does not matter which key is applied first and because a pair of keys are used, this is called asymmetric key cryptography. In PKC, one of the keys is designated as the public key and the other key is designated as the private key. The public key may be advertised but the private key is never revealed to another party. It is straightforward enough to send messages under this scheme. Suppose Alice wants to send Bob a message, Alice encrypts some information using Bob's public key; Bob decrypts the ciphertext using his private key.

The most common PKC implementation is RSA, named after the three MIT mathematicians who developed it — **Ronald Rivest**, **Adi Shamir, and Leonard Adleman.** RSA can be used for key exchange, digital signatures, or encryption of small blocks of data. The *Figures 11 and 12* highlights how digital signature are created and verified. The detailed discussion will be made in the next unit. RSA uses a variable size encryption block and variable size key. The key pair is derived from a very large number, *n*, that is the product of two large prime numbers selected through special rules; these primes may be 100 or more digits in length each, yielding an *n* with roughly twice as many digits as the prime factors. The public key includes *n* and a derivate of one of the factor of *n*; an adversary cannot determine the prime factor of *n* (and, therefore, the private key) from this information alone and that is what makes the RSA algorithm so secure.

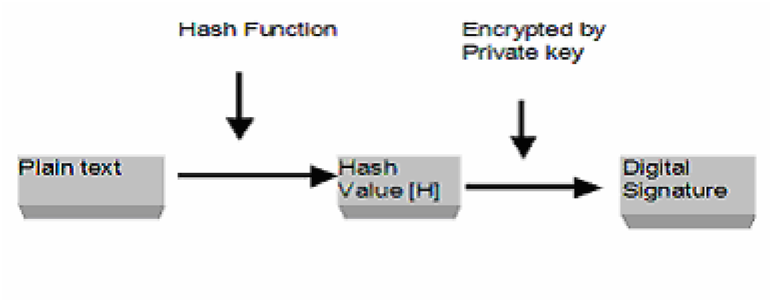


Figure11: Digital Signature

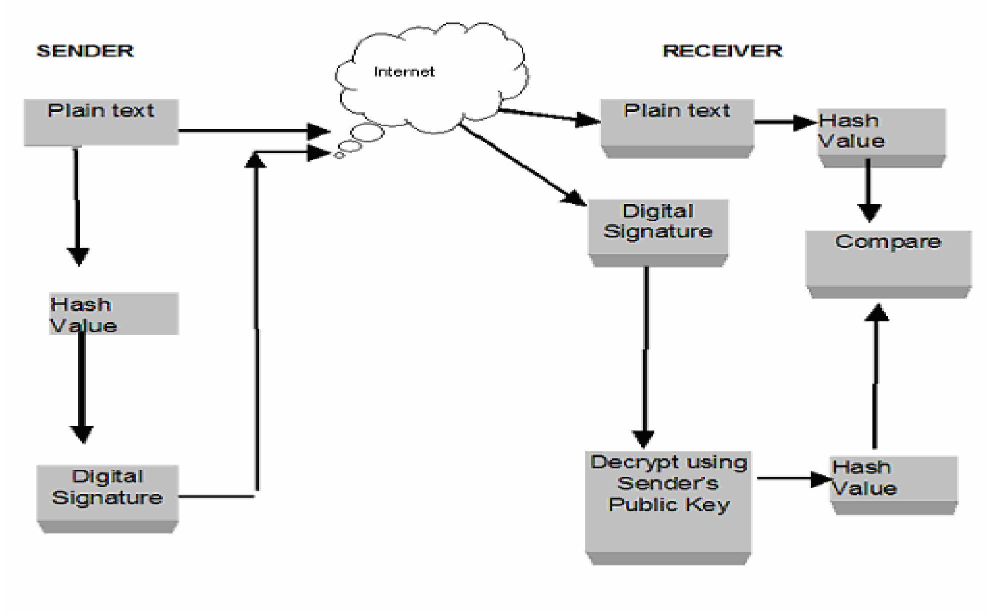


Figure12: Digital signature verification process

Goldwasser, Micali, and Rivest gave the rigorous definition of security for signature scheme and provided the first construction that probably satisfied that definition, under a suitable assumption. The assumption that claw-free pair of permutations exist, is implied by the assumption that integer factorisation is hard. The earlier signature scheme due to **Merkle** was also shown to satisfy this definition. **Rompel, Naor** and **Yung** showed how to construct a digital signature scheme using any one-way function. The inefficiency of the above mentioned schemes, and due to the fact that these schemes require the signer's secret key to change between invocations of the signing algorithm make these solutions impractical.

