CHAPTER NINE

Cryptography

One cannot escape the feeling that these mathematical formulae have an independent existence and an intelligence of their own, that they are wiser than we are, wiser even than their discoverers, that we get more out of them than we originally put in to them.

Heinrich Hertz (1857–1894)

learning objectives

After studying this chapter the students will be able to:

LO 1 Know the basic operations used by common encryption techniques.

LO 2 Explain the concept of secret-key cryptography, Data Encryption Standard (DES), International Data Encryption Algorithm (IDEA) and RC Ciphers.

LO 3 Underline the concept of public-key cryptography and implement the Rivest, Shamir and Adleman (RSA) algorithm, Pretty Good Privacy (PGP) and one-way Hashing.

LO 4 Understand and solve problems related to the Diffie-Hellman protocol, elliptic curve cryptography, quantum cryptography, biometric encryption and chaos-based cryptography.

9.1 Introduction to Cryptography

This chapter comes with a video overview by the author. Scan here to know more or Visit http://qrcode.flipick.com/index.php/627

Cryptography is the science of devising methods that allow information to be sent in a secure form in such a way that the only person able to retrieve this information is the intended recipient. Encryption is based on algorithms that scramble information into unreadable or non-discernable form. Decryption is the process of restoring the scrambled information to its original form (see Fig. 9.1).

DID YOU KNOW ? A **Cryptosystem** is a collection of algorithms and associated procedures for hiding and revealing (un-hiding!) information. **Cryptanalysis** is the process (actually, the art) of analysing a cryptosystem, either to verify its integrity or to break it for ulterior motives. An **Attacker** is a person or system that performs cryptanalysis in order to break a cryptosystem. Attackers are also referred to as hackers, interlopers or eavesdroppers. The process of attacking a cryptosystem is often called **Cracking** or **Hacking**.

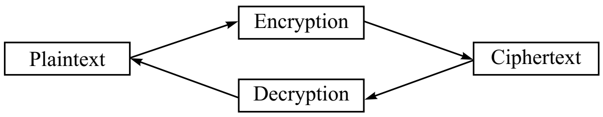

Fig. 9.1 The process of encryption and decryption.

The job of a **Cryptanalyst** is to find the weaknesses in the cryptosystem. In many cases, the developers of a cryptosystem announce a public challenge with a large prize-money for anyone who can crack the scheme. Once a cryptosystem is broken (and the cryptanalyst discloses his techniques), the designers of the scheme try to strengthen the algorithm. Just because a cryptosystem has been broken does not render it useless. The hackers may have broken the system under optimal conditions using equipment (fast computers, dedicated microprocessors, etc.) that is usually not available to regular people. Some cryptosystems are rated in terms of the number of years and the price of the computing equipment it would take to break them!

In the last few decades cryptographic algorithms, being mathematical by nature, have become sufficiently advanced that they can only be handled by computers. This, in effect, means that the uncoded message (prior to encryption) is binary in form and can, therefore, be anything—a picture, a voice, an e-mail or even a video.

DID YOU **KNOW** ?

Cryptography is not merely used for military and diplomatic communications as many people tend to believe. In reality, cryptography (although obviously essential for the military and diplomatic services) has many commercial uses and applications. From protecting confidential company information, to protecting a telephone call, to allowing someone to order a product on the Internet without the fear of their credit card number being intercepted and used against them, cryptography is all about increasing the level of privacy of individuals and groups. For example, cryptography is often used to prevent forgers from counterfeiting winning lottery tickets. Each lottery ticket can have two numbers printed onto it, one plaintext and one the corresponding cipher. Unless the counterfeiter has cryptanalysed the lottery's cryptosystem he or she will not be able to print an acceptable forgery.

In this chapter

We begin with an overview of different encryption techniques. We will then introduce the basic operations used by common encryption techniques, in LO 1.

We will then understand the concept of secret-key cryptography. Some specific secret-key cryptographic techniques will be studied. Specifically, the Data Encryption Standard (DES) and the International Data Encryption Algorithm (IDEA) will be discussed in detail. A brief introduction to RC ciphers will also be given, in LO 2.

The public-key cryptography will be introduced next and the Rivest, Shamir and Adleman (RSA) algorithm will be discussed as a popular example. A hybrid cryptographic algorithm and Pretty Good Privacy (PGP) will be introduced. We will also learn about one-way Hashing, in LO 3.

Finally, a flavour of some other cryptographic techniques in use today will be given. The chapter will discuss the Diffie-Hellman Protocol and the Elliptic Curve Cryptography. The domain of cryptography is ever-expanding, and this chapter will introduce some of the emerging techniques such as Quantum Cryptography, Biometric Encryption and Chaos-based cryptography. The chapter will conclude with a discussion on Cryptanalysis and the politics of cryptography, in LO 4.

9.2 An Overview of Encryption Techniques

LO 1

Know the basic operations used by common encryption techniques.

The goal of a cryptographic system is to provide a high level of confidentiality, integrity, non-repudiation and authenticity to information that is exchanged over networks.

Confidentiality provides privacy for messages and stored data by hiding information using encryption techniques. Message integrity provides assurance to all parties that a message remains unchanged from the time it was created to the time it was opened by the recipient. Altered messages can lead to actions that are inappropriate. Non-repudiation can provide a way of proving that the message came from someone even if they try to deny it. Authentication provides two services. Firstly, it identifies the origin of the message. Secondly, it verifies the identity of a user logging into a system and continues to verify their identity in case someone tries to break into the system.

Definition 9.1 A message being sent is known as **Plaintext**. The message is coded using a cryptographic algorithm. This process is called **Encryption**. An encrypted message is known as **Ciphertext**, and is turned back into plaintext by the process of **Decryption**.

It must be assumed that any eavesdropper has access to all communications between the sender and the recipient. A method of encryption is only secure even if with such complete access, the eavesdropper is still unable to recover the original plaintext from the ciphertext.

There is a big difference between *security* and *obscurity*. Suppose, a message is left for somebody in an airport locker and the details of the airport is also known. If the locker number is known only to the intended recipient, then this message is not secure, merely obscure. If however, all potential eavesdroppers know the exact location of the locker, and they still cannot open the locker and access the message, then this message is secure.

Definition 9.2 A **key** is a value that causes a cryptographic algorithm to run in a specific manner and produce a specific ciphertext as an output. The key size is usually measured in bits. The bigger the key size, the more secure will be the algorithm.

Example 9.1 Suppose we have to encrypt and send the following stream of binary data (which might be originating from voice, video, text or any other source)

0 1 1 0 0 0 1 0 1 0 0 1 1 1 1 1 …

We can use a 4-bit long key, $x = 1011$, to encrypt this bit stream. To perform encryption, the plaintext (binary bit stream) is first subdivided into blocks of 4 bits.

0 1 1 0 0 0 1 0 1 0 0 1 1 1 1 1…

Each sub-block is XORed (binary addition without carry) with the key, $x = 1011$. The basic XOR operation (⊕) is as follows.

$$0 \oplus 0 = 0,\ 0 \oplus 1 = 1,\ 1 \oplus 0 = 1,\ 1 \oplus 1 = 1.$$

The encrypted message will be

1 1 0 1 1 0 0 1 0 0 1 0 0 1 0 0…

The recipient must also possess the knowledge of the key in order to decrypt the message. The decryption process is fairly simple in this case. The ciphertext (the received binary bit stream) is first subdivided in to blocks of 4 bits. Each sub-block is XORed with the key, $x = 1011$. The decrypted message will be the original plaintext

$$0110 \quad 0010 \quad 1001 \quad 1111\ldots$$

It should be noted that just one key is used both for encryption and decryption.

