IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 14, Issue 5 (Sep. - Oct. 2013), PP 01-31
www.iosrjournals.org

A Logical Issue for the Internet Community to Address Is That of Effective Cryptography Standards for the Conduct of Business and Personal Communications

Bappaditya Jana

Abstract: *Many organizations are working hard to secure themselves from the growing threats of message hacking through various trends in cryptography. Yet the headlines are dominated with the latest news of message passing disaster more frequently than any time before. This document intends to review this problem and propose several possible solutions. The cryptographic industry has been responding to these threats with ever-quicker responses to the rapid onslaught of malicious techniques, while corporations establish strict cryptographic techniques.*

Placing an organizations cryptographic techniques at the desktop level is like closing all the doors in a house Â¦..while leaving windows and other entry points open. The present document discusses various cryptographic techniques of all times such as the three basic algorithms namely private key algorithm,, public key algorithm and the hash functions. The need for having three encryption techniques has also been encrypted .A detailed discussion has been done on the classical cryptography and the drawbacks of the classical cryptography to ensure the need for going to new trends in cryptography like quantum cryptography, elliptic curve cryptography. These new techniques that has emerged out of various exploitations in the field of cryptography rises a fair amount of hope that we can over come the problems we are facing in a headhoc way. These proven technologies can meet the needs of the most demanding of environments while their respective focus on manageability has automated many tasks and simplified administrative functions through easy-to-use interfaces developed through years of customer feedback..And at the end of the document we can conclude that soon we can save secrecy involved in message passing from the dangerous clutches of message
Index Terms- *Security Threats, Cryptosystems, Ciph Downline Load Security ertext,Encryption,Decryption, Interception, Interruption, Fabrication, Authentication, Password Hashing.*

I. Introduction

The Individual and Authority (defined as civil government, military, and corporations) have always had a complex relationship with cryptography. Craving digital privacy, individuals highly value the effectiveness and transparency of the algorithms protecting personal and financial secrets. On the other hand, governments want to intercept criminal communication, the military wants to maintain a proven military asset, and corporations, especially those that sell media, want to safeguard their multibillion-dollar markets. These later desires often run counter to the privacy-rights of the individuals.

After establishing basic technical literacy, I will argue that the future advent on quantum cryptology, based on the fantastic yet proven field of quantum mechanics, represents a revolution in our information society. I will show that the past 50 years of digital cryptography has been characterized by a constant “tug-of-war” between the individual and authority. Quantum cryptology will end this decade-long struggle and also define who will finally win what cryptographic rights. However, the result of quantum cryptography is largely dependent on what precedents we establish in this generation. Lastly, I will attempt to make educated predictions on how our individual privacy rights will be affected by this technology.Cyberspace—the Internet and other computer-based networks—is becoming one of the most important infrastructures that characterize modern societies. Among the networks of cyberspace are systems that control and manage other infrastructures such as banking, emergency services, energy delivery, and many transportation and military systems. Thus, many regions’ economic and social stability may depend on these networks. The computer-communications networks of cyberspace are the underlying technological bases that will enable all “visions of the information society.”

Dependencies on networks for communication and business operations continue to grow along with the growth of cyberspace. Today, the Internet in particular, which has grown without any planning or central organization, is a vast network of networks. As of 1989, the Internet interconnected around 20 countries and 100,000 hosts. The majority of those hosts were located in the United States. As of early 2002, there were hundreds of millions of host computers2 and perhaps at least a half billion users worldwide.

More than half of the users are now located outside the U.S. and perhaps a quarter outside of the OECD countries. It is increasingly beyond the scope of single nations to control users who would inflict damage to or via the systems of cyberspace.

Destructive acts using computer networks have cost billions of dollars and increasingly threaten the resources of network-connected critical infrastructures. Threats to network infrastructures are potentially extensive not only as their value increases in terms of the infrastructures themselves, the value of hosted services, and the value of what is located on them, but also because of their widespread and low-cost access.

These infrastructures of cyberspace are vulnerable due to three kinds of failure: complexity, accident, and hostile intent. However, we lack a comprehensive understanding of these vulnerabilities—largely because of the extraordinary complexities of many of the problems, and perhaps from too little effort to acquire this understanding. But there is ample evidence that vulnerabilities are there: examples of all three kinds of failure abound, and vulnerabilities are found almost every time people seriously look for them.

Introduction To Security

We start our description of security in distributed systems by taking a look atsome general security issues. First, it is necessary to define what a secure systemis. We distinguish security policies from security mechanisms, and take a look atthe Globus wide-area system for which a security policy has been explicitly formulated. Our second concern is to consider some general design issues for securesystems. Finally, we briefly discuss some cryptographic algorithms, which play akey role in the design of security protocols.

Security Threats, Policies, and Mechanisms

Security in computer systems is strongly related to the notion of dependability. Informally, a dependable computer system is one that we justifiably trust todeliver its services (Laprie, 1995)., dependabilityincludes availability, reliability, safety, and maintainability. However, if we are toput our trust in a computer system, then confidentiality and integrity should alsobe taken into account. Confidentiality refers to the property of a computer system whereby its information is disclosed only to authorized parties. Integrity isthe characteristic that alterations to a system's assets can be made only in anauthorized way. In other words, improper alterations in a secure computer systemshould be detectable and recoverable. Major assets of any computer system are itshardware, software, and data.

Another way of looking at security in computer systems is that we attempt toprotect the services and data it offers against security threats. There are fourtypes of security threats to consider (Pfleeger, 1997):

1. Interception
2. Interruption
3. Modification
4. Fabrication

Interception refers to the situation that an unauthorized party has gainedaccess to a service or data. A typical example of interception is where communication between two parties has been overheard by someone else. Interception alsohappens when data are illegally copied, for example, after breaking into aperson's private directory in a file system.

Modifications involve unauthorized changing of data or tampering with a serviceso that it no longer adheres to its original specifications. Examples of modificationsinclude intercepting and subsequently changing transmitted data, tamperingwith database entries, and changing a program so that it secretly logs theactivities of its user.

Fabrication refers to the situation in which additional data or activity are generatedthat would normally not exist. For example, an intruder may attempt to addan entry into a password file or database. Likewise, it is sometimes possible tobreak into a system by replaying previously sent messages. We shall come acrosssuch examples later in this chapter.

Note that interruption, modification, and fabrication can each be seen as aform of data falsification.

Encryption is fundamental to computer security. Encryption transforms datainto something an attacker cannot understand. In other words, encryption providesa means to implement confidentiality. In addition, encryption allows us to checkwhether data have been modified. It thus also provides support for integritychecks.

Authentication is used to verify the claimed identity of a user, client, server,and so on. In the case of clients, the basic premise is that before a service will dowork for a client, the service must learn the client's identity. Typically, users areauthenticated by means of passwords, but there are many other ways to authenticateclients.

In this sense, logging accesses makes attacking sometimes a riskier business.Example: The Globus Security ArchitectureThe notion of security policy and the role that security mechanisms play indistributed systems for enforcing such policies is often best explained by taking alook at a concrete example. Consider the security policy defined for the Globuswide-area system (Chervenak et al., 2000). Globus is a system supporting largescaledistributed computations in which many hosts, files, and other resources aresimultaneously used for doing a computation. Such environments are also referredto as computational grids (Foster and Kesselman, 1998). Resources in these gridsare often located in different administrative domains that may be located in differentparts of the world.

Because users and resources are vast in number and widely spread across differentadministrative domains, security is essential. To devise and properly usesecurity mechanisms, it is necessary to understand what exactly needs to be protected,and what the assumptions are with respect to security. Simplifying matterssomewhat, the security policy for Globus entails the following eight statements,which we explain below (Foster et al., 1998):

1. The environment consists of multiple administrative domains.
2. Local operations (i.e., operations that are carried out only within asingle domain) are subject to a local domain security policy only.
3. Global operations (i.e., operations involving several domains) requirethe initiator to be known in each domain where the operation is carriedout.
4. Operations between entities in different domains require mutualauthentication.
5. Global authentication replaces local authentication.

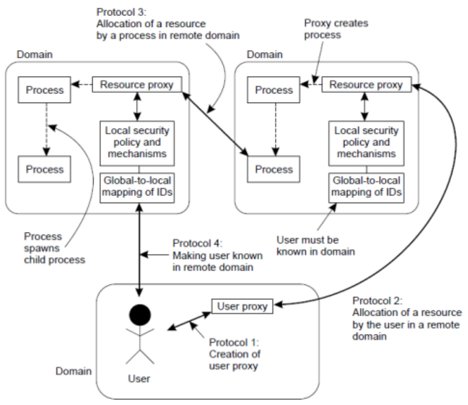

Protocol 3: Allocation of a resource by a process in remote domain
Proxy creates process
Domain
Domain
Process
Resource proxy
Process
Resource proxy
Local security policy and mechanisms
Local security policy and mechanisms
Process
Process
Global-to-local mapping of IDs
Global-to-local mapping of IDs
Process spawns child process
Protocol 4: Making user known in remote domain
User must be known in domain
User proxy
Protocol 2: Allocation of a resource by the user in a remote domain
Protocol 1: Creation of user proxy
Domain
User

Figure 1. The Globus security architecture.

credentials and the local ones can be registered by the user in a mapping tablelocal to that domain. Specific details of each protocol are described in (Foster et al., 1998). Theimportant issue here is that the Globus security architecture reflects its securitypolicy as stated above. The mechanisms used to implement that architecture, inparticular the above mentioned protocols, are common to many distributed systems,and are discussed extensively in this chapter. The difficulty in designingsecure distributed systems is not so much caused by security mechanisms, but bydeciding on how those mechanisms are to be used to enforce a security policy. Inthe next section, we consider some of these design decisions.

Design Issues

A distributed system, or any computer system for that matter, must providesecurity services by which a wide range of security policies can be implemented.There are a number of important design issues that need to be taken into accountwhen implementing general-purpose security services. In the following pages, wediscuss three of these issues: focus of control, layering of security mechanisms,and simplicity (see also (Gollmann, 1999).

Focus of Control

When considering the protection of a (possibly distributed) application, thereare essentially three different approaches that can be followed, as shown inFig. 8-2. The first approach is to concentrate directly on the protection of the datathat is associated with the application. By direct, we mean that irrespective of thevarious operations that can possibly be performed on a data item, the main concernis to ensure data integrity. Typically, this type of protection occurs in databasesystems in which various integrity constraints can be formulated that areautomatically checked each time a data item is modified (see, for example, (Ullman,1988).

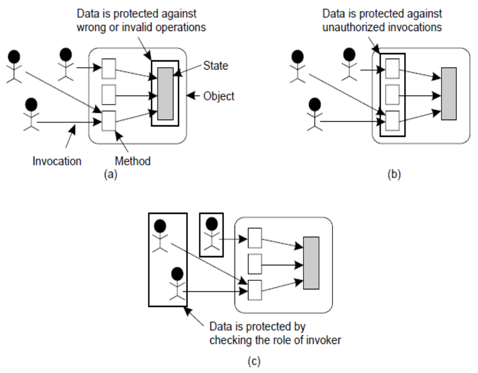

Data is protected against wrong or invalid operations
State
Object
Invocation
Method
(a)
Data is protected against unauthorized invocations
(b)
Data is protected by checking the role of invoker
(c)

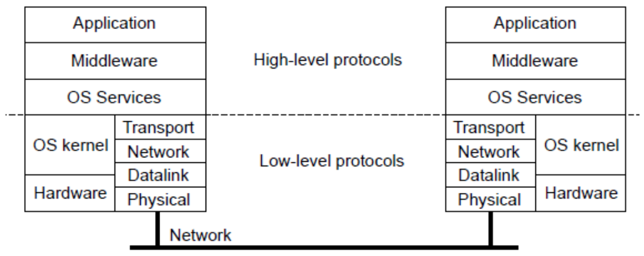
Figure 2. Three approaches for protection against security threats. (a) Protectionagainst invalid operations (b) Protection against unauthorized invocations.(c) Protection against unauthorized users.

The second approach is to concentrate on protection by specifying exactlywhich operations may be invoked, and by whom, when certain data or resourcesare to be accessed. In this case, the focus of control is strongly related to accesscontrol mechanisms, which we discuss extensively later in this chapter. For example,in an object-based system, it may be decided to specify for each method thatis made available to clients which clients are permitted to invoke that method.
Alternatively, access control methods can be applied to an entire interface offeredby an object, or to the entire object itself. This approach thus allows for variousgranularities of access control.

Layering of Security Mechanisms

An important issue in designing secure systems is to decide at which levelsecurity mechanisms should be placed. A level in this context is related to the logicalorganization of a system into a number of layers. For example, computer networks**are often organized into layers following some reference model, as we discussedin Chap. 2. In Chap. 1, we introduced the organization of distributed systems**consisting of separate layers for applications, middleware, operating systemservices, and the operating system kernel. Combining the layered organization ofcomputer networks and distributed systems, leads roughly to what is shown inFig. 8-3.

Figure 3. The logical organization of a distributed system into several layers.



In essence, Fig. 8-3 separates general-purpose services from communicationservices. This separation is important for understanding the layering of security indistributed systems and, in particular, the notion of trust. The difference betweentrust and security is important. A system is either secure or it is not (taking variousprobabilistic measures into account), but whether a client considers a system to besecure, is a matter of trust (Pfleeger, 1997). In which layer security mechanismsare placed depends on the trust a client has in how secure the services are in a particularlayer.

As an example, consider an organization located at different sites that are connectedthrough a communication service such as Switched Multi-megabit DataService (SMDS). An SMDS network can be thought of as a link-level backboneconnecting various local-area networks at possibly geographically dispersed sites,as shown in Fig. 8-4 (for more information on SMDS see (Klessig and Tesink,1995).