Several other signature schemes have been shown to be secure in the so-called random-oracle model. The random-oracle model is a model of computation in which it is assumed that a given hash function behaves as an ideal random function. An ideal random function is a function where the input domain is the set of binary strings, such that each binary string is mapped to a random binary string of the same length. Although this assumption is evidently false, as it formalises the notion that the hash function is like a black box and its output cannot be determined until it is evaluated. A proof of security in the random oracle model gives some evidence of resilience to attack. The RSA signatures with special message formatting, the Fiat-Shamir, and the Schnorr signature schemes have been analysed and proven secure in the random oracle model. However, it is known that security in the random oracle model does not imply security in the plain model.

Gennaro, Halevi, Rabin, Cramer Shoup proposed the first signature schemes whose efficiency is suitable for practical use and whose security analysis does not assume an ideal random function. These schemes are considered to be secure, under RSA assumption.

PKC depends upon the existence of so-called *one-way functions*, or mathematical functions that are easy to compute whereas their inverse function is relatively difficult to compute. Let me give you two simple examples:

1) Multiplication vs. factorisation
2) Exponentiation vs. logarithms

The important Public-key cryptography algorithms are:

- ***RSA***
- ***Diffie-Hellman:*** After the RSA algorithm was published, **Diffie** and **Hellman** came up with their own algorithm. D-H is used for secret-key key exchange only, and not for authentication or digital signatures.
- ***Digital Signature Algorithm (DSA):*** The algorithm specified in NIST's Digital Signature Standard (DSS), provides digital signature capability for the authentication of messages.
- ***ElGamal:*** Designed by **Taher Elgamal**, is a PKC system similar to Diffie-Hellman and used for key exchange.
- ***Elliptic Curve Cryptography (ECC):*** A PKC algorithm based upon elliptic curves. ECC can offer levels of security with small keys comparable to RSA and other PKC methods. It was designed for devices with limited compute power and/or memory, such as smartcards and PDAs.
- ***Public-Key Cryptography Standards (PKCS):*** A set of interoperable standards and guidelines for public-key cryptography, designed by RSA Data Security Inc.
  - PKCS #1: RSA Cryptography Standard (Also RFC 3447)

  - PKCS #2: *Incorporated into PKCS #1.*
  - PKCS #3: Diffie-Hellman Key-Agreement Standard
  - PKCS #4: *Incorporated into PKCS #1.*
  - PKCS #5: Password-Based Cryptography Standard (PKCS #5 V2.0 is also RFC 2898)
  - PKCS #6: Extended-Certificate Syntax Standard (being phased out in favor of X.509v3)
  - PKCS #7: Cryptographic Message Syntax Standard (Also RFC 2315)
  - PKCS #8: Private-Key Information Syntax Standard
  - PKCS #9: Selected Attribute Types (Also RFC 2985)
  - PKCS #10: Certification Request Syntax Standard (Also RFC 2986)
  - PKCS #11: Cryptographic Token Interface Standard
  - PKCS #12: Personal Information Exchange Syntax Standard
  - PKCS #13: Elliptic Curve Cryptography Standard
  - PKCS #14: Pseudorandom Number Generation Standard is no longer available
  - PKCS #15: Cryptographic Token Information Format Standard

- ***Cramer-Shoup:*** A public-key cryptosystem proposed by **R. Cramer** and **V. Shoup** of IBM in 1998.
- ***Key Exchange Algorithm (KEA):*** A variation on Diffie-Hellman; proposed as the key exchange method for Capstone.
- ***LUC:*** A public-key cryptosystem designed by **P.J. Smith** and based on Lucas sequences. Can be used for encryption and signatures, using integer factoring.