Example 9.2 Let us devise an algorithm for text messages, which we shall call *character* + x. Let $x = 5$. In this encryption technique, we replace every alphabet by the fifth one following it, i.e., A becomes F, B becomes G, C becomes H, and so on. The recipients of the encrypted message just need to know the value of the key, x, in order to decipher the message. The key must be kept separate from the encrypted message being sent. Because there is just one key which is used for encryption and decryption, this kind of technique is called **Symmetric Cryptography** or **Single Key Cryptography** or **Secret Key Cryptography**. The problem with this technique is that the key has to be kept confidential. Also, the key must be changed from time to time to ensure secrecy of transmission. This means that the secret key (or the set of keys) has to be communicated to the recipient. This might be done physically.

To get around this problem of communicating the key, the concept of **Public Key Cryptography** was developed by Diffie and Hellman. This technique is also called the **Asymmetric Encryption**. The concept is simple. There are two keys, one is held privately and the other one is made public. What one key can lock, the other key can unlock.

Example 9.3 Suppose we want to send an encrypted message to recipient A using the public key encryption technique. To do so we will use the public key of the recipient A and use it to encrypt the message. After recipient A receives the message, he decrypts it with his private key. Only the private key of recipient A can decrypt a message that has been encrypted with his public key. Similarly, recipient B can only decrypt a message that has been encrypted with his public key. Thus, no private key ever needs to be communicated and hence one does not have to trust any communication channel to convey the keys.

INDUSTRIAL RELEVANCE

Let us consider another scenario, where we want to send somebody a message and also provide a proof that the message is actually from us (a lot of harm can be done by providing bogus information, or rather, misinformation!). In order to keep a message private and also provide the authentication (that it is indeed from us), we can perform a special encryption on the plain text with our *private* key, then encrypt it again with the *public* key of the recipient. The recipient uses his private key to open the message and then use our public key to verify the authenticity. This technique is said to use **Digital Signatures**. The FIPS 186 standard is the **Digital Signature Standard** (DSS), which specifies the **Digital Signature Algorithm** (DSA).

There is another important encryption technique called the **One-way Function**. It is basically a non-reversible quick encryption method. The encryption is easy and fast, but the decryption is not. Suppose we send a document to recipient A and want to check at a later time whether the document has been tampered with. We can do so by running a one-way function, which produces a fixed length value called a **Hash** (also called the **Message Digest**). The hash is the unique signature of the document that can be sent along with the document. Recipient A can run the same one-way function to check whether the document has been altered.

The actual mathematical function used to encrypt and decrypt messages is called a cryptographic algorithm or cipher. This is only part of the system used to send and receive secure messages. This will become clearer further on when specific systems are discussed in detail.

As with most historical ciphers, the security of the message being sent relied on the algorithm itself remaining secret. This technique is known as a **Restricted Algorithm**. These have the following fundamental drawbacks:

(i) The algorithm obviously has to be restricted to only those people that you want to be able to decode your message. Therefore, a new algorithm must be invented for every discrete group of users.
(ii) A large or changing group of users cannot utilise them, as every time one user leaves the group, everyone must change algorithms.
(iii) If the algorithm is compromised in any way, a new algorithm must be implemented.

Because of these drawbacks, restricted algorithms are no longer popular, and have given way to key based algorithms.

Practically all modern cryptographic systems make use of a key. Algorithms that use a key system allow all details of the respective algorithms to be widely available. This is because all of the security lies in the key. With a key-based algorithm the plaintext is encrypted and decrypted by the algorithm which uses a certain key, and the resulting ciphertext is dependent on the key, and not the algorithm. This means that an eaves-dropper can have a complete copy of the algorithm in use, but without the specific key used to encrypt that message it is useless. A **nonce** is an arbitrary number that may be used only once in cryptographic algorithms, e.g., a timestamp, a counter, or a random number. A nonce differs with each transaction. It is often a random or pseudo-random number used in an authentication protocol to ensure that old communications cannot be reused for carrying out replay attacks.

This chapter deals with cryptographic **Algorithms**. The word *algorithm* comes from the name of the ninth century Persian mathematician, Abu Abdullah Muhammad ibn Musa al-Khwarizmi (780–850 A.D.), whose works introduced Indian numerals and algebraic concepts. Two general principles which guide the design of practical algorithms are *diffusion* and *confusion*. Diffusion means spreading out of the influence of a single plaintext digit over many ciphertext digits so as to hide the statistical structure of the plaintext. An extension of this idea is to spread the influence of a single key digit over many digits of ciphertext. Confusion means use of transformations which complicate dependence of the statistics of ciphertext on the statistics of plaintext.

DID YOU KNOW ?

According to **Kerckhoff's Principle**, a cryptosystem should remain secure even if everything about the system, except the key, is known. We will assume for our discussions that the adversary will know how the encryption and decryption algorithms work.

9.3 Operations used by Encryption Algorithms

LO 1

Although the methods of encryption/decryption have changed dramatically since the advent of computers, there are still only two basic operations that can be carried out on a piece of plaintext—substitution and transposition. The only real difference is that, earlier these were carried out with the alphabet; nowadays they are carried out on binary bits.

Substitution

Substitution operations replace bits in the plaintext with other bits decided upon by the algorithm, to produce ciphertext. This substitution then just has to be reversed to produce plaintext from ciphertext. This can be made increasingly complicated. For instance one plaintext character could correspond to one of a number of ciphertext characters (homophonic substitution), or each character of plaintext is substituted by a character of corresponding position in a length of another text (running cipher).

Example 9.4 Julius Caesar was one of the first to use substitution encryption to send messages to troops during the war. The substitution method he invented advances each character three spaces in the alphabet. Thus,

$$\text{THIS IS SUBSTITUTION CIPHER} \tag{9.1}$$

becomes

$$\text{WKLV LU VXEVWLWXWLRQ FLSKHU} \tag{9.2}$$

This is an example of **Caesar cipher**.

Example 9.5 The **Vigenère cipher** is a method of substitution encryption by using a series of different Caesar ciphers based on the letters of a keyword. A Vigenère cipher uses a Vigenère square as shown below.

Plaintext→ *↓Key*	*A*	*B*	*C*	*D*	...	*W*	*X*	*Y*	*Z*
A	A	B	C	D	...	W	X	Y	Z
B	B	C	D	E	...	X	Y	Z	A
C	C	D	E	F	...	Y	Z	A	B
:	:	:	:	:	...	:	:	:	:
X	X	Y	Z	A	...	T	U	V	W
Y	Y	Z	A	B	...	U	V	W	X
Z	Z	A	B	C	...	V	W	X	Y

Let us choose the key 'BAY'. Our key has only 3 letters, while we need to encrypt seven letters of the plaintext. So, we repeat our key as follows so that there are seven letters: B A Y B A Y B. Each one of the letters in the key will be used to encrypt (and decrypt) each corresponding letter in the plaintext. In order to encrypt, we choose the plaintext letter in the top row of the Vigenère square and choose the key letter in the first column. The ciphertext is the entry in the Vigenère square corresponding to the column of the plaintext and the row of the key. Using the Vigenère square above, we carry out the following encryption.