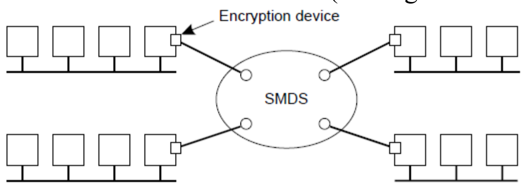


Figure 4. Several sites connected through a wide-area backbone service.

Security can be provided by placing encryption devices at each SMDS router,as also shown in Fig. 8-4. These devices automatically encrypt and decrypt packetsthat are sent between sites, but do not otherwise provide secure communicationbetween hosts at the same site. If Alice at site A sends a message to Bob atsite B, and she is worried about her message being intercepted, she must at leasttrust that the encryption of intersite traffic to work properly. This means, forexample, that she must trust the system administrators at both sites to have takenthe proper measures against tampering with the devices.

Distribution of Security Mechanisms

Dependencies between services regarding trust lead to the notion of aTrusted Computing Base (TCB). A TCB is the set of all security mechanismsin a (distributed) computer system that are needed to enforce a security policy.The smaller the TCB, the better. If a distributed system is built as middleware onan existing network operating system, its security may depend on the security ofthe underlying local operating systems. In other words, the TCB in a distributedsystem may include the local operating systems at various hosts.

Middleware-based distributed systems thus require trust in the existing localoperating systems they depend on. If such trust does not exist, then part of thefunctionality of the local operating systems may need to be incorporated into thedistributed system itself. Consider a microkernel operating system, in which mostoperating-system services run as normal user processes. In this case, the file system,for instance, can be entirely replaced by one tailored to the specific needs ofa distributed system, including its various security measures. Note that thisapproach may gradually replace a middleware-based distributed system by a distributedoperating system.

Consistent with this approach is to separate security services from other typesof services by distributing services across different machines depending on therequired security. For example, for a secure distributed file system, it may be possibleto isolate the file server from clients by placing the server on a machine witha trusted operating system, possibly running a dedicated secure file system.Clients and their applications are placed on untrusted machines.

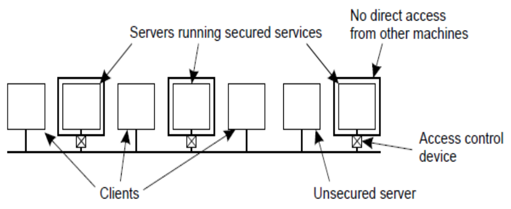

Servers running secured services
No direct access from other machines
Access control device
Clients
Unsecured server

Figure 5. The principle of RISSC as applied to secure distributed systems.

8.1.3 Cryptography

Fundamental to security in distributed systems is the use of cryptographictechniques. The basic idea of applying these techniques is simple. Consider asender S wanting to transmit message m to a receiver *R.* To protect the messageagainst security threats, the sender first encrypts it into an unintelligible message*m',* and subsequently sends m' to *R. R*, in turn, must decrypt the received messageinto its original form m.

Encryption and decryption are accomplished by using cryptographic methodsparameterized by keys, as shown in Fig. 8-6. The original form of the messagethat is sent is called the plaintext, shown as P in Fig. 8-6; the encrypted form isreferred to as the ciphertext, illustrated as C.

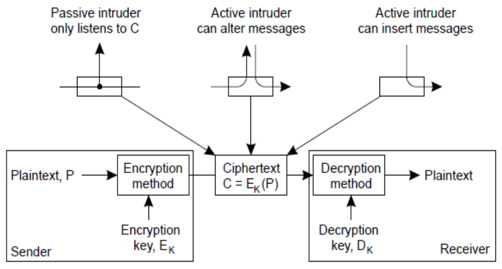


Figure 6. Intruders and eavesdroppers in communication.

To describe the various security protocols that are used in building securityservices for distributed systems, it is useful to have a notation to relate plaintext,ciphertext, and keys. Following the common notational conventions, we will useC = EK(P) to denote that the ciphertext C is obtained by encrypting the plaintextP using key K. Likewise, P = DK(C) is used to express the decryption of theciphertext C using key K, resulting in the plaintext P.

The second type of attack that needs to be dealt with is that of modifying themessage. Modifying plaintext is easy; modifying ciphertext that has been properlyencrypted is much more difficult because the intruder will first have to decrypt themessage before it can meaningfully modify it. In addition, he will also have toproperly encrypt it again or otherwise the receiver may notice that the messagehas been tampered with.

The third type of attack is when an intruder inserts encrypted messages intothe communication system, attempting to make R believe these messages camefrom S. Again, as we shall see later in this chapter, encryption can help protectagainst such attacks. Note that if an intruder can modify messages, he can alsoinsert messages.

There is a fundamental distinction between different cryptographic systems,based on whether or not the encryption and decryption key are the same. In asymmetric cryptosystem, the same key is used to encrypt and decrypt a message.
In other words,

$$P = D_K(E_K(P))$$

Symmetric cryptosystems are also referred to as secret-key or shared-key systems,because the sender and receiver are required to share the same key, and to ensurethat protection works, this shared key must be kept secret; no one else is allowedto see the key. We will use the notation KA,B to denote a key shared by A and B.

In an asymmetric cryptosystem, the keys for encryption and decryption aredifferent, but together form a unique pair. In other words, there is a separate keyKE for encryption and one for decryption, K_D, such that

$$P = D_{K_D}(E_{K_E}(P))$$

One of the keys in an asymmetric cryptosystem is kept private, the other is madepublic. For this reason, asymmetric cryptosystems are also referred to as publickeysystems. In what follows, we use the notation $K^+_A$to denote a public keybelonging to A, and K^-_A as its corresponding private key.

Anticipating the detailed discussions on security protocols later in thischapter, which one of the encryption or decryption keys that is actually made publicdepends on how the keys are used. For example, if Alice wants to send a confidentialmessage to Bob, she should use Bob's public key to encrypt the message.Because Bob is the only one holding the private decryption key, he is also theonly person that can decrypt the message.

On the other hand, suppose that Bob wants to know for sure that the messagehe just received actually came from Alice. In that case, Alice can keep herencryption key private to encrypt the messages she sends. If Bob can successfullydecrypt a message using Alice's public key (and the plaintext in the message hasenough information to make it meaningful to Bob), he knows that message musthave come from Alice, because the decryption key is uniquely tied to the encryptionkey. We return to such algorithms in detail below.

One final application of cryptography in distributed systems is the use of hashfunctions. A hash function H takes a message m of arbitrary length as input andproduces a bit string h having a fixed length as output:

$$h = H(m)$$

A hash h is comparable to the extra bits that are added to a message in communicationsystems to allow for error detection, such a cyclic-redundancy check(CRC).

Hash functions that are used in cryptographic systems have a number of properties.First, they are one-way functions, meaning that it is computationallyinfeasible to find the input m that corresponds to a known output h. On the otherhand, computing h from m is easy. Second, they have the weak collision resistanceproperty, meaning that given an input m and its associated outputh = H(m), it is computationally infeasible to find another, different input $m' \neq m$,such that $H(m) = H(m')$. Finally, cryptographic hash functions also have thestrong collision resistance property, which means that, when given only H, it iscomputationally infeasible to find any two different input values m and m', suchthat $H(m) = H(m')$.

Similar properties apply to any encryption function E and the keys that areused. Furthermore, for any encryption function E, it should be computationallyinfeasible to find the key K when given the plaintext P and associated ciphertext$C = E_K(P)$. Likewise, analogous to collision resistance, when given a plaintext Pand a key K, it should be impossible to find another key K ′ such that$E_K(P) = E_{K'}(P)$.

The art and science of devising algorithms for cryptographic systems has along history (Kahn, 1967), and building secure systems is often surprisingly difficult,or even impossible (Schneier, 2000). It is beyond the scope of this book todiscuss any of these algorithms in detail. However, to give some impression ofcryptography in computer systems, we will now briefly present three representativealgorithms. Detailed information on these and other cryptographic algorithmscan be found in (Kaufman et al., 1995; Menezes et al., 1996; and Schneier, 1996).

Before we go into the details of the various protocols, Fig. 7 summarizes thenotation and abbreviations we use.

Notation	Description
$K_{A,B}$	Secret key shared by A and B
K_A^+	Public key of A
K_A^-	Private key of A

Figure 7. Notation used in this chapter.

Symmetric Cryptosystems: DES

Our first example of a cryptographic algorithm is the Data Encryption Standard(DES), which is used for symmetric cryptosystems. DES is designed tooperate on 64-bit blocks of data. A block is transformed into an encrypted (64 bit)block of output in 16 rounds, where each round uses a different 48-bit key forencryption. Each of these 16 keys is derived from a 56-bit master key, as shown inFig. 8-8(a). Before an input block starts its 16 rounds of encryption, it is first subjectto an initial permutation, of which the inverse is later applied to the encryptedoutput leading to the final output block.

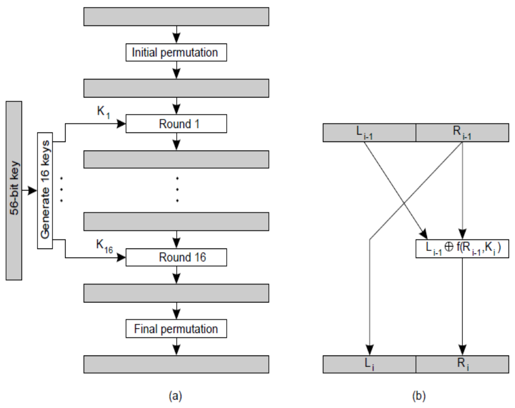


Figure 8. (a) The principle of DES. (b) Outline of one encryption round.

Each encryption round i takes the 64-bit block produced by the previous roundi − 1 as its input, as shown in Fig. 8-8(b). The 64 bits are split into a left part L_i −1and a right part R_i *−1,* each containing 32 bits. The right part is used for the leftpart in the next round, that is, $L_i = R_{i-1}$.

The hard work is done in the mangler function f. This function takes a 32-bitblock Ri −1 as input, together with a 48-bit key Ki , and produces a 32-bit block thatis XORed with Li −1 to produce Ri . (XOR is an abbreviation for the exclusive oroperation.) The mangler function first expands Ri −1 to a 48-bit block and XORs itwith Ki . The result is partitioned into eight chunks of six bits each. Each chunk isthen fed into a different S-box, which is an operation that substitutes each of the64 possible 6-bit inputs into one of 16 possible 4-bit outputs. The eight outputchunks of four bits each are then combined into a 32-bit value and permutedagain.

The 48-bit key Ki for round i is derived from the 56-bit master key as follows.First, the master key is permuted and divided into two 28-bit halves. For eachround, each half is first rotated one or two bits to the left, after which 24 bits areextracted. Together with 24 bits from the other rotated half, a 48-bit key is constructed.The details of one encryption round are shown in Fig. 9.

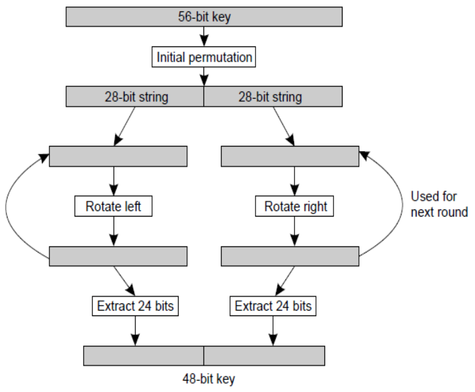


Figure 9. Details of per-round key generation in DES.

The principle of DES is quite simple, but the algorithm is difficult to breakusing analytical methods. Using a brute-force attack by simply searching for a keythat will do the job has become easy as has been demonstrated in recent times.Using DES three times in a special encrypt-decrypt-encrypt mode with differentkeys appears to be safe for the moment.

Public-Key Cryptosystems: RSA

Our second example of a cryptographic algorithm is widely used for publickeysystems: RSA, named after its inventors Rivest, Shamir, and Adleman (1978).The security of RSA comes from the fact that no methods are known to efficientlyfind the prime factors of large numbers. It can be shown that each integer can bewritten as the product of prime numbers. For example, 2100 can be written as

$$2100 = 2 \times 2 \times 3 \times 5 \times 5 \times 7$$

making 2, 3, 5, and 7 the prime factors in 2100. In RSA, the private and publickeys are constructed from very large prime numbers (consisting of hundreds ofdecimal digits). As it turns out, breaking RSA is equivalent to finding those twoprime numbers. So far, this has shown to be computationally infeasible despitemathematicians working on the problem for centuries.Generating the private and public keys requires four steps:

1. Choose two very large prime numbers, p and q.
2. Compute $n = p \times q$ and $z = (p - 1) \times (q - 1)$.
3. Choose a number d that is relatively prime to z.
4. Compute the number e such that $e \times d = 1 \bmod z$.

One of the numbers, say d, can subsequently be used for decryption, whereas e isused for encryption. Only one of these two is made public, depending on what thealgorithm is being used for.

Let us consider the case that Alice wants to keep the messages she sends toBob confidential. In other words, she wants to ensure that no one but Bob canintercept and read her messages to him. RSA considers each message m to be justa string of bits. Each message is first divided into fixed-length blocks, where eachblock mi , interpreted as a binary number, should lie in the interval $0 \leq m_i < n$.To encrypt message m, the sender calculates for each block mi the value$c_i = m_i^e$ (mod n), which is then sent to the receiver. Decryption at the receiver'sside takes place by computing $m_i = c_i^d$ (mod n). Note that for the encryption, bothe and n are needed, whereas decryption requires knowing the values d and n.

When comparing RSA to symmetric cryptosystems such as DES, RSA has thedrawback of being computationally more complex. As it turns out, encryptingmessages using RSA is approximately 100–1000 times slower than DES, dependingon the implementation technique used. As a consequence, many cryptographicsystems use RSA to exchange only shared keys in a secure way, but much less foractually encrypting ''normal'' data. We will see examples of the combination ofthese two techniques later in succeeding sections.