Public Key Infrastructure (PKI) in India

The Act provides the Controller of Certifying Authorities to license and regulate the working of CAs and also to ensure that CAs does not violate any of the provisions of the Act. CCA has created the framework for Public Key Infrastructure in India. It has prescribed technical standards for cryptography and physical security based on international standards/best practices of ITU, IETF, IEEE etc. CAs has to prove compliance with these standards through a stringent audit procedure that has been put in place. CAs also needs to get their CPS (Certification Practice Statement) approved by the CCA. India has seven Certifying Authorities: (1) SafeScrypt-Certifying Authority; (2) National Informatics Centre-Certifying Authority; (3) Tata Consultancy- Services Certifying Authorities; (4) Institute for Research and Development in Banking Technology-Certifying Authority; (5) Customs and Central Excise-Certifying Authority; (6) Mahanagar Telephone Nigam Limited-Certifying Authority; and (7) n(Code) Solutions CA (GNFC).

The CCA also maintains the National Repository of Digital Certificates (NRDC), as required under the IT Act 2000 that contains all the certificates issued by all the CAs in India. The CCA operates its signing activity through the Root Certifying Authority of India (RCAI), which is operational under stringent controls.

Public Key Cryptosystem

Asymmetric key algorithms or Public key algorithm use a different key for encryption and decryption (*Figure 13*), and the decryption key cannot (practically) be derived from the encryption key. Public key methods are important because they can be used to distribute encryption keys or other data securely even when the parties have no opportunity to agree on a secret key in private. All known public key methods are quite slow, and they are usually only used to encrypt session keys, that are then used to encrypt the bulk of the data using a symmetric encryption.

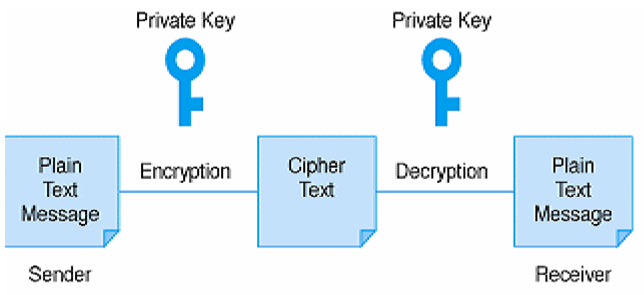


Figure13: Encryption and decryption process

3.4.1 RSA Public Key Algorithm (Rivest-Shamir-Adelman Algorithm)

RSA Public Key Algorithm is the most commonly used public key algorithm and this algorithm can be used both for encryption and for signing. It is generally considered to be secure when sufficiently long keys are used (512 bits is insecure, 768 bits is moderately secure, and 1024 bits is good). The security of RSA relies on the difficulty of factoring large integers. The recent advances in factoring large integers would make RSA vulnerable. It is patented in US and the Patent expired in year 2000.

RSA Algorithm

Key Generation Algorithm

1) Generate two large random primes, p and q, of approximately equal size such that their product $n = pq$ is of the required bit length, e.g., 1024 bits.

2) Compute n = pq and (φ) phi = (p –1)(q –1).

3) Choose an integer e, 1 < e < phi, such that gcd(e, phi) = 1.

4) Compute the secret exponent d, 1 < d < phi, such that ed ≡ 1 (mod phi).

5) The public key is (n, e) and the private key is (n, d). The values of p, q, and phi should also be kept secret.

Where

- n is known as the *modulus*.
- e is known as the *public exponent* or *encryption exponent*.
- d is known as the *secret exponent* or *decryption exponent*.

Encryption

Sender A does the following:

1) Obtains the recipient B’s public key (n, e).

2) Represents the plaintext message as a positive integer m.

3) Computes the ciphertext c = m$^{\wedge e}$ mod n.

4) Sends the ciphertext c to B.

Decryption

Recipient B does the following:

1) Uses his private key (n, d) to compute m = c$^{\wedge d}$ mod n.

2) Extracts the plaintext from the integer representative m.

Summary of RSA

- n = pq where p and q are distinct primes.
- phi, φ = (p-1)(q-1)
- e < n such that gcd(e, phi)=1
- d = e$^{\wedge -1}$ mod phi.
- c = m$^{\wedge e}$ mod n, 1<m<n.
- m = c$^{\wedge d}$ mod n.