Plaintext Character	*Key Character*	*Ciphertext Character*
A	B	B
B	A	B
A	Y	Y
D	B	E
C	A	C
A	Y	Y
B	B	C

Thus, the ciphertext becomes: B B Y E C Y C.

Transposition

Transposition (or permutation) does not alter any of the bits in plaintext, but instead move their positions around within it. If the resultant ciphertext is then put through more transpositions, the end result is increasingly secure.

XOR

XOR is an exclusive-or operation. It is a Boolean operator such that if one of two bits is true, then so is the result, but if both are true or both are false then the result is false.

For example,

$$\begin{aligned} 0 \text{ XOR } 0 &= 0 \\ 1 \text{ XOR } 0 &= 1 \\ 0 \text{ XOR } 1 &= 1 \\ 1 \text{ XOR } 1 &= 0 \end{aligned} \tag{9.3}$$

A surprising amount of commercial software uses simple XOR functions to provide security, including the USA digital cellular telephone network and many office applications, and it is trivial to crack. However the XOR operation, as will be seen later in this paper, is a vital part of many advanced cryptographic algorithms when performed between long blocks of bits that also undergo substitution and/or transposition.

Example 9.6 Let us consider a **Multiplicative cipher**, which is a slightly more sophisticated technique than the Caesar cipher. Instead of adding a key, we multiply by the key number. Suppose, the key is, $k = 3$. We follow these steps for enciphering.

1. Replace each letter by the number it represents.
2. Multiply each number by the key, $k = 3$. Use modulo 26 arithmetic.
3. Replace the resulting number by letter equivalent.

Thus, for $k = 3$, we will have:

Plain	*A*	*B*	*C*	*D*	*E*	...	*Y*	*Z*
Number	1	2	3	4	5	…	25	26
$\times k$	3	6	9	12	15	…	75	78
Mod(26)	3	6	9	12	15	…	23	0/26
Cipher	**C**	**F**	**I**	**L**	**O**	…	**W**	**Z**

Note that, in order for this cipher to work, the key, k, and the modulus, n, should be relatively prime.

DON'T FORGET

Example 9.7 Let us consider an **Affine cipher**. Recall that an affine transform is of the type $Y = aX + b$. For an Affine cipher, we follow these steps for enciphering.

1. Replace each letter by the number it represents.
2. Multiply each number by a and then add b. Use modulo 26 arithmetic.
3. Replace the resulting number by a letter equivalent.

Note that we can, alternately, add b' first and then multiply by a'.

Plain	*A*	*B*	*C*	*D*	*E*	...	*Y*	*Z*
Number	1	2	3	4	5	…	25	26
+ 3	4	5	6	7	8		2	3
× 5	20	25	30	35	40	…	10	15
Mod(26)	20	25	4	9	14	…	10	15
Cipher	**T**	**Y**	**D**	**I**	**N**	…	**J**	**O**

DON'T FORGET

Definition 9.3 The **Avalanche effect** is a property of any encryption algorithm such that a small change in either the plaintext or the key produces a significant change in the ciphertext.

In the absence of the avalanche effect, a cryptanalyst can possibly make predictions about the input, given only the output. This may enable him/her to partially or completely break the cryptographic algorithm. Thus, the avalanche effect is a desirable condition for any cryptographic algorithm.

Learning Review

YOU ARE NOW READY TO ATTEMPT THE FOLLOWING QUESTIONS LINKED TO LO 1:

1. How many one-to-one affine ciphers can be constructed for the English alphabet? M
2. What is the total number of unique keys possible for the Playfair cipher? Do account for the trivial cases also. M
3. Consider a multiplicative cipher with $k = 2$. Build the cipher table and explain the problem with this cipher. M
4. In a message encrypted by the Affine cipher, the trigram NHM appears maximum number of times. Can you deduce the keys for this Affine cipher? D

If you have successfully solved the above problems, you have mastered LO 1. Congratulations!

CONGRATULATIONS LEVEL 1 DONE

The answers are available here. Scan to check
Or
Visit http://qrcode.flipick.com/index.php/622

Levels of Difficulty

- S **Simple:** Level 1 and Level 2 Category
- M **Medium:** Level 3 and Level 4 Category
- D **Difficult:** Level 5 and Level 6 Category

9.4 Symmetric (Secret Key) Cryptography

LO 2

Explain the concept of secret-key cryptography, Data Encryption Standard (DES), International Data Encryption Algorithm (IDEA) and RC Ciphers.

Symmetric Algorithms (or **Single Key Algorithms** or **Secret Key Algorithms**) have one key that is used both to encrypt and decrypt the message, hence their name. In order for the recipient to decrypt the message they need to have an identical copy of the key. This presents one major problem, the distribution of the keys. Unless the recipient can meet the sender in person and obtain a key, the key itself must be transmitted to the recipient, and is thus susceptible to eavesdropping. However, single key algorithms are fast and efficient, especially if large volumes of data need to be processed.

In symmetric cryptography, the two parties that exchange messages use the same algorithm. Only the key is changed from time to time. The same plaintext with a different key results in a different ciphertext. The encryption algorithm is available to the public, hence should be strong and well tested. The more powerful the algorithm, the lesser likely that an attacker will be able to decrypt the resulting cipher.

The size of the key is critical in producing a strong ciphertext. The U.S. National Security Agency (NSA) stated in the mid-1990s that a 40-bit length was acceptable to them (i.e., they could crack it sufficiently quickly!). Increasing processor speeds, combined with loosely-coupled multi-processor configurations, have brought the ability to crack such short keys within the reach of possible hackers. In 1998, it was suggested that in order to be strong, the key size needs to be at least 56 bits long. It was argued by an expert group as early as 1996 that 90 bits is a more appropriate length. Today, the most secure schemes use at least 128-bit keys, but commonly 256-bit keys, or even longer are preferred.

Symmetric cryptography provides a means of satisfying the requirement of message content security, because the content cannot be read without the secret key. There remains a risk of exposure, however, because neither party can be sure that the other party has not exposed the secret key to a third party (whether accidentally or intentionally).

Take a look..

Definition 9.4 *The **Playfair cipher** is a symmetric key encryption technique and is a digraph substitution cipher. Instead of single letters, as in the simple substitution cipher, the Playfair cipher encrypts pairs of letters (digraphs).*

This Playfair cipher technique is relatively harder to break than the simple substitution cipher. This is because the simple frequency analysis used for the substitution ciphers does not work with it. The frequency analysis of digraphs is possible, but considerably more difficult (well 'more difficult' is a relative term!). For the English alphabet, there are 600 possible digraphs rather than the 26 possible monographs. The Playfair algorithm is based on the use of a 5×5 matrix of letters constructed using a keyword.

Symmetric cryptography can also be used to address the integrity and authentication requirements. The sender creates a summary of the message, or **Message Authentication Code (MAC)**, encrypts it with the secret key, and sends that with the message. The recipient then re-creates the MAC, decrypts the MAC that was sent, and compares the two. If they are identical, then the message that was received must have been identical with that which was sent.

DID YOU KNOW ?

As mentioned earlier, a major difficulty with symmetric schemes is that the secret key has to be possessed by both parties, and hence has to be transmitted from whoever creates it to the other party. Moreover, if the key is compromised, all of the message transmission security measures are undermined. The steps taken to provide a secure mechanism for creating and passing on the secret key are referred to as **Key Management**.