Hash Functions: MD5

As a last example of a widely used cryptographic algorithm, we take a look atMD5 (Rivest, 1992). MD5 is a hash function for computing a 128-bit, fixedlength message digest from an arbitrary length binary input string. The inputstring is first padded to a total length of 448 bits (modulo 512), after which thelength of the original bit string is added as a 64-bit integer. In effect, the input isconverted to a series of 512-bit blocks.

The structure of the algorithm is shown in Fig. 8-10. Starting with some constant128-bit value, the algorithm proceeds in k phases, where k is the number of512-bit blocks comprising the padded message. During each phase, a 128-bit digestis computed out of a 512-bit block of data coming from the padded message,and the 128-bit digest computed in the preceding phase.

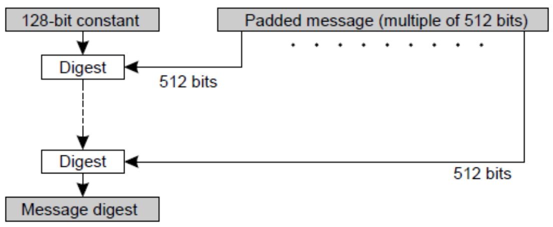


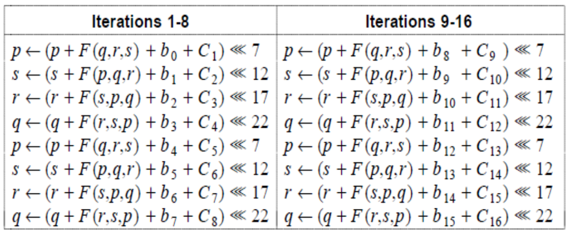
Figure 10. The structure of MD5.

A phase in MD5 consists of four rounds of computations, where each round usesone of the following four functions:

F (x,y,z) = (x AND y) OR ((NOT x) AND z)
G(x,y,z) = (x AND z) OR (y AND (NOT z))
H(x,y,z) = x XOR y XOR z
I (x,y,z) =y XOR (x OR (NOT z))

Each of these functions operates on 32-bit variables x, y, and z. To illustrate howthese functions are used, consider a 512-bit block b from the padded message thatis being processed during phase k. Block b is divided into 16 32-bit subblocksb0,b1,...,b15. During the first round, function F is used to change four variables(denoted as p, q, r, and s, respectively) in 16 iterations as shown in Fig. 8-11

These variables are carried to each next round, and after a phase has finished,passed on to the next phase. There are a total of 64 predefined constants C_i . Thenotation x <<< n is used to denote a left rotate: the bits in x are shifted n positionsto the left, where the bit shifted off the left is placed in the rightmost position.



Iterations 1-8	Iterations 9-16
$p \leftarrow (p + F(q,r,s) + b_0 + C_1) \lll 7$	$p \leftarrow (p + F(q,r,s) + b_8 + C_9) \lll 7$
$s \leftarrow (s + F(p,q,r) + b_1 + C_2) \lll 12$	$s \leftarrow (s + F(p,q,r) + b_9 + C_{10}) \lll 12$
$r \leftarrow (r + F(s,p,q) + b_2 + C_3) \lll 17$	$r \leftarrow (r + F(s,p,q) + b_{10} + C_{11}) \lll 17$
$q \leftarrow (q + F(r,s,p) + b_3 + C_4) \lll 22$	$q \leftarrow (q + F(r,s,p) + b_{11} + C_{12}) \lll 22$
$p \leftarrow (p + F(q,r,s) + b_4 + C_5) \lll 7$	$p \leftarrow (p + F(q,r,s) + b_{12} + C_{13}) \lll 7$
$s \leftarrow (s + F(p,q,r) + b_5 + C_6) \lll 12$	$s \leftarrow (s + F(p,q,r) + b_{13} + C_{14}) \lll 12$
$r \leftarrow (r + F(s,p,q) + b_6 + C_7) \lll 17$	$r \leftarrow (r + F(s,p,q) + b_{14} + C_{15}) \lll 17$
$q \leftarrow (q + F(r,s,p) + b_7 + C_8) \lll 22$	$q \leftarrow (q + F(r,s,p) + b_{15} + C_{16}) \lll 22$

Figure 11. The 16 iterations during the first round in a phase in MD5.

The second round uses the function G in a similar fashion, whereas H and Iare used in the third and fourth round, respectively. Each step thus consists of 64iterations, after which the next phase is started, but now with the values that p, q,r, and s have at that point.

II. Cryptography

Cryptography (or cryptology; from Greek κρυπτός, "hidden, secret"; and γράφειν, graphein, "writing", or -λογία, -logia, "study", respectively)[1] is the practice and study of techniques for secure communication in the presence of third parties (called adversaries).[2] More generally, it is about constructing and analyzing protocols that overcome the influence of adversaries[3] and which are related to various aspects in information security such as data confidentiality, data integrity, authentication, and non-repudiation.[4] Modern cryptography intersects the disciplines of mathematics, computer science, and electrical engineering. Applications of cryptography include ATM cards, computer passwords, and electronic commerce.

Fundamental Tenet of Cryptography
Computational Difficulty

It is important for crypto graphic algorithms to be reasonably efficient for the good guys to compute. The good guys are the ones with knowledge of the keys.1 Cryptographic algorithms are not impossible to break without the key. A bad guy can simply try all possible keys until one works. The security of a cryptographic scheme depends on how much work it is for the bad guy to break it. If the best possible scheme will take 10 million years to break using all of the computers in the world, then it can be considered reasonably secure.

Going back to the combination lock example, a typical combination might consist of three numbers, each a number between 1 and 40. Let's say it takes 10 seconds to dial in a combination. That's reasonably convenient for the good guy. How much work is it for the bad guy? There are 403 possible combinations, which is 6 4000. At 10 seconds per try, it would take a week to try all combinations, though on average it would only take half that long (even though the right number is always the last one you try!).

Often a scheme can be made more secure by making the key longer. In the combination lock analogy, making the key longer would consist of requiring four numbers to be dialed in. This would make a little more work for the good guy. It might now take 13 s econds to dial in the combination. But the bad gu y has 40 times as many combinations to try, at 13 seconds each, so it would take a year to try all combinatio ns. (And if it took that lo ng, he might want to stop to eat or sleep).

With cryptography, computers can be used to exhaustively try keys. Computers are a lot faster than people, and they don't get tired, so thousands or millions of keys can be tried per second. Also, lots of keys can be tried in parallel if you have multiple computers, so time can be saved b y spending money on more comp uters.

Sometimes a cryptographic algorithm has a variable-length key. It can be made more secure by increasing the length of the key. Increasing the length of the key by one bit makes the good guy's job just a little bit harder, but makes the bad guy's job up to twice as hard (because the number of possible keys doubles). Some cryptographic algorithms have a fixed-length key, but a similar algorithm with a lon ger key can be devised if necessary. If computers get 1000 times fas ter, so that the bad guy's job becomes reasonably practical, making the key 10 bits longer will make the bad guy's job as hard as it was before the advance in computer speed. However, it will be much easier for the good guys (because their computer speed increase far o utweighs the increment in key length). So the faster computers get, the better life gets for the good guys.

Keep in mind that breaking the cryptographic scheme is often only one way of getting what you want. For instance, a bolt cutter works no matter how many digits are in the combination.

BREA KING AN ENCRYPTION SCHEME

What do we mean when we speak of a bad guy Fred breaking an encryption scheme? The three basic attacks are known as ciphertext only, known plaintext, and chosen plaintext.

Ciphertext Only

In a ciphertext only attack, Fred has seen (and presumably stored) s ome ciphertext that he can analyze at leisure. Typically it is not difficult for a bad guy to obtain ciphertext. (If a bad guy can't access the encrypted data, then there would have been no need to encrypt the data in the first place!)

How can Fred figure out the plaintext if all he can see is the ciphertext? One possible strategy is to search through all the keys. Fred tries the decrypt operation with each key in turn. It is essential for this attack that Fred be able to recognize when he has succeeded. For instance, if the message was English text, then it is highly unlikely that a decryption operation with an incorrect key could produce something that looked like intelligible text. Because it is important for Fred to be able to differentiate plaintext from gibberish, this attack is s ometimes known as a recognizable plaintext attack.

Known Plaintext

Sometimes life is easier for the attacker. Suppose Fred has somehow obtained some <plaintext, ciphertext> pairs. How might he have o btained these? One possibility is that secret data might not remain secret forever. For instance, the data might consist of specifying the next city to be attacked. Once the attack occurs, the plaintext to the previous day's ciphertext is now known. With a monoalp habetic cipher, a small amount of known plaintext would be a bonanza. From it, the attacker would learn the mappings of a substantial fraction of the most common letters (every letter that was used in the plaintext Fred obtained).

Chosen Plaintext

On rare occasions, life may be easier still for the attacker. In a "chosen plaintext" attack, Fred can choose any plaintext he wants, and get the system to tell him what the corresponding ciphertext is. How could such a thing be p ossible? Suppose the telegraph company offered a service in which they encrypt and transmit messages for you. Suppose Fred had eavesdropped on Alice's encrypted message. Now he'd like to break the telegraph company's encryption scheme so that he can decrypt Alice's message.
He can obtain the corresponding ciphertext to any message he chooses by paying the telegraph company to send the message for him, encrypted. For instance, if Fred knew they were usin g a monoalphabetic cipher, he might send the message

The quick brown fox jumps over the lazy dog.

knowing that he would thereby get all the letters of the alphabet encrypted and then be able to decrypt with certainty any encrypted message.

It is possible that a cryptosystem secure against ciphertext only and known plaintext attacks might still be susceptible to chosen plaintext attacks. For instance, if Fred knows that Alice's message is either Surrender or Fight on, then no matter how wonderful an encryption scheme the telegraph company is using, all he has to do is send the two messages and see which one looks like the encrypted data he saw when Alice's mes sage was transmitted.

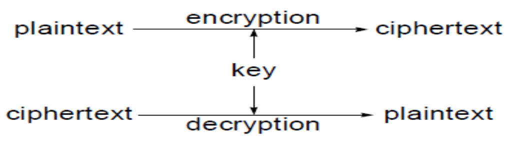
A cryptosystem should resist all three sorts of attacks. That way its users don 't need to worry about whether there are any opportunities for attackers to k now or choose plaintext. Like wearin g both a belt and suspenders, many systems that use cryptographic algorithms will also go out of their way to prevent any chance of chosen plaintext attacks.

Types Of Cryptographic Functions

There are three kinds of cryptographic functions: hash functions, secret key functions, an d public key functions. We will describe what each kind is, and what it is useful for. Public key cryptography involves the use of two keys. Secret key cryptography involves the use of one key. Hash functions involve the use of zero keys! Try to imagine what that could possibly mean, and what use it could possibly have—an algorithm everyone knows with no secret key, and yet it has uses in security.
Since secret key cryptography is probably the most intuitive, we'll describe that first.

Secret Key Cryptography

Secret key cryptography involves the us e of a single key. Given a message (called plaintext) and the key, encryption produces unintelligible data (called an IRS Publication—no! no! that was just a finger slip, we meant to say "ciphertext"), which is about the same length as the plaintext was. Decryption is the reverse of encryption, and uses the same key as encryption.



Secret key cryptography is sometimes referred to as conventional cryptography or symmetric cryptography. The Captain Midnight code and the monoalphabetic cipher are both examples of secret key algorithms, though both are easy to break. In this chapter we describe the functionality of cryptographic algorithms, but not the details of particular algorithms. **In Chapter 3** Secret Key Cryptography we describe the details of two secret key cryptographic algorithms (DES and IDEA) in current use.

2.4.1 Security Uses of Secret Key Cryptography

The next few sections describe the types of things one might do with secret key cryptography.

2.4.2 Transmitting Over an Insecure Channel

It is often impossible to prevent eavesdropping when transmitting information. For instance, a telephone conversation can be tapped, a letter can be intercepted, and a message transmitted on a LAN can be received by unauthorized stations.

If you and I agree on a shared secret (a key), then by using secret key cryptography we can send messages to one another on a medium that can be tapped, without worrying about eavesdroppers. All we need to do is for the sender to encrypt the messages and the receiver to decrypt them using the shared secret. An eavesdropper will only see unintelligible data. This is the classic use of cryptography.

2.4.3 Secure Storage on Insecure Media

If I have information I want to preserve but which I want to assure no one else can look at, I have to be able to store the media where I am sure no one can get it. Between clever thieves an d court orders, there are very few places that are truly secure, and none of these is convenient. If I invent a key and encrypt the information using the key, I can store it anywhere and it is safe so lon g as I can remember the key. Of course, forgetting the key makes the data irrevocably lost, so this must be used with great care.

2.4.4 Authentication

In spy movies, when two agents who don't know each other must rendezvous, they are each given a password or pass phrase that they can use to recognize o ne another. This has the problem that anyone overhearing their conversation or initiating one falsely can gain information useful for replaying later and impersonating the person to whom they are talking.

The term strong authentication means that someone can prove knowledge of a secret without revealing it. Strong authentication is possible with cryptography. Strong authentication is particularly useful when two computers are trying to communicate over an insecure net work (since few people can execute cryptographic algorithms in their heads). Suppose Alice and Bob share a key KAB and they want to verify they are speaking to each other. They each pick a random number, which is known as a challenge. Alice picks rA. Bob picks rB. The value x encrypted with the key KAB is known as the response to the challenge x.