Example RSA Algorithm

Let us select two prime numbers, p = 7 and q= 17. Keys are generated as follows:

1. Two prime numbers, p = 7 and q = 17
2. n = pq = 7 x17 = 119
3. P(n) = (p-1) (q-1) = (7-1) (17-1) = 6 x 16 = 96
4. Find e which is relatively prime to ϕ(n) = 96 and less than ϕ(n).
   e = 5 for this case
5. Find d such that d.e = 1 mod 96 and d<96 d = 77
6. So, public key in = [5,119] and private key is = [77, 119]

Encryption and Decryption using RSA algorithm

Encryption: Let plain text input of M = 19
Cipher text C = M^e(mod n)

$$= 19^5 \text{ (mod119)} = \frac{2476099}{119}$$

= 20807 with remainder of 66
Cipher Tex t= 66

Decryption:

M=c^d(modn)
= 66^{77}(mod 119)
= 19

3.4.2 Diffie-Hellman

Diffie-Hellman is a commonly used public-key algorithm for key exchange. It is generally considered to be secure when sufficiently long keys and proper generators are used. The security of Diffie-Hellman relies on the difficulty of the discrete logarithm problem (which is believed to be computationally equivalent to factoring large integers). Diffie-Hellman is claimed to be patented in the United States, but the patent expired on April 29, 1997. There are also strong rumours that the patent might in fact be invalid (there is evidence of it having been published over an year before the patent application was led). Diffie-Hellman is sensitive to the selection of the strong prime, size of the secret exponent, and the generator.

The "Diffie-Hellman Method For Key Agreement" allow two hosts to create and share a secret key.

1) First the hosts must get the "Diffie-Hellman parameters". A prime number, 'p' (larger than 2) and "base", 'g', an integer that is smaller than 'p'. They can either be hard coded or fetched from a server.

2) The hosts each secretly generate a private number called 'x', which is less than "p – 1".

3) The hosts next generate the public keys, 'y'. They are created with the function:

   y = g^x % p

4) The two host now exchange the public keys ('y') and the exchanged numbers are converted into a secret key, 'z'.

   z = y^x % p

   'z' can now be used as the key for whatever encryption method is used to transfer information between the two hosts. Mathematically, the two hosts should have generated the same value for 'z'.

   z = (g^x % p)^x' % p = (g^x' % p)^x % p

All of these numbers are positive integers

x^y	means: x is raised to the y power
x%y	means: x is divided by y and the remainder is returned

Example:

Prime Number p = 353 and primitive root of 353, in this case is g = 3. Let A is sender B is receives A & B select secret key as XA = 97 and XB = 233. Now A & B computes their public keys. YA = 3^{233}. Now A & B computes their public keys.
YA = 3^{97} mod 353 = 40
YB = 3^{233} mod 353 = 248

After exchanging their public keys, A & B can compute the common secret key:
A computes:

Z=(YB) mod 253
B computes = 248^{97} mod 353 = 160
Z = $(YA)^{xb}$mod353 = 40^{233} mod 353 = 160

3.4.3 Elliptic Curve Public Key Cryptosystems

Elliptic Curve Public Key Cryptosystems is an emerging field. They have been slow to execute, but have become feasible with modern computers. They are considered to be fairly secure, but haven't yet undergone the same scrutiny as done on RSA algorithm.

The Public-Key cryptography systems use hard-to-solve problems as the basis of the algorithm. The most predominant algorithm today for public-key cryptography is RSA, based on the prime factors of very large integers. While RSA can be successfully attacked, the mathematics of the algorithm has not been comprised, per se; instead, computational brute-force attacks have broken the keys. The protection mechanism is "simple" — keep the size of the integer to be factored ahead of the computational curve!

In 1985, cryptographers **Victor Miller** (IBM) and **Neal Koblitz** (University of Washington) proposed the Elliptic Curve Cryptography (ECC) independently (as shown in *Figure 14*). ECC is based on the difficulty of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP). Like the prime factorisation problem, ECDLP is another "hard" problem that is deceptively simple to state: Given two points, P and Q, on an elliptic curve, find the integer *n*, if it exists, such that P = nQ.

Elliptic curves combine number theory and algebraic geometry. These curves can be defined over any field of numbers (i.e., real, integer, complex), although, we generally see them used over finite fields for applications in cryptography. An elliptic curve consists of the set of real numbers (x, y) that satisfies the equation:

$y^2 = x^3 + ax + b$

The set of all the solutions to the equation forms the elliptic curve. Changing *a* and *b* changes the shape of the curve, and small changes in these parameters can result in major changes in the set of (x,y) solutions.