The technique does not adequately address the non-repudiation requirement, because both parties have the same secret key. Hence each is exposed to the risk of fraudulent falsification of a message by the other, and a claim by either party not to have sent a message is credible, because the other may have compromised the key.

There are two types of symmetric algorithms, block ciphers and stream ciphers.

Definition 9.5 Block Ciphers usually operate on groups of bits called blocks. Each block is processed a multiple number of times. In each round the key is applied in a unique manner. The more the number of iterations, the longer is the encryption process, but results in a more secure ciphertext.

Example 9.8 A **Feistel cipher** is a symmetric cryptographic structure used in the construction of block ciphers. It is named after the German-born cryptographer Horst Feistel. The Feistel structure has the advantage that both the encryption and decryption operations are very similar, thereby substantially reducing the size of the code or hardware required to implement such a cipher. In the classic Feistel structure, one half of the data block is used to modify the other half of the data block. Subsequently, the halves are swapped.

Definition 9.6 Stream Ciphers operate on plaintext one bit (or one byte) at a time. Plaintext is streamed as raw bits through the encryption algorithm. While a block cipher will produce the same ciphertext from the same plaintext using the same key, a stream cipher will not. The ciphertext produced by a stream cipher will vary under the same conditions.

Example 9.9 Let us look at a simple stream cipher. Consider a plaintext stream $P = P_1 P_2 P_3 \ldots.$, and a key stream $K = k_1 k_2 k_3 \ldots$

The encryption is done as follows: $C_1 = \mathrm{E}(P_1, k_1)$, $C_2 = \mathrm{E}(P_2, k_2)$, $C_3 = \mathrm{E}(P_3, k_3)$, …

The resulting ciphertext stream from the stream cipher would be $C = C_1 C_2 C_3 \ldots$

INDUSTRIAL RELEVANCE

Example 9.10 Consider the stream cipher A5/1 used in the Global System for Mobile (GSM) communication. The A5/1 stream cipher uses three Linear Feedback Shift Registers (LFSRs) with irregular clocking. The details of the three LFSRs are given below.

LFSR number	**Length in bits**	**Feedback polynomial**
1	19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$
2	22	$x^{22} + x^{21} + 1$
3	23	$x^{23} + x^{22} + x^{21} + x^{8} + 1$

The registers are clocked in a stop-and-go fashion using a majority rule. Each register has an associated clocking bit. At each cycle, the clocking bit of all three registers is examined and the majority bit is determined. If the clocking bit agrees with the majority bit, the register is clocked.

The A5/1 is known to be a weak cipher and several attacks have been published.

Let us now consider a basic question: How long should a key be? There is no single answer to this question. It depends on the specific situation. To determine how much security one needs, the following questions must be answered:

1. How much is the data to be protected worth?
2. How long does it need to be secure?
3. What are the resources available to the cryptanalyst/hacker?

A customer list might be worth Rs. 1000, an advertisement data might be worth Rs. 50,000 and the master key for a digital cash system might be worth millions. In the world of stock markets, the secrets have to be kept for a couple of minutes. In the newspaper business today's secret is tomorrow's headlines. The census data of a country have to be kept secret for months (if not years). Corporate trade secrets are interesting to rival companies and military secrets are interesting to rival militaries. Thus, the security requirements can be specified in these terms. For example, one may require that the key length must be such that there is a probability of 0.0001% that a hacker with the resources of Rs. 1 million could break the system in 1 year, assuming that the technology advances at a rate of 25% per annum over that period. The minimum key requirements for different applications are listed in Table 9.1. This table should be used as a guideline only.

Table 9.1 Minimum key requirements for different applications

Type of information	*Lifetime*	*Minimum key length*
Tactical military information	Minutes/hours	56-64 bits
Product announcements	Days/weeks	64 bits
Interest rates	Days/weeks	64 bits
Trade secrets	decades	112 bits
Nuclear bomb secrets	> 50 years	128 bits
Identities of spies	> 50 years	128 bits
Personal affairs	> 60 years	> 128 bits
Diplomatic embarrassments	> 70 years	> 256 bits

Future computing power is difficult to estimate. A rule of thumb is that the efficiency of computing equipment divided by price doubles every 18 months, and increases by a factor of 10 every five years. Thus, in 50 years the fastest computer will be 10 billion times faster than today's! These numbers refer to the general-purpose computers. We cannot predict what kind of specialised crypto-system breaking computers might be developed in the years to come.

Two symmetric algorithms, both block ciphers, will be discussed in this chapter. These are the **Data Encryption Standard** (DES) and the **International Data Encryption Algorithm** (IDEA).

9.5 Data Encryption Standard (DES)

LO 2

INDUSTRIAL RELEVANCE

DES, an acronym for the Data Encryption Standard, is the name of the Federal Information Processing Standard (FIPS) 46-3, which describes the data encryption algorithm (DEA). The DEA is also defined in the ANSI standard X9.32. ANSI X9.9 standard specifies a DES-based message authentication code (MAC) algorithm for wholesale banking. ANSI X9.23 standard addresses message formatting and representation issues related to the use of DES encryption in wholesale banking transactions.

Created by IBM, DES came about due to a public request by the US National Bureau of Standards (NSB) requesting proposals for a standard cryptographic algorithm that satisfied the following criteria.

(i) Provides a high level of security
(ii) The security depends on keys, not the secrecy of the algorithm
(iii) The security is capable of being evaluated
(iv) The algorithm is completely specified and easy to understand
(v) It is efficient to use and adaptable
(vi) Must be available to all users
(vii) Must be exportable

DEA is essentially an improvement of the algorithm Lucifer developed by IBM in the early 1970s. The US National Bureau of Standards published the DES in 1975. While the algorithm was basically designed by IBM, the NSA and NBS (now NIST) played a substantial role in the final stages of the development. The DES has been extensively studied since its publication and is the best known and widely used symmetric algorithm in the world.

The DEA has a 64-bit block size and uses a 56-bit key during execution (8 parity bits are stripped off from the full 64-bit key). The DEA is a symmetric cryptosystem, specifically a 16-round Feistel cipher and was originally designed for implementation in hardware. When used for communication, both sender and receiver must know the same secret key, which can be used to encrypt and decrypt the message, or to generate and verify a message authentication code (MAC). The DEA can also be used for single-user encryption, such as to store files on a hard disk in encrypted form. In a multi-user environment, secure key distribution may be difficult; public-key cryptography provides an ideal solution to this problem.

NIST has re-certified DES (FIPS 46-1, 46-2, 46-3) every five years. FIPS 46-3 reaffirms DES usage as of October 1999, but single DES is permitted only for legacy systems. FIPS 46-3 includes a definition of triple-DES (TDEA, corresponding to X9.52). DES and triple-DES have now been replaced with the Advanced Encryption Standard (AES).

DES has now been in world-wide use for over 30 years, and due to the fact that it is a defined standard means that any system implementing DES can communicate with any other system using it, DES is used in banks and businesses all over the world, as well as in networks (as Kerberos) and to protect the password file on UNIX Operating Systems (as CRYPT).

DES Encryption

DES is a symmetric, block-cipher algorithm with a key length of 64 bits, and a block size of 64 bits (i.e., the algorithm operates on successive 64 bit blocks of plaintext). Being symmetric, the same key is used for encryption and decryption, and DES also uses the same algorithm for encryption and decryption.