2.4.5 Integrity Check

A secret key scheme can be used to generate a fixed-length cryptographic checksum associated with a message. This is a rather nonintuitive use of secret key technolo gy. What is a checksum? An o rdinary (noncryptographic) checksum protects against accidental corruption of a message. The original derivation of the term checksum comes from the operation of breaking a message into fixed-length blocks (for instance, 32-bit words) and adding them up. The sum is sent along with the message. The receiver similarly breaks up the message, repeats the addition, and checks the sum. If the message had been garbled en route, the sum will not match the sum sent and the message is rejected, unless, o f course, there were two or more errors in the transmission that canceled one another. It turns out this is not terribly unlikely, given that if flaky hardware turns a bit off somewhere, it is likely to turn a corresponding bit on somewhere else. To protect against such "regular" flaws in hardware, more complex checksums called CRCs were devised. But these still only protect against faulty hardware and not an intelligent attacker. Since CRC algorithms are published, an attacker who wanted to change a message could do so, compute the CRC on the new message, and send that along.

To provide protection against malicious changes to a message, a secret checksum algorithm is required, such that an attacker not knowing the algorithm can't compute the right checksum for the message to be accepted as authentic. As with encryptio n algorithms, it's better to have a commo n (known) algorithm and a secret key. This is what a cryptographic checksum does. Given a key an d a message, the algorithm produces a fixed-length message integrity code (MIC) that can be sent with the message.

If anyone were to modify the message, and they didn't know the key, they would have to guess a MIC and the chance of getting it right depends on the length. A typical MIC is at least 48 bits long, so the chance of getting away with a forged message is only one in 280 trillion (or about the chance of going to Las Vegas with a dime and letting it ride on red at the roulette table until you have enough to pay off the U.S. national debt).

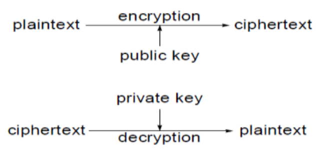
Such message integrity codes have been in use to protect the integrity of large interbank electronic funds transfers for quite some time. The mes sages are not kept secret from an eavesdropper, but their integrity is ensured.

2.5 Public Key Cryptography

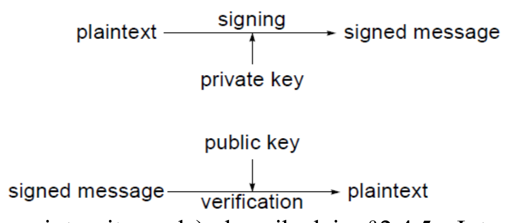
Public key cryptography is sometimes also referred to as asymmetric cryptography. Public key cryptography is a relatively new field, invented in 1975 [DIFF76b] (at least that's the first published record—it is rumored that NSA or similar o rganizations may have discovered this technology earlier). Unlike secret key cryptography, keys are not shared. Instead, each individual has two keys: a private key that need not be revealed to an yone, and a public key that is preferably known to the entire world.
Note that we call the private key a pr ivate key and not a secret key. This convention is an attempt to make it clear in any context whether public key cryptography or secret key cryptography is being used. There are people in this world whose sole purpose in life is to try to confuse people.

They will use the term secret key for the private key in public key cryptography, or use the term private key for the secret key in secret key technology. One of the most important contributions we can make to the field is to convince people to feel strongly about using the terminology correctly—the term secret key refers only to the single secret n umber used in secret key cryptograph y. The term private key MUST be used when referring to the key in public key cryptography that must not be made public. (Yes, when we speak we sometimes accidentally say the wrong thing, but at least we feel guilty about it.)

There is something unfortunate about the terminology public and private. It is that both words begin with p. We will sometimes want a single letter to refer to one of the keys. The letter p won't do. We will use the letter e to refer to the public key, since the public key is used when encrypting a message. We'll use the letter d to refer to the private key, because the private key is used to decrypt a message. Encryption and decryption are two mathematical functions that are inverses of each other.



There is an additional thing one can do with public key technology, which is to generate a digital signature on a message. A digital signature is a number associated with a message, like a



Check sum or the MIC (message integrity code) described in §2.4.5 Integrity Check. However, unlike a checksum, which can be generated by anyone, a digital signature can only be generated by someone knowing the private key. A public key signature differs from a secret key MIC because verification of a MIC requires knowledge of the same secret as was used to create it. Therefore anyone who can verify a MIC can also generate one, and so be able to substitute a different message and corresponding MIC. In contrast, verification of the signature only requires knowledge of the public key. So Alice can sign a message by generating a signature only she can generate, and other people can verify that it is Alice's signature, but cannot forge her signature. This is called a signature because it shares with handwritten signatures the property that it is possible to be able to recognize a signature as authentic without being able to forge it.

2.5.1 Security Uses of Public Key Cryptography

Public key cryptography can do anything secret key cryptography can do, but the known public key cryptographic algorithms are orders of magnitude slower than the best known secret key cryptographic algorithms and so are usually o nly used for things secret key cryptography can't do. Public key cryptography is very useful because network security based on public key technology tends to be more secure and more easily configurable. Often it is mixed with secret key technology. For example, public key cryptography might be used in the beginning of communication for authentication and to establish a temporary shared secret key, then the secret key is used to encrypt the remainder of the conversation using secret key technology.

For instance, suppose Alice wants to talk to Bob. She uses his public key to encrypt a secret key, then uses that secret key to encrypt whatever else she wants to send him. Only Bob can decrypt the secret key. He can then communicate using that secret key with whoever sent that message. Notice that given this protocol, Bob does not know that it was Alice who sent the message. This could be fixed by having Alice digitally sign the encrypted secret key using her private key. Now we'll describe the types of things one might do with public key cryptography.

2.5.2 Transmitting Over an Insecure Channel

Suppose Alice's <public key, private key> pair is <e_A, d_A>. Suppose Bob's key pair is <e_B, d_B>. Assume Alice knows Bob's p ublic key, and Bob knows Alice's public key. Actually, accurately learning other people's public keys is one of the big gest challenges in using public key cryptography and will be discussed in detail in §7.7.2 Certification Authorities (CAs). But for now, do n't worry about it.

$$\begin{tikzcd}
\text{Alice} & \text{Bob} \\
\text{encrypt } m_A \text{ using } e_B \arrow[r] & \text{decrypt to } m_A \text{ using } d_B \\
\text{decrypt to } m_B \text{ using } d_A & \text{encrypt } m_B \text{ using } e_A \arrow[l]
\end{tikzcd}$$

2.5.3 Secure Storage on Insecure Media

This is really the same as what one would do with secret key cryptography. You'd encrypt the data with your public key. Then nobody can decrypt it except you, since decryption will require the use of the private key. It has the advantage over encryption with s ecret key technology that y ou don't have to risk giving your private key to the machine that is going to encrypt the data for you. As with secret key technology, if you lose your private key, the data is irretrievably lost. If you are worried ab out that, you can encrypt an additional copy of the data under the public key of someone you trust, like your lawyer.

2.5.4 Authentication

Authentication is an area in which public key technology potentially gives a real benefit. With secret key cryptography, if Alice and Bob want to communicate, they have to share a secret. If Bob wants to be able to prove his identity to lots of entities, then with secret key technology he will need to remember lots of s ecret keys, one for each entity to which he would like to prove his identity. Possibly he could us e the same shared secret with Alice as with Carol, but that has the disadvantage that then Carol an d Alice could impersonate Bob to each other.

Public key technology is much more convenient. Bob only needs to remember a single secret, his own private key. It is true that if Bob wants to be able to verify the identity of thousands of entities, then he will need to know thousands of public keys, but in general the entities verifying identities are computers which don 't mind remembering thousands of things, whereas the entities proving their identities are often humans, which do mind remembering thing s.

Here's an example of how Alice can use public key cryptography for verifying Bob's identity assuming Alice knows Bob's public key. Alice chooses a random number r, encrypts it using Bob's public key eB, and sends the result to Bob. Bob proves he knows dB by decrypting the message an d sending r back to Alice.

$$\begin{tikzcd}
\text{Alice} & \text{Bob} \\
\text{encrypt } r \text{ using } e_B \arrow[r] & \text{decrypt to } r \text{ using } d_B \\
{} & r \arrow[l]
\end{tikzcd}$$

Another advantage of public key authentication is that Alice does not need to keep any secret information. For instance, Alice might be a computer system in which backup tapes are unencrypted and easily stolen. With secret key based authentication, if Carol stole a backup tape an d read the key that Alice shares with Bob, she could then trick Bob into thinking she was Alice. In contrast, with public key based authentication, the only information on Alice's backup tapes is public key information, and that cannot be used to impersonate Bob.

In large-scale systems, like computer networks with thousands of users and services, authentication is usually done with trusted intermediaries. As we'll see in §7.7 Trusted Intermediaries, public key based authentication using intermediaries has several important advantages over secret key based authentication.

2.5.5 **Digital Signatures**

It is often useful to prove that a message was generated by a particular individual, especially if the individual is not necessarily around to be asked about authorship of the mess age. This is easy with public key technology. Bob's signature for a message m can only be generated by someone with knowledge of Bob's private key. And the signature depends on the contents of m. If m is modified in any way, the signature no longer matches. So digital signatures provide two important functions. They prove who generated the information, and they prove that the information has not been modified in any way by anyone since the message and matching signature were generated.

An important example of a use of a signature is in electronic mail to verify that a mail message really did come from the claimed source.

Digital signatures offer an important advantage over secret key based cryptographic checksums—non-repudiation. Suppose Bob sells widgets and Alice routinely buys them. Alice an d Bob might agree that rather than placing orders through the mail with signed purchase orders, Alice will send electronic mail messages to order widgets. To protect against someone forging orders and causing Bob to manufacture more widgets than Alice actually needs, Alice will include a message integrity code on her messages. This could be either a secret key based MIC or a public key based signature. But suppose sometime after Alice places a big order, she changes her mind (the bottom fell out of the wid get market). Since there's a big penalty for canceling an order, she doesn't fess up that she's canceling, but instead denies that she ever placed the order. Bob sues. Bob knows Alice really placed the order because it was cryptographically signed. But if it was signed with a secret key algorithm, he can't prove it to anyone! Since he knows the same secret key that Alice used to sign the order, he could have forged the signature on the message himself and he can't prove to the judge that he didn't! If it was a public key signature on the other hand, he can show the signed message to the judge and the judge can verify that it was signed with Alice's key. A lice can still claim of course that someone must have stolen and misused her key (it might even be true!), but the contract between Alice and Bob could reasonably hold her responsible for damages caused by her inadequately protecting her key. Unlike secret key cryptography, where the keys are shared, you can always tell who's responsible for a sig nature generated with a private key.

Public key algorithms are discussed further in Chapter 5 Public Key Algorithms

2.6 **Hash Algorithms**

Hash algorithms are also known as message digests or one-way transformations.

A cryptographic hash function is a mathematical transformation that takes a message of arbitrary length (transformed into a string of bits) and computes from it a fixed-length (short) number.

We'll call the hash of a message m, h(m). It has the following properties:

- For any message m, it is relatively easy to compute h(m). This just means that in order to be practical it can't take a lot of processing time to compute the hash.
- Given h(m), there is no way to find an m that hashes to h(m) in a way that is substantially easier than going through all possible values of m and computing h(m) for each one.
- Even though it's obvious that many different values of m will be transformed to the same value h(m) (because there are many more possible values of m), it is computationally infeasible to find two values that hash to the same thing.

.

2.6.1 Password Hashing

When a user types a password, the system has to be able to determine whether the user got it right. If the system stores the passwords unencrypted, then anyone with access to the system storage or backup tapes can steal the passwords. Luckily, it is not necessary for the system to know a password in order to verify its correctness. (A proper password is like pornograph y. You can't tell what it is, but you know it when you see it.)

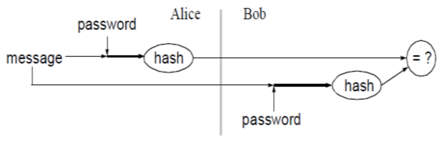
Instead of storing the password, the system can store a hash of the password. When a password is supplied, it computes the password's hash and compares it with the stored value. If they match, the password is deemed correct. If the hashed password file is obtained by an attacker, it is not immediately useful because the passwords can't be derived from the hashes. Historically, some systems made the password file publicly readable, an express ion of co nfidence in the security of the hash. Even if there are no cryptographic flaws in the hash, it is possible to guess passwords an d hash them to see if they match. If a user is careless and chooses a password that is guessable (say, a word that would appear in a 50 000-word dictionary or book of common names), an exhaustive search would "crack" the password even if the encryption were sound. For this reason, many systems hide the hashed password list (and those that don't should).

Message Integrity

Cryptographic hash functions can be used to generate a MIC to protect the integrity of messages transmitted over insecure media in much the same way as secret key cryptography. If we merely sent the message and used the hash of the message as a MIC, this would not be secure, since the hash function is well-known. The bad guy can modify the message and compute a new hash for the new message, and transmit that.

However, if Alice and Bob have agreed on a password, Alice can use a hash to generate a MIC for a message to Bob by taking the message, concatenating the password, and computing the hash of message| password. Alice then sends the hash and the message (without the password) to Bob. Bob concatenates the password to the received message and computes the hash of the result. If that matches the received hash, Bob can have confidence the message was sent by someone know- ing the password. [Note: there are some cryptographic subtleties to making this actually secure;

See fig 22Computing a MIC with a Hash



2.6.3 **Message Fingerprint**

If you want to know whether some large data structure (e.g. a program) has been modified from one day to the next, you could keep a copy of the data on some tamper-proof backing store and periodically compare it to the active version. With a hash function, you can save storage: you simply save the message digest of the data on the tamper-proof backing store (which because the hash is small could be a piece of paper in a filing cabinet). If the message digest hasn't changed, you can be confident none of the data has.

A note to would-be users—if it hasn't already occurred to you, it has occurred to the bad guys—the program that computes the hash must also be in dependently protected for this to be secure. Otherwise the bad guys can change the file but also change the hashing program to report the checksum as though the file were unchanged!