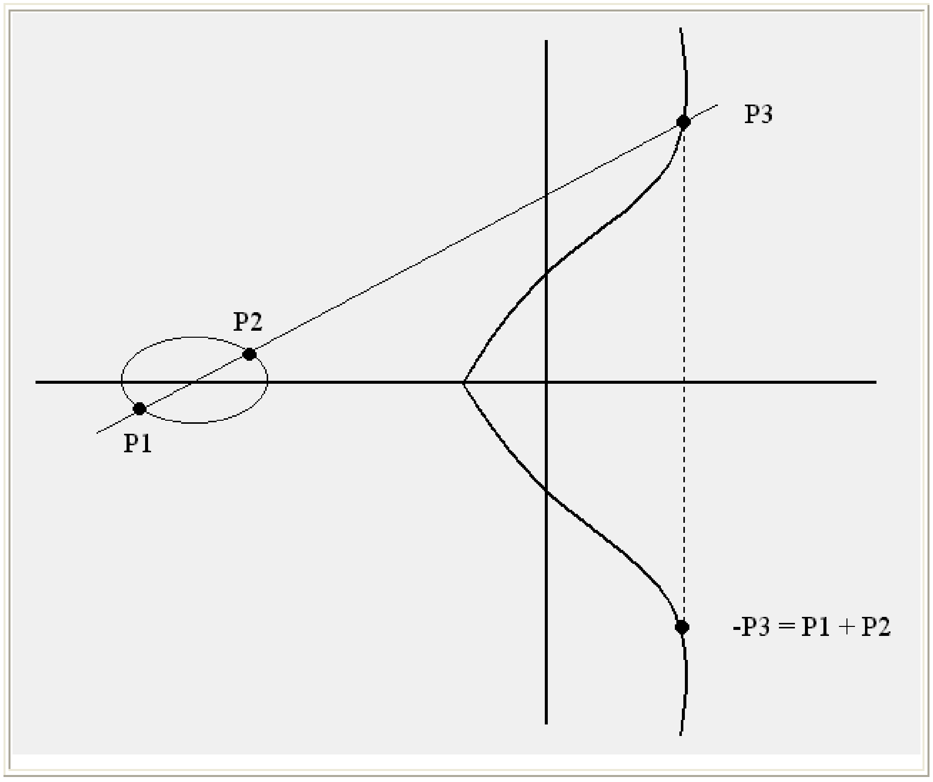


Figure14: Elliptic Curve Cryptography

The *figure* above shows the addition of two points on an elliptic curve. Elliptic curves have the interesting property that adding two points on the elliptic curve yields a third point on the curve. Therefore, adding two points, P1 and P2, gets us to point P3, also on the curve. Small changes in P1 or P2 can cause a large change in the position of P3.

Let us analyse the original problem as stated above. The point Q is calculated as a multiple of the starting point, P, *or*, Q = nP. An attacker might know P and Q but finding the integer, *n*, is a difficult problem to solve. Q is the public key, then, and *n* is the private key.

RSA has been the mainstay of PKC for over two decades. But ECC is exciting because of their potential to provide similar levels of security compared to RSA but with significantly reduced key sizes. Certicom Corp. (http://www.certicom.com/), one of the major proponents of ECC, suggests the key size relationship between ECC and RSA as per the following table:

ECC and RSA Key Comparison

RSA Key Size	Time to Break Key (MIPS Years)	ECC Key Size	RSA:ECC Key-Size Ratio
512	10^4	106	5:1
768	10^8	132	6:1
1,024	10^{11}	160	7:1
2,048	10^{20}	210	10:1
21,000	10^{78}	600	35:1

Since the ECC key sizes are so much shorter than comparable RSA keys, the length of the public key and private key is much shorter in elliptic curve cryptosystems. Therefore, this results in faster processing, and lower demands on memory and bandwidth. In practice, the final results are not yet in; RSA, Inc. notes that ECC is faster than RSA for signing and decryption, but slower than RSA for signature verification and encryption.