First a transposition is carried out according to a set table (the initial permutation), the 64-bit plaintext block is then split into two 32-bit blocks, and 16 identical operations called rounds are carried out on each half. The two halves are then re-joined together, and the reverse of the initial permutation is carried out. The purpose of the first transposition is not clear, as it does not affect the security of the algorithm, but is thought to be for the purpose of allowing plaintext and ciphertext to be loaded into 8-bit chips in byte-sized pieces.

In any round, only one half of the original 64-bit block is operated on. The rounds alternate between the two halves. One round in DES consists of the following.

Key transformation

The 64-bit key is reduced to 56 by removing every eighth bit (these are sometimes used for error checking). Sixteen different 48-bit sub-keys are then created—one for each round. This is achieved by splitting the 56-bit key into two halves, and then circularly shifting them left by 1 or 2 bits, depending on the round. After this, 48 of the bits are selected. Because they are shifted, different groups of key bits are used in each subkey. This process is called a compression permutation due to the transposition of the bits and the reduction of the overall size.

Expansion permutation

After the key transformation, whichever half of the block is being operated on undergoes an expansion permutation. In this operation, the expansion and transposition are achieved simultaneously by allowing the first and fourth bits in each 4 bit block to appear twice in the output, i.e., the fourth input bit becomes the fifth and seventh output bits (see Fig 9.2).

The expansion permutation achieves 3 things: Firstly, it increases the size of the half-block from 32 bits to 48, the same number of bits as in the compressed key subset, which is important as the next operation is to XOR the two together. Secondly, it produces a longer string of data for the substitution operation that subsequently compresses it. Lastly, and most importantly, because in the subsequent substitutions the first and fourth bits appear in two S-boxes (described shortly), they affect two substitutions. The effect of this is that the dependency of the output bits on the input bits increases rapidly, and so does the security of the algorithm.

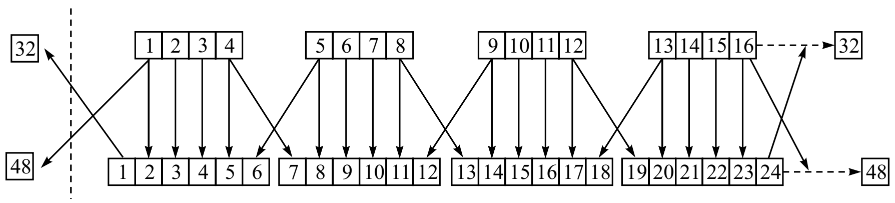

Fig. 9.2 The expansion permutation.

XOR

The resulting 48-bit block is then XORed with the appropriate subset key for that round.

Substitution

The next operation is to perform substitutions on the expanded block. There are eight substitution boxes, called S-boxes (see Fig 9.3). The first S-box operates on the first 6 bits of the 48-bit expanded block, the 2nd

S-box on the next six, and so on. Each S-box operates from a table of 4 rows and 16 columns, each entry in the table is a 4-bit number. The 6-bit number the S-box takes as input is used to look up the appropriate entry in the table in the following way. The 1st and 6th bits are combined to form a 2-bit number corresponding to a row number, and the 2nd to 5th bits are combined to form a 4-bit number corresponding to a particular column. The net result of the substitution phase is eight 4-bit blocks that are then combined into a 32-bit block.

It is the non-linear relationship of the S-boxes that really provide DES with its security, all the other processes within the DES algorithm are linear, and as such relatively easy to analyse.

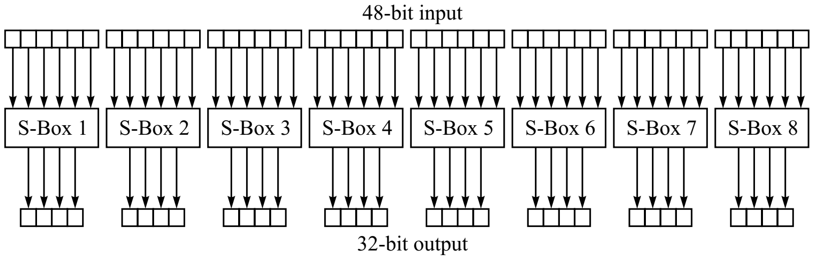

Fig. 9.3 The S-box substitution.

Permutation

The 32-bit output of the substitution phase then undergoes a straightforward transposition using a table sometimes known as the P-box.

After all the rounds have been completed, the two 'half-blocks' of 32 bits are recombined to form a 64-bit output, the final permutation is performed on it, and the resulting 64-bit block is the DES encrypted ciphertext of the input plaintext block.

DES Decryption

Decrypting DES is very easy (if one has the correct key!). Thanks to its design, the decryption algorithm is identical to the encryption algorithm—the only alteration that is made, is that to decrypt DES ciphertext, the subsets of the key used in each round are used in reverse, i.e., the sixteenth subset is used first.

Security of DES

Unfortunately, with advances in the field of cryptanalysis and the huge increase in available computing power, DES is no longer considered to be very secure. There are algorithms that can be used to reduce the number of keys that need to be checked, but even using a straightforward brute-force attack and just trying every single possible key there are computers that can crack DES in a matter of minutes. It is rumoured that the US National Security Agency (NSA) can crack a DES encrypted message in 3–15 minutes.

If a time limit of 2 hours to crack a DES encrypted file is set, then you have to check all possible keys (2^{56}) in 2 hours, which is roughly 5 trillion keys per second. While this may seem like a huge number, consider that a $10 Application-Specific Integrated Circuits (ASICs) chip can test 200 million keys per second, and many of these can be paralleled together. It is suggested that a $10 million investment in ASICs would allow a computer to be built that would be capable of breaking a DES encrypted message in 6 minutes.

DES can no longer be considered a sufficiently secure algorithm. If a DES-encrypted message can be broken in minutes by supercomputers today, then the rapidly increasing power of computers means that it will be a trivial matter to break DES encryption in the future (when a message encrypted today may still need to be secure). An extension of DES called DESX is considered to be virtually immune to an exhaustive key search.

Advanced Encryption Standard (AES)

AES is a symmetric-key block cipher. It is based on the **Rijndael cipher** developed by two cryptographers, Joan Daemen and Vincent Rijmen. Rijndael is a family of ciphers with different key and block sizes. AES is a symmetric-key algorithm, where the same key is used for both encrypting and decrypting the data. For AES, three types of Rijndael ciphers are used, each with a block size of 128 bits, but three different key lengths: 128, 192 and 256 bits. Rijndael cipher was designed to have the following characteristics.

(i) Resistance against known attacks
(ii) Speed and efficient coding over a wide range of platforms
(iii) Design simplicity

AES is based on the principle known as a substitution-permutation network, where a combination of substitution and permutation is used. AES does not use the Feistel cipher structure. It may be recalled that in the classic Feistel structure, one half of the data block is used to modify the other half of the data block. Subsequently, the halves are swapped. In AES, the entire data block is processed in parallel during each round using substitutions and permutations.

9.6 International Data Encryption Algorithm (IDEA)

LO 2

IDEA was created in its first form by Xuejia Lai and James Massey in 1990, and was called the Proposed Encryption Standard (PES). In 1991, Lai and Massey strengthened the algorithm against differential cryptanalysis and called the result Improved PES (IPES). The name of IPES was changed to International Data Encryption Algorithm (IDEA) in 1992. IDEA is perhaps best known for its implementation of PGP (Pretty Good Privacy).