Downline Load Security

It is common practice to have special-purpose devices connected to a network, like routers or printers, that do not have the nonvolatile memory to store the programs they normally run. Instead, they keep a bootstrap program smart enough to get a program from the network and run it. This scheme is called downline load.

Suppose you want to downline load a program and make sure it hasn't been corrupted (whether intentionally or not). If you know the proper hash of the program, you can compute the hash of the loaded program and make sure it has the proper value before running the program.

Digital Signature Efficiency

The best-known public key algorithms are sufficiently processor-intensive that it is desirable to compute a message digest of the message and sign that, rather than to sign the message directly. The message digest algorithms are much less processor-intensive, and the message digest is much shorter than the message.

Algorithms and Keys

A cryptographic algorithm, also called a cipher, is the mathematical function used for encryptionand decryption. (Generally, there are two related functions: one for encryption and the other for decryption.)

If the security of an algorithm is based on keeping the way that algorithm works a secret, it is a restricted algorithm. Restricted algorithms have historical interest, but are woefully inadequate by today's standards. A large or changing group of users cannot use them, because every time a user leaves the group everyone else must switch to a different algorithm. If someone accidentally reveals the secret, everyone must change their algorithm.

Even more damning, restricted algorithms allow no quality control or standardization. Every groupof users must have their own unique algorithm. Such a group can't use off-the-shelf hardware or software products; an eavesdropper can buy the same product and learn the algorithm. They have towrite their own algorithms and implementations. If no one in the group is a good cryptographer, then they won't know if they have a secure algorithm.

Despite these major drawbacks, restricted algorithms are enormously popular for low-security applications. Users either don't realize or don't care about the security problems inherent in their system.

Modern cryptography solves this problem with a key, denoted by K. This key might be any one of alarge number of values. The range of possible values of the key is called the keyspace. Both the encryption and decryption operations use this key (i.e., they are dependent on the key and this fact is denoted by the k subscript), so the functions now become:

$$E_K(M) = C$$
$$D_K(C) = M$$

Those functions have the property that (see Figure 1.2):

$$D_K(E_K(M)) = M$$

Some algorithms use a different encryption key and decryption key (see Figure 3). That is, the encryption key, K_1, is different from the corresponding decryption key, K_2. In this case: E

$$E_{K_1}(M) = C$$
$$D_{K_2}(C) = M$$
$$D_{K_2}(E_{K_1}(M)) = M$$

All of the security in these algorithms is based in the key (or keys); none is based in the details of thealgorithm. This means that the algorithm can be published and analyzed. Products using the algorithm can be mass-produced. It doesn't matter if an eavesdropper knows your algorithm; if shedoesn't know your particular key, she can't read your messages.

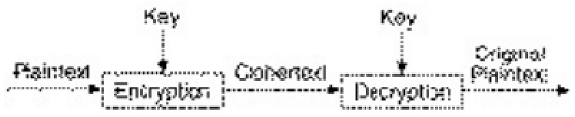


Figure 3 Encryption and decryption with a key.

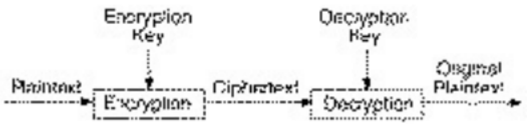


Figure 4 Encryption and decryption with two different keys.

A cryptosystem is an algorithm, plus all possible plaintexts, ciphertexts, and keys.

key-based algorithms

There are two general types of key-based algorithms: symmetric and public-key. Symmetric algorithms, sometimes called conventional algorithms, are algorithms where the encryption key can be calculated from the decryption key and vice versa. In most symmetric algorithms, the encryptionkey and the decryption key are the same. These algorithms, also called secret-key algorithms, singlekey algorithms, or one-key algorithms, require that the sender and receiver agree on a key before they can communicate securely. The security of a symmetric algorithm rests in the key; divulging the key means that anyone could encrypt and decrypt messages. As long as the communication needs toremain secret, the key must remain secret.

Encryption and decryption with a symmetric algorithm are denoted by:

$$E_K(M) = C$$
$$D_K(C) = M$$

Symmetric algorithms

Symmetric algorithmscan be divided into two categories. Some operate on the plaintext a single bit (or sometimes byte) at a time; these are called stream algorithms or stream ciphers. Others operate on the plaintext in groups of bits. The groups of bits are called blocks, and the algorithms are calledblock algorithms or block ciphers. For modern computer algorithms, a typical block size is 64 bits—large enough to preclude analysis and small enough to be workable. (Before computers, algorithms generally operated on plaintext one character at a time. You can think of this as a streamalgorithm operating on a stream of characters.)

Public-Key Algorithms

Public-key algorithms (also called asymmetric algorithms) are designed so that the key used forencryption is different from the key used for decryption. Furthermore, the decryption key cannot (at least in any reasonable amount of time) be calculated from the encryption key. The algorithms are called "public-key" because the encryption key can be made public: A complete stranger can use the encryption key to encrypt a message, but only a specific person with the corresponding decryption key can decrypt the message. In these systems, the encryption key is often called the public key, and the decryption key is often called the private key. The private key is sometimes also called the secret key, but to avoid confusion with symmetric algorithms, that tag won't be used here. Encryption using public key K is denoted by:

$E_K(M) = C$

Even though the public key and private key are different, decryption with the corresponding privatekey is denoted by:

$D_K(C) = M$

Sometimes, messages will be encrypted with the private key and decrypted with the public key; thisis used in digital signatures (see Section 2.6). Despite the possible confusion, these operations are denoted by, respectively:

$E_K(M) = C$

$D_K(C) = M$

Cryptanalysis

The whole point of cryptography is to keep the plaintext (or the key, or both) secret from eavesdroppers (also called adversaries, attackers, interceptors, interlopers, intruders, opponents, or simply the enemy). Eavesdroppers are assumed to have complete access to the communications between the sender and receiver.

Cryptanalysis is the science of recovering the plaintext of a message without access to the key. Successful cryptanalysis may recover the plaintext or the key. It also may find weaknesses in a cryptosystem that eventually lead to the previous results. (The loss of a key through noncryptanalytic means is called a compromise.)

An attempted cryptanalysis is called an attack. A fundamental assumption in cryptanalysis, first enunciated by the Dutchman A. Kerckhoffs in the nineteenth century, is that the secrecy must reside entirely in the key [794]. Kerckhoffs assumes that the cryptanalyst has complete details of the cryptographic algorithm and implementation. (Of course, one would assume that the CIA does not make a habit of telling Mossad about its cryptographic algorithms, but Mossad probably finds out anyway.) While real-world cryptanalysts don't always have such detailed information, it's a good assumption to make. If others can't break an algorithm, even with knowledge of how it works, thenthey certainly won't be able to break it without that knowledge.

There are four general types of cryptanalytic attacks. Of course, each of them assumes that the cryptanalyst has complete knowledge of the encryption algorithm used:

1. Ciphertext-only attack. The cryptanalyst has the ciphertext of several messages, all of which have been encrypted using the same encryption algorithm. The cryptanalyst's job is torecover the plaintext of as many messages as possible, or better yet to deduce the key (or keys) used to encrypt the messages, in order to decrypt other messages encrypted with the same keys.
   Given: $C_1 = E_k(P_1)$, $C_2 = E_k(P_2)$,...$C_i = E_k(P_i)$
   Deduce: Either P1, P2,...Pi; k; or an algorithm to infer Pi+1 from Ci+1 = Ek(Pi+1)
2. Known-plaintext attack. The cryptanalyst has access not only to the ciphertext of several messages, but also to the plaintext of those messages. His job is to deduce the key (or keys) used to encrypt the messages or an algorithm to decrypt any new messages encrypted with the same key (or keys).
   Given: P_1, $C_1 = E_k(P_1)$, P_2, $C_2 = E_k(P_2)$,...P_i, $C_i = E_k(P_i)$
   Deduce: Either k, or an algorithm to infer P_{i+1} from $C_{i+1} = E_k(P_{i+1})$
3. Chosen-plaintext attack. The cryptanalyst not only has access to the ciphertext and associated plaintext for several messages, but he also chooses the plaintext that gets encrypted. This is more powerful than a known-plaintext attack, because the cryptanalyst canchoose specific plaintext blocks to encrypt, ones that might yield more information about thekey. His job is to deduce the key (or keys) used to encrypt the messages or an algorithm to decrypt any new messages encrypted with the same key (or keys).
   Given: P_1, $C_1 = E_k(P_1)$, P_2, $C_2 = E_k(P_2)$,...P_i, $C_i = E_k(P_i)$, where the cryptanalyst getsto choose P_1, P_2,...P_i
   Deduce: Either k, or an algorithm to infer P_{i+1} from $C_{i+1} = Ek(P_{i+1})$
4. Adaptive-chosen-plaintext attack. This is a special case of a chosen-plaintext attack. Not only can the cryptanalyst choose the plaintext that is encrypted, but he can also modify his choice based on the results of previous encryption. In a chosen-plaintext attack, a cryptanalyst might just be able to choose one large block of plaintext to be encrypted; in an adaptivechosen-plaintext attack he can choose a smaller block of plaintext and then choose another based on the results of the first, and so forth.
   a. There are at least three other types of cryptanalytic attack.
5. Chosen-ciphertext attack. The cryptanalyst can choose different ciphertexts to be decrypted and has access to the decrypted plaintext. For example, the cryptanalyst has accessto a tamperproof box that does automatic decryption. His job is to deduce the key.
   Given: $C_1, P_1 = D_k(C_1), C_2, P_2 = D_k(C_2),...C_i, P_i = D_k(C_i)$
   Deduce*: k*
   This attack is primarily applicable to public-key algorithms and will be discussed in Section 19.3. A chosen-ciphertext attack is sometimes effective against a symmetric algorithm as well. (Sometimes a chosen-plaintext attack and a chosen-ciphertext attack are together known as achosen-text attack.)

6. Chosen-key attack. This attack doesn't mean that the cryptanalyst can choose the key; it means that he has some knowledge about the relationship between different keys. It's strangeand obscure, not very practical, and discussed in Section 12.4.
7. Rubber-hose cryptanalysis. The cryptanalyst threatens, blackmails, or tortures someone until they give him the key. Bribery is sometimes referred to as a purchase-key attack. These are all very powerful attacks and often the best way to break an algorithm.

Known-plaintext attacks and chosen-plaintext attacks are more common than you might think. It is not unheard-of for a cryptanalyst to get a plaintext message that has been encrypted or to bribe someone to encrypt a chosen message. You may not even have to bribe someone; if you give a message to an ambassador, you will probably find that it gets encrypted and sent back to his country for consideration. Many messages have standard beginnings and endings that might be known to the cryptanalyst. Encrypted source code is especially vulnerable because of the regular appearance of keywords: #define, struct, else, return. Encrypted executable code has the same kinds of problems: functions, loop structures, and so on. Known-plaintext attacks (and even chosen-plaintext attacks) were successfully used against both the Germans and the Japanese during World War II.

And don't forget Kerckhoffs's assumption: If the strength of your new cryptosystem relies on the fact that the attacker does not know the algorithm's inner workings, you're sunk. If you believe that keeping the algorithm's insides secret improves the security of your cryptosystem more than lettingthe academic community analyze it, you're wrong. And if you think that someone won't disassemble your code and reverse-engineer your algorithm, you're naïve. (In 1994 this happened with the RC4algorithm—see Section 17.1.) The best algorithms we have are the ones that have been made public, have been attacked by the world's best cryptographers for years, and are still unbreakable. (The National Security Agency keeps their algorithms secret from outsiders, but they have the best cryptographers in the world working within their walls—you don't. Additionally, they discuss theiralgorithms with one another, relying on peer review to uncover any weaknesses in their work.)

Cryptanalysts don't always have access to the algorithms, as when the United States broke the Japanese diplomatic code PURPLE during World War II [794]—but they often do. If the algorithmis being used in a commercial security program, it is simply a matter of time and money to disassemble the program and recover the algorithm. If the algorithm is being used in a military communications system, it is simply a matter of time and money to buy (or steal) the equipment and reverse-engineer the algorithm.

Those who claim to have an unbreakable cipher simply because they can't break it are either geniuses or fools. Unfortunately, there are more of the latter in the world. Beware of people who extol the virtues of their algorithms, but refuse to make them public; trusting their algorithms is liketrusting snake oil.

Good cryptographers rely on peer review to separate the good algorithms from the bad.

Security of Algorithms

Different algorithms offer different degrees of security; it depends on how hard they are to break. Ifthe cost required to break an algorithm is greater than the value of the encrypted data, then you're probably safe. If the time required to break an algorithm is longer than the time the encrypted data must remain secret, then you're probably safe. If the amount of data encrypted with a single key is less than the amount of data necessary to break the algorithm, then you're probably safe.

I say "probably" because there is always a chance of new breakthroughs in cryptanalysis. On the other hand, the value of most data decreases over time. It is important that the value of the data always remain less than the cost to break the security protecting it.

Lars Knudsen classified these different categories of breaking an algorithm. In decreasing order of severity :

1. Total break. A cryptanalyst finds the key, K, such that DK(C) = P.
2. Global deduction. A cryptanalyst finds an alternate algorithm, A, equivalent to DK(C), without knowing K.
3. Instance (or local) deduction. A cryptanalyst finds the plaintext of an intercepted ciphertext.
4. Information deduction. A cryptanalyst gains some information about the key or plaintext. This information could be a few bits of the key, some information about the form of the plaintext, and so forth.

An algorithm is unconditionally secure if, no matter how much ciphertext a cryptanalyst has, thereis not enough information to recover the plaintext. In point of fact, only a one-time pad (see Section1.5) is unbreakable given infinite resources. All other cryptosystems are breakable in a ciphertextonly attack, simply by trying every possible key one by one and checking whether the resultingplaintext is meaningful. This is called a brute-force attack (see Section 7.1).