Nevertheless, ECC is particularly useful in applications where memory, bandwidth, and/or computational power is limited (e.g., a smartcard) and it is in this area that ECC use is expected to grow.

3.4.4 DSA

DSS (Digital Signature Standard): A signature-only mechanism endorsed by the United States Government. Its design has not been made public, and many people have found potential problems with it (e.g., leaking hidden data, and revealing your secret key if you ever happen to sign two different messages using the same random number).

The DSA is used by a signatory to generate a digital signature on data and by a verifier to verify the authenticity of the signature. Each signatory has a public and private key. The private key is used in the signature generation process and the public key is used in the signature verification process. For both signature generation and verification, the data which is referred to as a message, M, is reduced by means of the Secure Hash Algorithm (SHA) specified in FIPS YY. An adversary, who does not know the private key of the signatory, cannot generate the correct signature of the signatory. In other words, signatures cannot be forged. However, by using the signatory's public key, anyone can verify a correctly signed message.

A means of associating public and private key pairs for the corresponding users is required. That is, there must be a binding of a user's identity and the user's public key. A mutually trusted party may certify this binding. For example, a certifying authority could sign credentials containing a user's public key and identity to form a certificate. Systems for certifying credentials and distributing certificates are beyond the scope of this standard. NIST intends to publish separate document(s) on certifying credentials and distributing certificates.

The DSA makes use of the following parameters:

1) p = a prime modulus, where $2^{L-1} < p < 2^{L}$ for 512 = < L = <1024 and L a multiple of 64

2) q = a prime divisor of p - 1, where $2^{159} < q < 2^{160}$

3) $g = h^{(p-1)/q} \bmod p$, where h is any integer with 1 < h < p - 1 such that $h^{(p-1)/q} \bmod p > 1$(g has order q mod p)

4) x = a randomly or pseudorandomly generated integer with 0 < x < q

5) $y = g^{x} \bmod p$

6) k = a randomly or pseudorandomly generated integer with 0 < k < q

The integers p, q, and g can be public and can be common to a group of users. A user's private and public keys are x and y, respectively. They are normally fixed for a period of time. Parameters x and k are used for signature generation only, and must be kept secret. Parameter k must be regenerated for each signature.

The signature of a message M is the pair of numbers r and s computed according to the equations below:

$$r = (g^{k} \bmod p) \bmod q \text{ and}$$

$$s = (k^{-1}(SHA(M) + xr)) \bmod q.$$

In the above, k^{-1} is the multiplicative inverse of k, mod q; i.e., $(k^{-1} k) \bmod q = 1$ and $0 < k^{-1} < q$. The value of SHA (M) is a 160-bit string output by the Secure Hash Algorithm specified in FIPS 180. For use in computing s, this string must be converted to an integer.

As an option, one may wish to check if r = 0 or s = 0. If either r = 0 or s = 0, a new value of k should be generated and the signature should be recalculated (it is extremely unlikely that r = 0 or s = 0 if signatures are generated properly).

The signature is transmitted along with the message to the verifier.

Prior to verifying the signature in a signed message, p, q and g plus the sender's public key and identity are made available to the verifier after proper authentication.

Let M', r' and s' be the received versions of M, r, and s, respectively, and let y be the public key of the signatory. To verify first check to see that 0 < r' < q and 0 < s' < q; if either condition is violated, the signature shall be rejected. If these two conditions are satisfied, the verifier computes:

$$w = (s')^{-1} \bmod q$$

$$u1 = ((SHA(M')w) \bmod q$$

$$u2 = ((r')w) \bmod q$$

$$v = (((g)^{ul} (y)^{u2}) \bmod p) \bmod q.$$

If v = r', then the signature is verified and the verifier can have high confidence that the received message was sent by the party holding the secret key x corresponding to y. For a proof that v = r' when M' = M, r' = r, and s' = s, see Appendix 1.

If v does not equal r', then the message may have been modified, the message may have been incorrectly signed by the signatory, or the message may have been signed by an impostor. The message should be considered invalid.

ElGamal public key cryptosystem. Based on the discrete logarithm problem.

ElGamal consists of three components: the key generator, the encryption algorithm, and the decryption algorithm.