The Algorithm

IDEA is a symmetric, block-cipher algorithm with a key length of 128 bits, a block size of 64 bits, and as with DES, the same algorithm provides encryption and decryption.

IDEA consists of eight rounds using 52 sub-keys. Each round uses six sub-keys, with the remaining four being used for the output transformation. The sub-keys are created as follows.

Firstly, the 128-bit key is divided into eight 16-bit keys to provide the first eight sub-keys. The bits of the original key are then shifted 25 bits to the left, and then it is again split into eight sub-keys. This shifting and then splitting is repeated until all 52 sub-keys (SK1-SK52) have been created.

The 64-bit plaintext block is firstly split into four blocks (B1-B4). A round then consists of the following steps (OB stands for output block).

OB1 = B1 * SK1 (multiply 1st sub-block with 1st sub-key)
OB2 = B2 + SK2 (add 2nd sub-block to 2nd sub-key)
OB3 = B3 + SK3
OB4 = B4 * SK4 (multiply 3rd sub-block with 3rd sub-key)
OB5 = OB1 XOR OB3 (XOR results of steps 1 and 3)
OB6 = OB2 XOR OB4
OB7 = OB5 * SK5 (multiply result of step 5 with 5th sub-key)
OB8 = OB6 + OB7 (add results of steps 5 and 7)
OB9 = OB8 * SK6 (multiply result of step 8 with 6th sub-key)
OB10 = OB7 + OB9

OB11 = OB1 XOR OB9 (XOR results of steps 1 and 9)
OB12 = OB3 XOR OB9
OB13 = OB2 XOR OB10
OB14 = OB4 XOR OB10
The input to the next round is the four sub-blocks OB11, OB13, OB12 and OB14.

After the eighth round, the four final output blocks (F1-F4) are used in a final transformation to produce four sub-blocks of ciphertext (C1-C4) that are then re-joined to form the final 64-bit block of ciphertext.

C1 = F1 * SK49
C2 = F2 + SK50
C3 = F3 + SK51
C4 = F4 * SK52
Ciphertext = C1 & C2 & C3 & C4.

Security provided by IDEA

Not only is IDEA approximately twice as fast as DES, but it is also considerably more secure. Using a brute-force approach, there are 2^{128} possible keys. If a billion chips that could each test 1 billion keys a second were used to try and crack an IDEA-encrypted message, it would take them 10^{13} years which is considerably longer than the age of the universe! Being a fairly new algorithm, it is possible that a better attack than brute-force, which, when coupled with much more powerful machines in the future may be able to crack a message. However, for a long way into the future, IDEA seems to be an extremely secure cipher.

9.7 RC Ciphers

LO 2

The RC ciphers were designed by Ron Rivest for the RSA Data Security. RC stands for *Ron's Code* or *Rivest Cipher*. **RC2** was designed as a quick-fix replacement for DES, which is more secure. It is a block cipher with a variable key size that has a propriety algorithm. RC2 is a variable-key-length cipher. However, when using the Microsoft Base Cryptographic Provider, the key length is hard-coded to 40 bits. When using the Microsoft Enhanced Cryptographic Provider, the key length is 128 bits by default and can be in the range of 40 to 128 bits in 8-bit increments.

RC4 was developed by Ron Rivest in 1987. It is a variable-key-size stream cipher. The details of the algorithm have not been officially published, but are known. The algorithm is extremely easy to describe and program. Just like RC2, 40-bit RC4 is supported by the Microsoft Base Cryptographic provider, and the Enhanced provider allows keys in the range of 40 to 128 bits in 8-bit increments.

RC4 generates a pseudorandom stream of bits. As with any stream cipher, this key-stream can be used for encryption by XORing it with the plaintext using bit-wise operation. The decryption is performed in the same way. To generate the key-stream, the cipher makes use of a secret internal state which consists of two parts:

(i) A permutation of all 256 possible bytes (denoted by S).
(ii) Two 8-bit index-pointers (denoted by i and j).

Initially, the entries of S are set equal to the values from 0 through 255 in ascending order, that is, $S[0] = 0, S[1] = 1,\dots, S[255] = 255$. Based on a temporary vector, T (typically between 40 and 256-bit long), the contents of S are swapped. Because the only operation on S is a swap, the only effect is a permutation. S still contains all the numbers from 0 through 255. Once this has been completed, the stream of bits is generated using the *pseudo-random generation algorithm* (PRGA).

RC5 is a block cipher designed for speed. The block size, key size and the number of iterations are all variables. In particular, the key size can be as large as 2,048 bits.

All the encryption techniques discussed so far (DES, AES, IDEA, RC Ciphers) belong to the class of symmetric cryptography. We will next study the class of **Asymmetric Cryptographic Techniques**.

Learning Review

YOU ARE NOW READY TO ATTEMPT THE FOLLOWING QUESTIONS LINKED TO LO 2:

1. What is the difference between a message authentication code (MAC) and a one-way hash function? S
2. Consider RC4 with the internal state, S, and the two indices i and j. How many bits are used to store the internal state? Determine, in bits, how much information is represented by the state. S
3. List important design considerations for a stream cipher. S
4. What purpose do the S-boxes serve in DES? S
5. Suppose we launch a chosen-ciphertext attack on a *linear* block cipher of length 128 bits. This linear cipher satisfies: $E(P_1 \oplus P_2, k) = E(P_1, k) \oplus E(P_2, k)$, where $\oplus$ represents the XOR function, P_1 and P_2 represent the plaintext, $E(P_i, k)$ represents the ciphertext using the encryption algorithm, E. How many chosen-ciphertexts would be needed to break the cryptosystem? D

If you have successfully solved the above problems, you have mastered LO 2. Congratulations!

CONGRATULATIONS LEVEL 1 DONE

The answers are available here. Scan to check
Or
Visit http://qrcode.flipick.com/index.php/623

9.8 Asymmetric (Public-Key) Algorithms

LO 3

Underline the concept of public-key cryptography and implement the Rivest, Shamir and Adleman (RSA) algorithm, Pretty Good Privacy (PGP) and one-way Hashing.

Public-key Algorithms are asymmetric, that is, the key used to encrypt the message is different from the key used to decrypt the message. The encryption key, known as the **public key** is used to encrypt a message, but the message can only be decoded by the person that has the decryption key, known as the **private key**.

This type of algorithm has a number of advantages over traditional symmetric ciphers. It means that the recipient can make their public key widely available—anyone wanting to send them a message uses the algorithm and the recipient's public key to do so. An eavesdropper may have both the algorithm and the public key, but will still not be able to decrypt the message. Only the recipient, with their private key can decrypt the message.

DID YOU KNOW ? A disadvantage of public-key algorithms is that they are more computationally intensive than symmetric algorithms, and therefore encryption and decryption take longer. This may not be significant for a short text message, but certainly is for long messages or audio/video clips.

The **Public-Key Cryptography Standards** (PKCS) are specifications produced by RSA Laboratories in cooperation with secure systems developers worldwide for the purpose of accelerating the deployment of public-key cryptography. First published in 1991 as a result of meetings with a small group of early adopters of public-key technology, the PKCS documents have become widely referenced and implemented. Contributions from the PKCS series have become part of many formal and de facto standards, including ANSI X9 documents, PKIX, SET, S/MIME, and SSL.