Steganography

Steganography serves to hide secret messages in other messages, such that the secret's very existence is concealed. Generally the sender writes an innocuous message and then conceals a secret message on the same piece of paper. Historical tricks include invisible inks, tiny pin punctures on selected characters, minute differences between handwritten characters, pencil marks on typewritten characters, grilles which cover most of the message except for a few characters, and so on.

III. Symmetric Key Cryptography

Symmetric key ciphers are one of the workhorses of cryptography. They are used to secure bulkdata, provide a foundation for message authentication codes, and provide support for passwordbased encryption as well. As symmetric key cryptography gains its security from keeping a sharedkey secret, it is also often referred to as secret key cryptography, a term that you will see is usedin the JCE.This chapter introduces the concept of symmetric key cryptography and how it is used in the JCE.I will cover creation of keys for symmetric key ciphers, creating Cipher objects to be used forencryption, how modes and padding mechanisms are specified in Java, what other parameterobjects can be used to initialize ciphers and what they mean, how password-based encryption isused, methods for doing key wrapping, and how to do cipher-based I/O.By the end of this chapter you should

- Be well equipped to make use of a variety of symmetric key ciphers
- Understand the various cipher modes and paddings and what they are for
- Be able to construct or randomly generate symmetric keys
- Understand key wrapping
- Be able to utilize the I/O classes provided in the JCE

Finally, you should also have a few ideas about where to look when you are trying to debug applications using symmetric key ciphers and what might go wrong with them.

.

A Basic Utility Class

In the policy test program, you were mainly interested in whether you could create a cipher with a givenkey size and use it. This time you will carry out a simple encryption/decryption process so you can seehow ciphers get used from end to end. Before you can do this, you need to define some basic infrastructure that allows you to look at the output of your programs easily. Encrypted data, as you can imagine,is only human-readable by chance, so for the purposes of investigation, it is best to print the bytes youare interested in using hex, which, being base-16, nicely maps to two digits a byte.Here is a simple utility class for doing hex printing of a byte array:

```
package ;
/**
* General utilities for the second chapter examples.
*/
public class Utils
{
        private static String digits = "0123456789abcdef";
        /**
        * Return length many bytes of the passed in byte array as a hex string.
        *
        * @param data the bytes to be converted.
        * @param length the number of bytes in the data block to be converted.
* @return a hex representation of length bytes of data.
*/
public static String toHex(byte[] data, int length)
{
StringBuffer buf = new StringBuffer();
for (int i = 0; i != length; i++)
        {
                int v = data[i] & 0xff;
                buf.append(digits.charAt(v >> 4));
                buf.append(digits.charAt(v & 0xf));
}
        return buf.toString();
}
/**
* Return the passed in byte array as a hex string.
*
* @param data the bytes to be converted.
* @return a hex representation of data.
*/
public static String toHex(byte[] data)
{
return toHex(data, data.length);
}
}
```

Copy and compile the utility class. Now you have done that, look at the example that follows.

Using AES

Because this example is fairly simple, I'll explain the API I am using after it. However, you should notethat the example is using an algorithm called AES. Prior to November 2001, the stock standard algorithmfor doing symmetric key encryption was the Data Encryption Standard (DES) and a variant on it, namely,Triple-DES or DES-EDE. Now, following the announcement of the Advanced Encryption Standard (AES),your general preference should be to use AES. You will look at some other algorithms a bit later; however,a lot of work went into the development and selection of AES. It makes sense to take advantage of it, soAES is what you'll use in this example.

```
package;
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
/**
* Basic symmetric encryption example
*/
public class SimpleSymmetricExample
{
        public static void main(String[] args) throws Exception
        {
                byte[]          input = new byte[] {
                                        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
                                        (byte)0x88, (byte)0x99, (byte)0xaa, (byte)0xbb,
                                        (byte)0xcc, (byte)0xdd, (byte)0xee, (byte)0xff };
                byte[]          keyBytes = new byte[] {
                                        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                                        0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
                                        0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17 };
        SecretKeySpec   key = new SecretKeySpec(keyBytes, "AES");
        Cipher                  cipher = Cipher.getInstance("AES/ECB/NoPadding", BC");
        System.out.println("input text : " + Utils.toHex(input));
        // encryption pass
        byte[] cipherText = new byte[input.length];
        cipher.init(Cipher.ENCRYPT_MODE, key);
        int ctLength = cipher.update(input, 0, input.length, cipherText, 0);
        ctLength += cipher.doFinal(cipherText, ctLength);
        System.out.println("cipher text: " + Utils.toHex(cipherText)
                                                                + " bytes: " + ctLength);
        // decryption pass
        byte[] plainText = new byte[ctLength];
        cipher.init(Cipher.DECRYPT_MODE, key);
        int ptLength = cipher.update(cipherText, 0, ctLength, plainText, 0);
        ptLength += cipher.doFinal(plainText, ptLength);
        System.out.println("plain text : " + Utils.toHex(plainText)
                                                                + " bytes: " + ptLength);
        }
}
```

Readers who also spend their time browsing through the NIST FIPS publications may recognize this asone of the standard vector tests for AES in FIPS-197. As an aside, if you are planning to get seriouslyinvolved in this area, you would do well to have some familiarity with the relevant NIST FIPS publications. The most relevant ones have been listed in Appendix D, and amongst other things, they are a bighelp if you need to confirm for yourself the validity of an implementation of an algorithm they describe.

If all is going well, and your class path is appropriately set up, when you run the program using

```
        java chapter2.SimpleSymmetricExample
```

you will see

```
                input text : 00112233445566778899aabbccddeeff
                cipher text: dda97ca4864cdfe06eaf70a0ecd7191 bytes: 16
                plain text : 00112233445566778899aabbccddeeff bytes: 16
```

You may also get the exception:

```
        Exception in thread "main" java.security.NoSuchProviderException:
                                                        Provider 'BC' not found
```

which means the provider is not properly installed.

Or you may get the exception:

```
        Exception in thread "main" java.lang.SecurityException:
                                        Unsupported keysize or algorithm parameters
```

which instead means it can find the provider, but the unrestricted policy files are not installed.

If you see either of these exceptions, or have any other problems, look through Chapter 1 again andmake sure the Bouncy Castle provider has been correctly installed and the Java environment is correctlyconfigured.

On the other hand, if everything is working, it is probably time you looked at how the program works.

How It Works

As you can see, the example demonstrates that carrying out a symmetric key encryption operation is amatter of providing a key to use and a suitable object for doing the processing on the input data, be itplaintext to be encrypted or ciphertext to be decrypted. In Java the easiest way to construct a symmetrickey by hand is to use the SecretKeySpec class.

The Secret Key Spec Class

The javax.crypto.spec.SecretKeySpec class provides a simple mechanism for converting byte datainto a secret key suitable for passing to a Cipher object's init() method. As you'll see a bit later, it isnot the only way of creating a secret key, but it is certainly one that is used a lot. Looking at the previousTry It Out ("Using AES"), you can see that constructing a secret key can be as simple as passing a bytearray and the name of the algorithm the key is to be used with. For more details on the class, see theJavaDoc that comes with the JCE.

> One thing the `SecretKeySpec` will not do is stop you from passing a weak key to a `Cipher` object. Weak keys are keys that, for a given algorithm, do not provide strong cryptography and should be avoided. Not all algorithms have weak keys, but if you are using one that does, such as DES, you should take care to ensure that the bytes produced for creating the `SecretKeySpec` are not weak keys.

The Cipher Class

A look at the previous example program quickly reveals that the creation and use of a javax.crypto.Cipher object follows a simple pattern. You create one using Cipher.getInstance(), initialize itwith the mode you want using Cipher.init(), feed the input data in while collecting output at thesame time using Cipher.update(), and then finish off the process with Cipher.doFinal().

Cipher.getInstance()

ACipher object, rather than being created using a constructor directly, is created using the static factorymethod Cipher.getInstance(). In the case of the example, it was done by passing two arguments,one giving the kind of cipher you want to create, the other giving the provider you want to use to createit — given by the name "BC".

In the case of the cipher name "AES/ECB/NoPadding", the name is composed of three parts. The firstpart is the name of algorithm — AES. The second part is the mode in which the algorithm should beused, ECB, or Electronic Code Book mode. Finally, the string "NoPadding" tells the provider you donot wish to use padding with this algorithm. Just ignore the mode and padding, as you will be readingabout them soon. What is most important now is that when the full name of the Cipher object you wantto be created is given, it always follows the AlgorithmName/Mode/TypeOfPadding pattern. You canalso just give the algorithm name and provider, as in:

```
                        Cipher.getInstance("AES", "BC");
```

However if you do, it is purely up to the provider you are using as to which mode and padding willbe used in the Cipher object that has been returned. It is advised against doing this in the interests ofallowing your code to be portable between providers. Specify exactly what you need and you should bespared unnecessary surprises.

Cipher.init()

Having created a Cipher object using Cipher.getInstance() at a minimum, you then have to initialize it with the type of operation it is to be used for and with the key that is to be used. Cipher objectshave four possible modes, all specified using static constants on the Cipher class. Two of the modes areconnected with key wrapping, which you'll look at later, and the other two are Cipher.ENCRYPT_MODEand Cipher.DECRYPT_MODE, which were used previously. The Cipher.init() method can take otherparameters as well, which you'll look at later. For the moment it is enough to understand that if you donot call the init method, any attempt to use the Cipher object will normally result in anIllegalStateException.

Cipher.update()

Once the Cipher object is set up for encryption or decryption, you can feed data into it and accept datafrom it. There are several convenience update methods on the Cipher class that you can read about inthe JavaDoc, but the one used in the example is the most fundamental. Consider the line:

int ctLength = cipher.update(input, 0, input.length, cipherText, 0);

Cipher objects usually acquire a chunk of data, process it by copying the result into the output array (theargument cipherText), and then copy the next chunk and continue, filling the output array as they go.Thus, you cannot be sure how much data will be written each time you do an update; the number ofoutput bytes may be O (zero), or it may be between 0 and the length of the input. The starting offset thatthe processed blocks are written to is the last argument to the method, in this case 0. Regardless of howmany bytes get output during an update, you will only know how many bytes have been written to theoutput array if you keep track of the return value.

Cipher.doFinal()

Now consider the line:

ctLength += cipher.doFinal(cipherText, ctLength);

Cipher.doFinal() is very similar to Cipher.update() in that it may also put out 0 or more bytes,depending on the kind of Cipher object you specified with Cipher.getInstance(). Likewise, italso has a return value to tell you how many bytes it actually wrote to the output array (again thecipherText array). Note that the second argument is the offset at which writing of the output will startand is a value that has been preserved from the last Cipher.update().

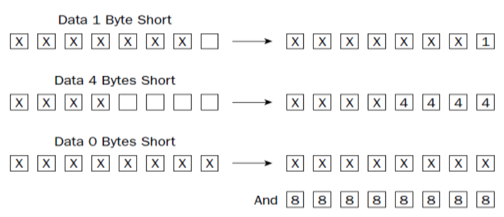
Failing to keep track of the return values from the `int` returning methods of `update()` and `doFinal()` is one of the most common error programmers make using the `Cipher` class. It is never okay to ignore the return values if you want to write flexible code.

Symmetric Block Cipher Padding

While the value of the test vector used in the last example was not due to random chance, neither wasthe length of it. Most of the popular ciphers are block ciphers, and their block size is normally more than1 byte long; DES and Blowfish, for example, have a block size of 8 bytes. AES, the latest addition to thefamily, has a block size of 16 bytes. The effect of this is that the input data to a cipher that is being usedin a blocking mode must be aligned to the block size of that cipher. Truth is, for most of us, the data wewish to encrypt is not always going to be a multiple of the block size of the encryption mechanism wewant to use. So while we can find out what the block size is using the Cipher.getBlockSize() methodand then try to take it into account, the easiest way to deal with this issue is to use padding mechanisms.

PKCS #5/PKCS #7 Padding

PKCS #5 padding was originally developed for block ciphers with a block size of 8 bytes. Later, with thewriting of PKCS #7, the authors of the standard specified a broader interpretation of the padding mechanism, which allowed for the padding mechanism to be used for block sizes up to 255 bytes. The PKCS inPKCS #5 and PKCS #7 comes from Public-Key Cryptography Standards that were developed by RSASecurity. They are also worth a read, and a list of the most relevant appears.



PKCS #5/#7 Padding with an 8 Byte Block Cipher

Figure 2-1

we can see from Figure 2-1 that the padding mechanism is quite simple; if you need to pad a block ofdata where the last input block is 3 bytes shorter than the block size of the cipher you are using; you add3 bytes of value 3 to the data before encrypting it. Then when the data is decrypted, you check the lastbyte of the last decrypted block of data and remove that many bytes from it. The only shortcoming ofthis approach is that you must always add the padding, so if the block size of your cipher is 8 bytes andyour data is a multiple of 8 bytes in length, you have to add a pad block with 8 bytes with the value 8 toyour data before you encrypt it, and as before, remove the extra 8 bytes at the other end when the data isdecrypted. The advantage of this approach is that the mechanism is unambiguous.

IV. Asymmetric Key Cryptography

Modern computing has generated a tremendous need for convenient, manageableencryption technologies. Symmetric algorithms, such as Triple DESand Rijndael, provide efficient and powerful cryptographic solutions, especiallyfor encrypting bulk data. However, under certain circumstances, symmetricalgorithms can come up short in two important respects: key exchangeand trust. In this chapter we consider these two shortcomings and learn howasymmetric algorithms solve them. We then look at how asymmetric algorithmswork at a conceptual level in the general case, with emphasis on theconcept of trapdoor one-way functions. This is followed by a more detailedanalysis of RSA, which is currently the most popular asymmetric algorithm.Finally, we see how to use RSA in a typical program using the appropriate.NET Security Framework classes.