The key generator works as follows:

- Alice generates an efficient description of a cyclic group *G* of order *q* with generator *g*. See below for specific examples of how this can be done.
- Alice chooses a random *x* from $\{0,...,q - 1\}$.
- Alice computes $h = g^x$.
- Alice publishes *h*, along with the description of *G*,*q*,*g*, as her public key. Alice retains *x* as her secret key.

The encryption algorithm works as follows: to encrypt a message *m* to Alice under her public key (G,q,g,h),

- Bob converts *m* into an element of *G*.
- Bob chooses a random *k* from $\{0,...,q - 1\}$, then calculates $c_1 = g^k$ and $c_2 = m \cdot h^k$.
- Bob sends the ciphertext (c_1,c_2) to Alice.

The decryption algorithm works as follows: to decrypt a ciphertext (c_1,c_2) with her secret key *x*,

- Alice computes $c_2(c_1^x)^{-1}$ as the plaintext message.

Note that the decryption algorithm does indeed produce the intended message since:

$$c_2(c_1^x)^{-1} \equiv \frac{m \cdot h^k}{g^{xk}} \equiv \frac{m \cdot g^{xk}}{g^{xk}} \equiv m \pmod q$$

If the space of possible messages is larger than the size of *G*, then the message can be split into several pieces and each piece can be encrypted independently. Typically, however, a short key to a symmetric-key cipher is first encrypted under ElGamal, and the (much longer) intended message is encrypted more efficiently using the symmetric-key cipher — this is termed *hybrid encryption*.

3.5 MATHEMATICAL BACKGROUND

In this section, we will find out mathematical background from mathematical Algorithms.

3.5.1 Exclusive OR (XOR)

Exclusive OR (XOR) is one of the fundamental mathematical operations used in cryptography (and many other applications). **George Boole**, a mathematician in the late 1800s, invented a new form of "algebra" that provides the basis for building electronic computers and microprocessor chips. **Boole** defined a bunch of primitive logical operations where there are one or two inputs and a single output depending upon the operation; the input and output are either TRUE or FALSE. The most elemental **Boolean** operations are:

- NOT: The output value is the inverse of the input value (i.e., the output is TRUE if the input is false, FALSE if the input is true).
- AND: The output is TRUE if all inputs are true, otherwise FALSE. (e.g., "the sky is blue AND the world is flat" is FALSE while "the sky is blue AND security is a process" is TRUE.)
- OR: The output is TRUE if either or both inputs are true, otherwise FALSE. (e.g., "the sky is blue OR the world is flat" is TRUE and "the sky is blue OR security is a process" is TRUE).
- XOR (Exclusive OR): The output is TRUE if exactly one of the inputs is TRUE, otherwise FALSE. (e.g., "the sky is blue XOR the world is flat" is TRUE while "the sky is blue XOR security is a process" is FALSE.)

In computers, Boolean logic is implemented in *logic gates*; for design purposes, XOR has two inputs and a single output, and its logic diagram looks like this:

XOR	0	1
0	0	1
1	1	0

So, in an XOR operation, the output will be a 1 if one input is a 1; otherwise, the output is 0. The real significance of this is to look at the "identity properties" of XOR. In particular, any value XORed with itself is 0 and any value XORed with 0 is just itself. Why does this matter? Well, if I take my plaintext and XOR it with a key, I get a jumble of bits. If I then take that jumble and XOR it with the same key, I return to the original plaintext.

3.5.2 The Modulo Function

The *modulo* function is, simply, the remainder function. It is commonly used in programming and is critical to the operation of any mathematical function using digital computers.

To calculate *X modulo Y* (usually written *X mod Y*), you merely determine the remainder after removing all multiples of Y from X. Clearly, the value X mod Y will be in the range from 0 to Y-1.

Some examples should clear up any remaining confusion:

- 15 mod 7 = 1
- 25 mod 5 = 0
- 33 mod 12 = 9
- 203 mod 256 = 203

Modulo arithmetic is useful in crypto because it allows us to set the size of an operation and be sure that we will never get numbers that are too large. This is an important consideration when using digital computers.

☞ **Check Your Progress 3**

1) Which of the following is an example of a symmetric key algorithm?

   (a)
   (b) RSA
   (c) Diffie-Hellman
   (d) Knapsack

2) Which of the following is a problem with symmetric key algorithms?