The following section describes the RSA algorithm, followed by the Pretty Good Privacy (PGP) hybrid algorithm.

9.9 The RSA Algorithm

LO 3

RSA, named after its three creators—Rivest, Shamir and Adleman, was the first effective public-key algorithm, and for years has withstood intense scrutiny by cryptanalysts all over the world. Unlike symmetric key algorithms, where, as long as one presumes that an algorithm is not flawed, the security relies on having to try all possible keys, public-key algorithms rely on being computationally unfeasible to recover the private key from the public key.

RSA relies on the fact that it is easy to multiply two large prime numbers together, but extremely hard (i.e., time consuming) to factor them back from the result. Factoring a number means finding its prime factors, which are the prime numbers that need to be multiplied together in order to produce that number. For example,

$$\begin{aligned} 10 &= 2 \times 5 \\ 60 &= 2 \times 2 \times 3 \times 5 \\ 2^{113} - 1 &= 3391 \times 23279 \times 65993 \times 1868569 \times 1066818132868207 \end{aligned}$$

The Algorithm

Two very large prime numbers, normally of equal length, are randomly chosen and then multiplied together.

$$N = A \times B \tag{9.4}$$

$$T = (A - 1) \times (B - 1) \tag{9.5}$$

A third number is then also chosen randomly as the public key (E) such that it has no common factors (i.e., it is relatively prime) with T. The private key (D) is then

$$D = E^{-1} \bmod T \tag{9.6}$$

To encrypt a block of plaintext (M) into ciphertext (C):

$$C = M^E \bmod N \tag{9.7}$$

To decrypt

$$M = C^D \bmod N \tag{9.8}$$

Example 9.11 Consider the following implementation of the RSA algorithm.

$$\begin{aligned} 1^{st} \text{ prime } (A) &= 37 \\ 2^{nd} \text{ prime } (B) &= 23 \end{aligned}$$

So,

$$N = 37 \times 23 = 851$$
$$T = (37 - 1) \times (23 - 1) = 36 \times 23 = 792$$

E must have no factors other than 1 in common with 792.
E (public key) could be 5.
D (private key) = 5^{-1} mod 792 = 317
To encrypt a message (M) of the character 'G':
If G is represented as 7 (7th letter in alphabet), then $M = 7$.
C (ciphertext) = 7^5 mod 851 = 638
To decrypt:
$M = 638^{317}$ mod 851 = 7.

Example 9.12 The previous example was a toy-example to illustrate the working of RSA algorithm. Let us consider a slightly more realistic example of RSA algorithm.

1st prime (A) = 3283807
2nd prime (B) = 7365557

So, $N = 3283807 \times 7365557 = 24187067635499$
$T = (3283807 - 1) \times (7365557 - 1) = 24187056986136$.

E must have no factors other than 1 in common with 24187056986136.
E (public key) could be 11828950082903.
Verify that gcd(24187056986136, 11828950082903) = 1.

DID YOU KNOW ?

D (private key) = $(11828950082903)^{-1}$ mod 792 = 6376223.
In real-life, A and B contain more than 100 digits!

Prime Numbers

The RSA algorithm relies on large prime numbers. Do we actually have arbitrarily sized prime numbers? To answer this question, we first define the **Prime Counting Function.**

Definition 9.7 The **Prime Counting Function** $\pi(n)$ counts the number of primes that are less than or equal to n.

Example 9.13 Table 9.2 illustrates the prime counting function.

Table 9.2 The Prime Counting Function

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
$\pi(n)$	1	2	2	3	3	4	4	4	4	5	5	6	6	6	…

From the table we note that we increase the value of $\pi(n)$ only when we encounter the next prime number. The function $\pi(n)$ is monotonically increasing.

Example 9.14 Do we have infinite number of primes? Let us assume that we have only finite number of prime numbers. Let these be $p_1, p_2, \ldots, p_n$. Consider the number $m = p_1, p_2 \ldots p_n + 1$. Because m is bigger than any prime, it must be a composite number. Hence it should be divisible by some prime number. However, it is not divisible by p_1 because we get the remainder '1' after dividing m by p_1. Similarly, it is not divisible by any p_i, $i = 1, 2, \ldots n$. Thus we get a contradiction and hence our assumption was incorrect. Therefore, we have infinitely many primes.

Theorem 9.1 The **Prime Density Theorem** states that

$$\lim_{n\to\infty} \frac{\pi(n)\ln(n)}{n} = 1 \tag{9.9}$$

Typically, $\pi(n) \approx \dfrac{n}{\ln(n)}$ is a good approximation. This tells us that we do have lots of prime numbers.

There are two common methods of generating large prime numbers.

(i) Construct provable primes
(ii) Randomly choose large odd numbers and apply primality tests.

Example 9.15 Let us use the prime density theorem to estimate the number of 50-digit prime numbers. This is obtained by subtracting the total number of primes up to 49 digits from the total number of primes up to 50 digits.

Thus, the number of 50-digit prime numbers $\approx \pi(10^{50}) - \pi(10^{49})$

$$= \frac{10^{50}}{\ln(10^{50})} - \frac{10^{49}}{\ln(10^{49})} = 7.79 \times 10^{47}.$$

Out of curiosity, let us randomly pick a 50-digit odd number. What is the probability that it will be a prime number?

The fraction of 50-digit numbers that are prime $\approx (7.79 \times 10^{47} / 10^{50}) \approx 1/128$.

Since we are excluding all even numbers, the odds become twice as much (no pun intended!). Hence, the probability that a 50-digit odd number selected randomly turns out to be a prime number $\approx 1/64$.

Theorem 9.2 Fermat's Little Theorem states that suppose p is a prime and a is an integer that is not a multiple of p, then $a^{p-1} \equiv 1 \pmod p$.

Definition 9.8 The **Primality Decision Problem**: Given a positive integer n, decide whether n is a prime or not.

One of the direct outcomes of Fermat's little theorem is a simple primality test. If n is a positive integer and $a^{n-1} \neq 1 \pmod n$, for some number a, $1 < a < n - 1$, then n is composite. In order to describe the computational complexity of primality tests we require the following definitions.

Definition 9.9 A **Turing Machine** is an imaginary computing device that yields a primitive but sufficiently general computational model. It is typically represented by a **Finite State Machine**.

Definition 9.10 An algorithm is called **Polynomial Time** if its worst case running time function is polynomial in the input size. Any algorithm whose running time cannot be bounded by a polynomial is called **Super Polynomial Time**.

Definition 9.11 The **Complexity Class *P*** refers to the class of decision problems $D \subseteq \{0, 1\}$ that is solvable in polynomial time by a deterministic Turing machine.

Definition 9.12 The **Complexity Class *NP*** refers to the class of decision problems $D \subseteq \{0, 1\}$ for which a 'yes' answer can be verified in polynomial time, given some extra information called a certificate or witness. The **Complexity Class co-*NP*** refers to the class of decision problems $D \subseteq \{0, 1\}$ for which a 'no' answer can be verified in polynomial time, given some extra information called a certificate or witness.

The primality test algorithms can either be deterministic or probabilistic. Only a few deterministic primality testing functions are efficient, i.e., run in polynomial time. The probabilistic primality testing algorithms are much more efficient. Knowing efficient primality testing algorithms implies that the primality decision problem is in the complexity class *P*.