We focus on the basic idea of asymmetric algorithms, and we look atRSA in particular from the encryption/decryption point of view.

Problems with Symmetric Algorithms

One big issue with using symmetric algorithms is the key exchange problem,which can present a classic catch-22. The other main issue is the problem oftrust between two parties that share a secret symmetric key. Problems of trust may be encountered when encryption is used for authentication and integritychecking., a symmetric key can be used to verify theidentity of the other communicating party, but as we will now see, this requiresthat one party trust the other.

The Key Exchange Problem

The key exchange problem arises from the fact that communicating partiesmust somehow share a secret key before any secure communication can beinitiated, and both parties must then ensure that the key remains secret. Ofcourse, direct key exchange is not always feasible due to risk, inconvenience,and cost factors. The catch-22 analogy refers to the question of how tosecurely communicate a shared key before any secure communication can beinitiated.

In some situations, direct key exchange is possible; however, much commercialdata exchange now takes place between parties that have never previouslycommunicated with one another, and there is no opportunity toexchange keys in advance. These parties generally do not know one anothersufficiently to establish the required trust (a problem described in the next section)to use symmetric algorithms for authentication purposes either. With theexplosive growth of the Internet, it is now very often a requirement that partieswho have never previously communicated be able to spontaneously communicatewith each other in a secure and authenticated manner. Fortunately, thisissue can be dealt with effectively by using asymmetric algorithms.

The Trust Problem

Ensuring the integrity of received data and verifying the identity of the sourceof that data can be very important. For example, if the data happens to be acontract or a financial transaction, much may be at stake. To varying degrees,these issues can even be legally important for ordinary email correspondence,since criminal investigations often center around who knew what and whenthey knew it. A symmetric key can be used to check the identity of the individualwho originated a particular set of data, but this authentication schemecan encounter some thorny problems involving trust.

Using Asymmetric Cryptography

To use asymmetric cryptography, Bob randomly generates a public/private keypair.He allows everyone access to the public key, including Alice. Then, whenAlice has some secret information that she would like to send to Bob, sheencrypts the data using an appropriate asymmetric algorithm and the public keygenerated by Bob. She then sends the resulting ciphertext to Bob. Anyone whodoes not know the matching secret key will have an enormously difficult timeretrieving the plaintext from this ciphertext, but since Bob has the matchingsecret key (i.e., the trapdoor information), Bob can very easily discover the original plaintext. Figure 4–1 shows how asymmetric cryptography is used.

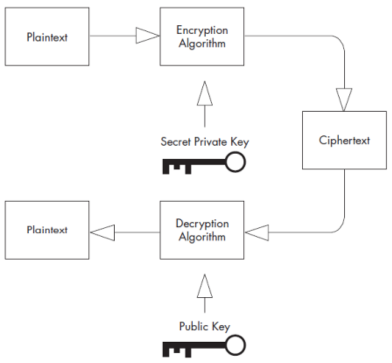


Fig 1.How asymmetric cryptography is used.

The Combination Lock Analogy

A traditional symmetric cipher is analogous to a lockbox with a combinationlock that has one combination used both to open it and close it.The analogyfor an asymmetric cipher is a somewhat stranger device: The single lock hastwo distinct combinations, one for opening it and another for closing it. Bykeeping one of these combinations secret and making the other combinationpublic, you can effectively control who can place or remove the contents inthe lockbox. This added flexibility supports two useful scenarios: confidentialitywithout prior key exchange and data integrity enforcement.

CONFIDENTIALITY WITHOUT PRIOR KEY EXCHANGE

Here is the first scenario. If you know the public combination for closing thelock but not the private combination for opening the lock, then once youhave placed something into the box and locked it, it becomes impossible foranybody who does not know the private opening combinationto obtain thecontents. This demonstrates spontaneous confidentiality (i.e., keeping a secretwithout prior key exchange). Hence, we have a solution to the key exchangeproblem described earlier.

ENFORCING DATA INTEGRITY

The other scenario is if only you know the private combination for closingthe lock, and you place contents into the lockbox and then lock it. Then anyonecan open the lock, but nobody else can lock other contents into thelockbox, since nobody else knows the private combination for closing thelock. Therefore, nobody else can tamper with its contents and then close thelock again. You might think that this is easy to defeat, since anyone couldeasily create his or her own key pair and then lock any data into the lockbox.However, only the newly created public key would then work, and theoriginal public key would fail to unlock the lockbox. Therefore, anyone withknowledge of the original public key would not be fooled by such an attack.Since tampering is detectable, this scenario demonstrates how data integritycan be enforced.

Advantages of the Asymmetric Approach

With the asymmetric (also known as public key) approach, only the privatekey must be kept secret, and that secret needs to be kept only by one party.This is a big improvement in many situations, especially if the parties have noprevious contact with one another. However, for this to work, the authenticityof the corresponding public key must typically be guaranteed somehow by atrusted third party, such as a CA. Because the private key needs to be keptonly by one party, it never needs to be transmitted over any potentially compromisednetworks. Therefore, in many cases an asymmetric key pair mayremain unchanged over many sessions or perhaps even over several years.Another benefit of public key schemes is that they generally can be used toimplement digital signature schemes that include nonrepudiation. Finally,because one key pair is associated with one party, even on a large network,the total number of required keys is much smaller than in the symmetric case.

Combining Asymmetric and Symmetric Algorithms

Since there is no secret key exchange required in order to use asymmetricalgorithms, you might be tempted to solve the symmetric key exchange problemby simply replacing the symmetric algorithm with an asymmetric algorithm.However, that would be like throwing the baby out with the bath water.We still want to take advantage of the superior speed and security offered by
symmetric algorithms, so, instead, we actually combine the two (and sometimesmore than two) algorithms.

Existing Asymmetric Algorithms

Recall that the only information that needs to be shared before initiating symmetricencryption is the secret key. Since this key is typically very small (typicallyno greater than 256 bits) compared to the bulk data (which could bemegabytes) that must be encrypted, it makes sense to use the asymmetricalgorithm to encrypt only the secret symmetric key, and then use this symmetrickey for encrypting the arbitrarily large bulk message data. The secret symmetric key is often referred to as asession keyin this scenario.

.

RSA: The Most Used Asymmetric Algorithm

The most common asymmetric cipher currently in use is RSA, which is fullysupported by the .NET Security Framework. Ron Rivest, Adi Shamir, andLeonard Adleman invented the RSA cipher in 1978 in response to the ideasproposed by Hellman, Diffie, and Merkel. Later in this chapter, we shall seehow to use the high-level implementation of RSA provided by the .NET SecurityFramework. But first, let's look at how RSA works at a conceptual level.

Underpinnings of RSA

Understanding the underpinnings of RSA will help you to develop a deeperappreciation of how it works. In this discussion we focus on the concepts ofRSA, and in Appendix B we look at two examples of implementing RSA fromscratch. One of these examples isTinyRSA, which is a toy version that limitsits arithmetic to 32-bit integers, and the other is a more realistic, multiprecisionimplementation namedBigRSA. You will probably never implement your ownRSA algorithm from scratch, since most cryptographic libraries, including the.NET Security Framework, provide excellent implementations (i.e., probablybetter than I could do). However, the RSA examples in Appendix B shouldhelp you to fully understand what goes on in RSA at a deeper level.

Here is how RSA works. First, we randomly generate a public and privatekey pair. As is always the case in cryptography, it is very important togenerate keys in the most random and therefore, unpredictable manner possible.Then, we encrypt the data with the public key, using the RSA algorithm.Finally, we decrypt the encrypted data with the private key and verify that itworked by comparing the result with the original data. Note that we areencrypting with the public key and decrypting with the private key. Thisachieves confidentiality. In the next chapter, we look at the flip side of thisapproach, encrypting with the private key and decrypting with the public key,to achieve authentication and integrity checking.

Here are the steps for generating the public and private key pair.

1. Randomly select two prime numbers p and q . For the algebra to work properly, these two primes must not be equal. To make the cipher strong, these prime numbers should be large, and they should be in the form of arbitrary precision integers with a size of at least 1024 bits.
2. Calculate the product: $n = p \cdot q$.
3. Calculate the Euler totient for these two primes, which is represented by the Greek letter φ . This is easily computed with the formula $\varphi = (p - 1) \cdot (q - 1)$.
4. Now that we have the values n and φ , the values p and q will no longer be useful to us. However, we must ensure that nobody else will ever be able to discover these values. Destroy them, leaving no trace behind so that they cannot be used against us in the future. Otherwise, it will be very easy for an attacker to reconstruct our key pair and decipher ourciphertext.
5. Randomly select a number e (the letter e is used because we will use this value during encryption) that is greater than 1, less than φ , and relatively prime to φ . Two numbers are said to be relatively prime if they have no prime factors in common. Note that e does not necessarily have to be prime. The value of e is used along with the value n to represent the public key used for encryption.
6. Calculate the unique value d (to be used during decryption) that satisfiesthe requirement that, if $d \cdot e$ is divided by φ, then the remainder of the division is 1. The mathematical notation for this is $d \cdot e = 1(\text{mod } \varphi)$. In mathematical jargon, we say that d is the multiplicative inverse of e modulo φ. The value of d is to be kept secret. If you know the value of φ, the value of d can be easily obtained from e using a technique known as the Euclidean algorithm. If you know n (which is public), but not p or q (which have been destroyed), then the value of φ is very hard to determine. The secret value of d together with the value n represents the private key.

Once we have generated a public/private key pair, we can encrypt amessage with the public key with the following steps.

1. Take a positive integer m to represent a piece of plaintext message. In order for the algebra to work properly, the value of m must be less than the modulus n, which was originally computed as $p \cdot q$. Long messages must therefore be broken into small enough pieces that each piece can be uniquely represented by an integer of this bit size, and each piece is then individually encrypted.

2. Calculate the ciphertext c using the public key containing e and n. This is calculated using the equation c = me(mod n).

Finally, we can perform the decryption procedure with the private keyusing the following steps.

1. Calculate the original plaintext message from the ciphertext using the private key containing d and n. This is calculated using the equation m = cd(mod n).
2. Compare this value of m with the original m, and you should see that they are equal, since decryption is the inverse operation to encryption.

Caveat: Provability Issues

Every asymmetric algorithm is based on some trapdoor one-way function.This leads to a critically important question: How do we know for certainthat a particular function is truly one-way? Just because nobody has publiclydemonstrated a technique that allows the inverse function to be calculatedquickly does not actually prove anything about the security of the algorithm.

If somebody has quietly discovered a fast technique to compute the inversefunction, he or she could be busily decrypting enormous amounts of ciphertextevery day, and the public may never become aware of it. It may seemparanoid, but high on the list of suspicions are major governments, sincethey employ large numbers of brilliant mathematicians who are outfitted withthe most powerful computing resources available, putting them in a betterposition than most for cracking the trapdoor.

.

Programming with .NET Asymmetric Cryptography

In this section, we look at the RSAAlgorithm and SavingKeysAsXml exampleprograms provided for this chapter. These two code examples show howto encrypt and decrypt using the RSA algorithm as well as how to store andretrieve key information using an XML format. The RSA code example uses the concrete RSACryptoServiceProvider class. Figure 4–2 shows where thisclass resides in the class hierarchy, under the abstract AsymmetricAlgorithmclass.

An RSA Algorithm Example

The RSAAlgorithm example uses the Encrypt method of the RSACryptoServiceProviderclass. This method takes two parameters, the first ofwhich is a byte array containing the data to be encrypted. The second parameteris a boolean that indicates the padding mode to be used. Padding isrequired, since the data to be encrypted is usually not the exact number ofrequired bits in length. Since the algorithm requires specific bit-sized blocksto process properly, padding is used to fill the input data to the desiredlength. If this second parameter is true, then the improved OAEP16 padding isused. Otherwise, the traditional PKCS#1 v1.5 padding is used. PKCS#1 v1.5has been traditionally the most commonly used padding scheme for RSAusage. However, it is recommended that all new RSA applications that will bedeployed on platforms that support OAEP should use OAEP. Note that OAEPpadding is available on Microsoft Windows XP and Windows 2000 with the

Namespace: System.Security.Cryptography

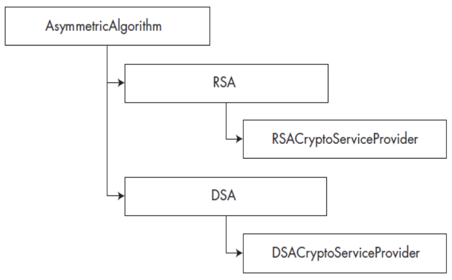


Fig 2.The asymmetric algorithm class hierarchy.

high-encryption pack installed. Unfortunately, previous versions of Windowsdo not support OAEP, which will cause the Encrypt method, with the secondparameter set to true, to throw a CryptographicException. The Encryptmethod returns the resulting encrypted data as a byte array. Here is the syntaxfor the Encrypt method.

```
public byte[] Encrypt(
byte[] rgb,
bool fOAEP
);
```

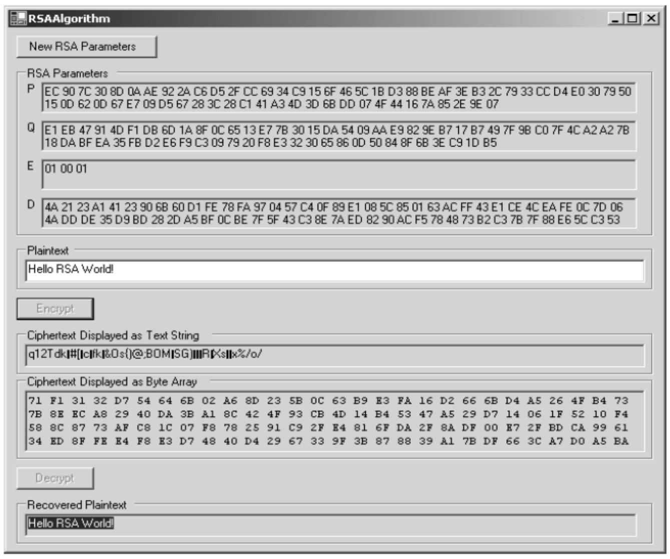
The complementary method to Encrypt is of course Decrypt. You canprobably guess how it works. The second parameter is a byte array containingthe ciphertext to be decrypted. The second parameter is the same as thatin the Encrypt method, indicating the padding mode, as described previously.The return value is a byte array that will contain the resulting recoveredplaintext. Here is the syntax for the Decrypt method.

```
        public byte[] Decrypt(
        byte[] rgb,
        bool fOAEP
        )
```

Figure 4–3 shows the RSAAlgorithm example being used to encryptand decrypt a plaintext message. You enter the plaintext in the TextBox atthe top of the form. You then click on the Encrypt button, which fills in all butthe last form field, including the resulting ciphertext and RSA parameters thatwere used. You then click on the Decrypt button, which displays the recoveredplaintext in the field at the bottom of the form. Of course, the recoveredplaintext should be identical to the original plaintext.