   (a) It is slower than asymmetric key encryption
   (b) Most algorithms are kept proprietry
   (c) Work factor is not a function of the key size.
   (d) It provides secure distribution of the secret key.

3) Which of the following is an example of an asymmetric key algorithms?

   (a) ELLIPTIC CURVE
   (b) IDEA
   (c) 3 DES
   (d) DES

4) In public key cryptography:

   (a) Only the public key can encrypt, and only the private can decrypt
   (b) Only the private key can encrypt, and only the public can decrypt
   (c) The public key is used to encrypt and decrypt
   (d) If the public key encrypts, and only the private can decrypt.

5) In a block cipher, diffusion:

   (a) Conceals the connection between the ciphertext and plaintext
   (b) Spread the influence of a plaintext character over many ciphertext characters.
   (c) Cannot be accomplished
   (d) Is usually implemented by non-linear S-boxes.

6) State whether following statements are True or False. T ☐ F ☐

   (a) Work factor of double DES is the same as for single DES
   (b) Elliptic curse cryptosystem have a lower strength per bit than RSA.
   (c) In digitally-signed message transmission using a hash function the message digest is encrypted in the public key of the sender.

3.6 SUMMARY

Information security can be defined as technological and managerial procedures applied to computer systems to ensure the availability, integrity, and confidentiality of the information managed by computer. Cryptography is a particularly interesting field because of the amount of work that is, by necessity, done in secret. The irony is that today, secrecy is not the key to the goodness of a cryptographic algorithm. Regardless of the mathematical theory behind an algorithm, the best algorithms are those that are well known and well documented because they are also well tested and well studied! In fact, time is the only true test of good cryptography; any cryptographic scheme that stays in use year after year is most likely to be a good one. The strength of cryptography lies in the choice and management of the keys; longer keys will resist attack better than shorter keys.

With the introduction of IT Act 2000, the electronic transactions and electronically signed documents are valid under the court of law. Use of PKI in India is expected to increase rapidly and for activation of global e-commerce and so also the use of digital signatures within domestic and other Global PKI domains. This requires the interoperability of PKI based applications.

Digitisation of information, networking, and WWW (world wide web), has changed the economics of information. The Copyright Act 1957 as amended up to 1999 takes care of the technological changes that still require major amendments in order to deal with the challenges posed by the computer storage, Internet and the digital revolution. With India being party to Trade Related Aspects of Intellectual Properties (TRIPs), electronic transactions of IPRs and consultancy services related to them is expected to be a new possibility in India. This may also promote our technology base to keep pace with global trends. As India has already enacted IT Act 2000, this allows transactions signed electronically for e-commerce primarily to be enforced in a court of law. Several issues arise when we consider using digital documents and exchanging them over the Internet, such as eavesdropping, tampering, impersonation etc. All these can be remedied with the use of public key infrastructure (PKI). However, the question of the time when a document was created may be answered by using Digital Time stamping service, which is based on PKI. This information may prove to be crucial for most e-commerce legally binding transactions, in particular for supporting non-repudiation of digitally signed transactions.

3.7 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Cryptography is the service of writing in secret code. It is the key technology used when communicating over any un-trusted medium, particularly for Internet.

2) (i) Authentication,
   (ii) Privacy/confidentiality,
   (iii) Integrity, and
   (iv) Non-repudiation

3) (i) Secret key (or symmetric) cryptography,
   (ii) Public key (or asymmetric) cryptography, and
   (iii) Hash functions

Check Your Progress 2

1) (a) 1
   (b) 0
   (c) Indeterminated
   (d) 10

2) Breaks a message into fixed length units for encryption.

3) Variable

4) 128, 192, or 256 bits

5) Rijndael algorithm

6) Variable block feedback

Check Your Progress 3

1) Rijndael

2) It is slower than asymmetric key encryption

3) Elliptic Curve

4) If the public key encrypts, and only the private can decrypt

5) Spread the influence of a plaintext character over many ciphertext characters.

6) (a) True
   (b) False
   (c) False

3.8 FURTHER READINGS

1) *Network Security Essential - Application and standard*, Willam Stallings, Pearson Education, New Delhi.

2) *Computer Networks,* A.S. Tanenbaum, 4th Edition, Practice Hall of India, New Delhi, 2002.