An important open question in complexity theory is whether $NP = P$ or $NP \neq P$? It is however widely believed that $NP \neq P$. This is also supported by our intuition that solving a problem is more involved than verifying the solution. Also, if $NP = P$, there would be no computationally secure cryptographic system in the mathematical sense.

Example 9.16 Let us use Fermat's little theorem to check whether $n = 409$ is a prime. We use a probabilistic approach.

(i) Randomly choose a base $a = 237$. Here $n = 409$.
$a^{n-1} = 237^{408} \pmod{409} \equiv 1$.
So, it passes the test for this randomly chosen base.

(ii) Again, randomly choose another base $a = 356$.
$a^{n-1} = 356^{408} \pmod{409} \equiv 1$.
So, it again passes the test for this randomly chosen base.

(iii) Again, randomly choose another base $a = 202$.
$a^{n-1} = 202^{408} \pmod{409} \equiv 1$.
So, it again passes the test for this randomly chosen base.
We are getting convinced (probabilistically)! Let us try one more time.

(iv) Again, randomly choose another base $a = 105$.
$a^{n-1} = 105^{408} \pmod{409} \equiv 1$.
So, it again passes the test for this randomly chosen base.
So, probably 409 a prime.
Can you show by another method that indeed 409 is a prime number?

DON'T FORGET

Let us see how RSA can be used for generating digital signatures. Suppose Bob requires Alice to sign a digital document, P. To do this, Alice will use her private key, D to encrypt P and send the digital signature, $s = D(P)$. Bob will then use the corresponding public key $E = D^{-1}$ to obtain $D^{-1}(s) = D^{-1}(D(P)) = P$ (original document). In case, $D^{-1}(s) \neq P$, Bob rejects the signature as invalid.

Example 9.17 Consider an RSA-based digital signature scheme. Suppose the RSA parameters are $A = 23$ and $B = 79$ and the keys are $D = 103$ and $E = 883$.

Let the digital message be $P = 1776$.

We first compute the digital signature, $s = D(P) = P^D \bmod N$.

Here, $N = A \times B = 1817$.

Thus, $s = 1776^{103} \pmod{1817} = 1790$.

To verify the digital signature, we must use the public key, E.

$D^{-1}(s) = s^E \bmod N = 1790^{883} \pmod{1817} = 1776$.

Thus, the signature is verified.

Security of RSA

The security of RSA algorithm depends on the ability of the hacker to factorise numbers. New, faster and better methods for factoring numbers are constantly being devised. The current best for long numbers is the *Number Field Sieve*. Prime Numbers of a length that was unimaginable a mere decade ago are now factored easily. Obviously the longer number is, the harder it is to factor, and so the better the security of RSA. As theory and computers improve, larger and larger keys will have to be used. The disadvantage in using extremely long keys is the computational overhead involved in encryption/decryption. This will only become a problem if a new factoring technique emerges that requires keys of such lengths to be used that the necessary key length increases much faster than the increasing average speed of computers utilising the RSA algorithm. The security of RSA cryptosystems may get threatened in the following two ways.

(i) If a polynomial time algorithm for factoring primes gets discovered, or
(ii) If a new revolution in computer hardware/distributed computing/quantum computing takes place.

DID YOU KNOW ?

In 1997, a specific assessment of the security of 512-bit RSA keys shows that one may be factored for less than \$1,000,000 in cost and eight months of effort. It is therefore believed that 512-bit keys provide insufficient security for anything other than short-term needs. RSA Laboratories currently recommends key sizes of 768 bits for personal use, 1024 bits for corporate use, and 2048 bits for extremely valuable keys like the root-key pair used by a certifying authority. Security can be increased by changing a user's keys regularly and it is typical for a user's key to expire after two years (the opportunity to change keys also allows for a longer length key to be chosen).

Even without using huge keys RSA is about 1000 times slower to encrypt/decrypt than DES, this has resulted in it not being widely used as a stand-alone cryptography system. However, it is used in many hybrid cryptosystems such as PGP. The basic principle of hybrid systems is to encrypt plaintext with a symmetric algorithm (usually DES or IDEA); the symmetric algorithm's key is then itself encrypted with a public-key algorithm such as RSA. The RSA-encrypted key and symmetric algorithm-encrypted message are then sent to the recipient, who uses his private RSA key to decrypt the symmetric algorithm's key, and then that key to decrypt the message. This is considerably faster than using RSA throughout, and allows a different symmetric key to be used each time, considerably enhancing the security of the symmetric algorithm.

INDUSTRIAL RELEVANCE

RSA's future security relies solely on advances in factoring techniques. Barring an astronomical increase in the efficiency of factoring techniques, or available computing power, the 2048-bit key will ensure very secure protection into the foreseeable future. For instance an Intel Paragon, which can achieve 50,000 mips (million operations per second), would take a million years to factor a 2048-bit key using current techniques. The ANSI X9.31 standard specifies a signature mechanism based on an RSA signature algorithm. ISO 11166 is a standard for banking key management. Part 2 of this standard prescribes the RSA algorithm for both encryption and digital signatures.

9.10 Pretty Good Privacy (PGP)

LO 3

Pretty Good Privacy (PGP) is a hybrid cryptosystem that was created by Phil Zimmerman and released onto the Internet as a freeware program in 1991. PGP is not a new algorithm in its own right, but rather a series of other algorithms that are performed along with a sophisticated protocol. PGP's intended use was for e-mail security, but there is no reason why the basic principles behind it could not be applied to any type of transmission.

PGP and its source code is freely available on the Internet. This means that since its creation PGP has been subjected to an enormous amount of scrutiny by cryptanalysts, who have yet to find an exploitable fault in it.

PGP has four main modules: a symmetric cipher–IDEA for message encryption; a public-key algorithm–RSA to encrypt the IDEA key and hash values; a one-way hash function—MD5 for signing and a random number generator.

The fact that the body of the message is encrypted with a symmetric algorithm (IDEA) means that PGP generated e-mails are a lot faster to encrypt and decrypt than using simple RSA. The key for the IDEA module is randomly generated each time as a one-off session key, this makes PGP very secure, as even if one message is cracked, all previous and subsequent messages would remain secure. This session key is then encrypted with the public key of the recipient using RSA. Given that keys up to 2048 bits long can be used, this is extremely secure. MD5 can be used to produce a hash of the message, which can then be signed by the sender's private key. Another feature of PGP's security, is that the user's private key is encrypted using a hashed pass-phrase rather than simply a password, making the private key extremely resistant to copying even with access to the user's computer.

Generating true random numbers on a computer is notoriously hard, PGP tries to achieve randomness by making use of the keyboard latency when the user is typing. This means that the program measures the gap of time between each key-press. While at first this may seem to be distinctly non-random, it is actually fairly effective—people take longer to hit some keys than others, pause for thought, make mistakes and vary their overall typing speed on all sorts of factors such as knowledge of the subject and tiredness. These measurements are not actually used directly, but used to seed a pseudo-random number generator. There are other ways of generating random numbers, but to be much better than this gets very complex.

DID YOU KNOW ?

PGP uses a very clever, but complex, protocol for key management. Each user generates and distributes their public key. If James is happy that a person's public key belongs to who it claims to belong to, then he can sign that person's public key, showing others they believe this