Now let's look at the code in the RSAAlgorithm example code. ThebuttonEncrypt_Click method is called when the user clicks on the Encryptbutton. This encrypts the contents of the plaintext textbox using the establishedpublic RSA key. The public/private RSA key pair is provided by theprogram automatically when it starts, but it may subsequently be changedusing the New RSA Parameters button. There are a few places in the codewhere user interface elements are being enabled and disabled, which are notgermane to our focus on RSA functionality. Therefore, these user interfacecode sections are ignored here. If you are curious about how these user interfacedetails work, please study the simple code sections following each of the//do UI stuff comments.

We first generate the initial RSA parameters by calling the GenerateNewRSAParams method in the RSAAlgorithm_Load method. The GenerateNewRSAParams method is also called each time the user clicks on the New RSAParameters button, which is handled by the buttonNewRSAParams_Clickmethod. The GenerateNewRSAParams method is very simple. It just creates



```
RSA Parameters
P  EC 90 7C 30 8D 0A AE 92 2A C6 D5 2F CC 69 34 C9 15 6F 46 5C 1B D3 88 BE AF 3E B3 2C 79 33 CC D4 E0 30 79 50
   15 0D 62 0D 67 E7 09 D5 67 28 3C 28 C1 41 A3 4D 3D 6B DD 07 4F 44 16 7A 85 2E 9E 07
Q  E1 EB 47 91 4D F1 DB 6D 1A 8F 0C 65 13 E7 7B 30 15 DA 54 09 AA E9 82 9E B7 17 B7 49 7F 9B C0 7F 4C A2 A2 7B
   18 DA BF EA 35 FB D2 E6 F9 C3 09 79 20 F8 E3 32 30 65 86 0D 50 84 8F 6B 3E C9 1D B5
E  01 00 01
D  4A 21 23 A1 41 23 90 6B 60 D1 FE 78 FA 97 04 57 C4 0F 89 E1 08 5C 85 01 63 AC FF 43 E1 CE 4C EA FE 0C 7D 06
   4A DD DE 35 D9 BD 28 2D A5 BF 0C BE 7F 5F 43 C3 8E 7A ED 82 90 AC F5 78 48 73 B2 C3 7B 7F 88 E6 5C C3 53

Plaintext
Hello RSA World!

Ciphertext Displayed as Text String
q12Tdk▌#[▌c▌fk▌&0s{)@;BOM▌SG)▌▌▌R▌Xs▌▌x%/o/

Ciphertext Displayed as Byte Array
71 F1 31 32 D7 54 64 6B 02 A6 8D 23 5B 0C 63 B9 E3 FA 16 D2 66 6B D4 A5 26 4F B4 73
7B 8E EC A8 29 40 DA 3B A1 8C 42 4F 93 CB 4D 14 B4 53 47 A5 29 D7 14 06 1F 52 10 F4
58 8C 87 73 AF C8 1C 07 F8 78 25 91 C9 2F E4 81 6F DA 2F 8A DF 00 E7 2F BD CA 99 61
34 ED 8F FE E4 F8 E3 D7 48 40 D4 29 67 33 9F 3B 87 88 39 A1 7B DF 66 3C A7 D0 A5 BA

Recovered Plaintext
Hello RSA World!
```

Fig 3 The RSAAlgorithm example program.

an RSACryptoServiceProvider class object, stores its public and private RSAparameters by calling the RSA class's ExportParameters method, and displaysa few of the more important of these parameters in the user interface. TheseRSA parameters are actually stored in two fields of type RSAParameters. TheRSAParameters field named rsaParamsExcludePrivate gets a copy of thepublic-only RSA parameters (i.e., the modulus and exponent values only),which is required for encryption purposes in the buttonEncrypt_Clickmethod. The other RSAParameters field, named rsaParamsIncludePrivategets a copy of the combined public and private RSA parameters, which isrequired in the buttonDecrypt_Click method.

Here is the GenerateNewRSAParams method. Note that the Export Parameters method is called twice. The first time, the parameter passed intothis method is true, and the second time, it is false. Passing true indicates thatyou want to include all key parameter information, including the private keyinformation. False indicates that only the public key information is to bestored. We separate these cases into two distinct fields to demonstrate how the encryption will use only the public information, but the decryption willuse both the public and private key information. This is a crucial point inunderstanding asymmetric cryptography. This would perhaps be even clearerif we broke the encryption and decryption portions of this example into twoseparate applications, but this example is provided as a simple monolithicprogram purely for easy study. You should at some point take a moment toverify that the encryption and decryption functions in this program do indeeduse only their own appropriate version of this RSA parameter information,using the corresponding ImportParameters method.

```
private void GenerateNewRSAParams()
```

```
{
        //establish RSA asymmetric algorithm
        RSACryptoServiceProvider rsa =
                new RSACryptoServiceProvider();
        //provide public and private RSA params
        rsaParamsIncludePrivate =
                rsa.ExportParameters(true);
        //provide public only RSA params
        rsaParamsExcludePrivate =
                rsa.ExportParameters(false);
```

When we create an instance of the RSACryptoServiceProvider class,we actually get the RSA implementation provided by the underlying cryptographicservice provider (CSP). This class is directly derived from the RSAclass. The RSA class allows other RSA implementations to be implemented asother derived classes; however, the CSP implementation is currently the onlyone available.

The two fields that store the RSA parameter information when Export Parameters is called are declared as RSAParameters type fields, as shownin the following code snippet. The rsaParamsExcludePrivate filed will beused for encryption, and the rsaParamsIncludePrivate field will be used indecryption in this example.

```
//public modulus and exponent used in encryption
RSAParameters rsaParamsExcludePrivate;
//public and private RSA params use in decryption
RSAParameters rsaParamsIncludePrivate;
```

In the buttonEncrypt_Click method we then create a new instance ofRSACryptoServiceProvider class, and we initialize it with the stored publickey information by calling the RSA object's ImportParameters method,specifying rsaParamsExcludePrivate as the parameter. Next, we obtain theplaintext in the form of a byte array named plainbytes. Finally, we performthe main function of this method by calling on the Encrypt method of theRSA object. This returns another byte array, which is an instance field namedcipherbytes. This is an instance field rather than a local variable, because weneed to communicate this byte array to the decryption method, and local variablesare not maintained across method calls.

```
private void buttonEncrypt_Click(
object sender, System.EventArgs e)
{
        //do UI stuff
        ...
        //establish RSA using parameters from encrypt
        RSACryptoServiceProvider rsa =
                new RSACryptoServiceProvider();
        //import public only RSA parameters for encrypt
        rsa.ImportParameters(rsaParamsExcludePrivate);
        //read plaintext, encrypt it to ciphertext
        byte[] plainbytes =
                Encoding.UTF8.GetBytes(textPlaintext.Text);
        cipherbytes =
        rsa.Encrypt(
        plainbytes,
        false); //fOAEP needs high encryption pack
//display ciphertext as text string
...
//display ciphertext byte array in hex format
...
//do UI stuff
...
}
...
//variable communicated from encrypt to decrypt
byte[] cipherbytes;
```

The buttonDecrypt_Click method is called when the user clicks onthe Decrypt button. Again, an RSA object is created. The RSA object isrepopulated with the information provided by calling the RSA object'sImportParameters method, but this time, the parameter to this method isthe rsaParamsIncludePrivate, which includes both public and private RSAkey information. The plaintext is then obtained by calling the Decryptmethod of the RSA object. Since a matching set of RSA algorithm parameterswere used for both encryption and decryption, the resulting plaintext matchesperfectly with the original plaintext.

```
private void buttonDecrypt_Click(
        object sender, System.EventArgs e)
{
        //establish RSA using parameters from encrypt
        RSACryptoServiceProvider rsa =
                new RSACryptoServiceProvider();
        //import public and private RSA parameters
        rsa.ImportParameters(rsaParamsIncludePrivate);
        //read ciphertext, decrypt it to plaintext
        byte[] plainbytes =
                rsa.Decrypt(
                cipherbytes,
                false); //fOAEP needs high encryption pack
        //display recovered plaintext
        ...
        //do UI stuff
                ...
        }
        ...
//variable communicated from encrypt to decrypt
```

IV. Conclusion

Transaction security is vital in e-commerce. Hesitation or scepticism in transaction security over the Internet is a crucial issue needs to be taken care seriously. People are happy with the development of the web where they can browse the Internet and find information they need easily. However, when it comes to decide to buy a product/service over the Internet many people worry about the transaction security. Similarly, firms worry about online frauds. According to the study conducted by the World-wide E-Commerce Fraud Prevention Network13, half of US businesses believe that online fraud is a significant problem, while almost 10 percent say it is “the most significant problem” they face. The study also found that only 1 percent of companies say they “never worry” about online fraud.

Encryption technology discussed in this paper is the key technology to make online transaction over the Internet secure. Of course no one can guarantee 100% 13security. Fraud exists in current commerce systems: cash can be counterfeited, checks altered, credit card numbers stolen. Yet these systems are still successful because the benefits and conveniences outweigh the losses. Similarly fraud will still exist in ecommerce even though encryption technology is good enough to protect electronic transactions, but at least a good encryption technology can reduce fraud significantly.The Internet backbone was set up with United States Government money and support, and the principleof an information superhighway is supported by the US Government. However, there is a strongimpulse in the US and other countries to claw back political control over the Internet. Particularlyproblematic is the unprecedented scope of surveillance methods. These measures, being put in placepossibly before the American people fully grasp the significance of them, may become the status quo,and difficult to shift in the future. However, in the area of cryptography, the US is facing a quietrebellion on a number of fronts. One is the domestic resistance to the key forfeiture proposals andlegislation which electronic civil rights activists believe will infringe individual privacy and freedom ofAmericans. The recent strong lobbying efforts by the Internet community in 1995 in respect of the ExonCommunications Decency Act (where the Internet community believed legislation to control offensivematerial would damage the Net), and the resulting turnaround between the Senate passing theCommunications Decency Act legislation and the Congress passing the Cox-Wylie Amendment, (amore low-key and practical approach to the problem) would indicate the Internet community inAmerica is rapidly learning to use its teeth. Another advance is the pragmatic arming of other countrieswith the weapons of future commerce, such as cryptography, securing of electronic communicationsagainst piracy and damage, and Internet access and literacy. These factors are likely to proceed to thepoint where the US technological supremacy may be under threat, and deregulation of cryptographywill become unavoidable. Economic and defence adjustments would then have to be made. However, itis possible these may be more to the perceptions of Americans, rather than to the possibility that due tosecure encrypted communications, the American economy may suffer disastrous damage, taxes willsuddenly not be paid, the war against drugs will be lost completely, and bombers will run amok.

Security in the Internet is improving. The increasing use of the Internet for commerce is improving the deployed technology to protect the financial transactions. Extension of the basic technologies to protect multicast communications is possible and can be expected to be deployed as multicast becomes more widespread.

Control over routing remains the basic tool for controlling access to streams. Implementing particular policies will be possible as multicast routing protocols improve. Cryptography is a tool which may alleviate many of the perceived problems of using the Internet for communications. However, cryptography requires the safe implementation of complex mathematical equations and protocols, and there are always worries about bad implementations. A further worry is that users are integral to securing communications, since they must provide appropriate keys. As the founders of First Virtual point out cybercommerce...a safe application of cryptographic technology will pay close attention to how public keys are associated with user identities, how stolen keys are detected and revoked and how long a stolen key is useful to a criminal.

Cryptography may be groovy technology, but since security is a human issue, cryptography is only as good as the practices of the people who use it. Users leave keys lying around, choose easily remembered keys, don't change keys for years. The complexity of cryptography effectively puts it outside the understanding of most people and so motivation for the practices of cryptographic security is not available.

Thischapter Cryptography and Network Security CSE Technical Seminar speaks about Cryptography and its importance in securing the data from any foreign influence. 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 hieroglyphs in an inscription. In data and telecommunications, cryptography is necessary when communicating over any untrusted medium, which includes just about any network, particularly the Internet. Within the context of any application-to-application communication, there are some specific security requirements. The research abstract explains in depth all the above mentioned aspects.

References:-

[1]. ANSI X9.19, Financial Institution Retail Message Authentication, American Bankers Association, August 13, 1986.

[2]. ANSI X9.52, Triple Data Encryption Algorithm Modes of Operation, American Bankers Association, 1998.

[3]. E. Barkan, E. Biham, N. Keller, Instant ciphertext-only cryptanalysis of GSM encrypted communication, Advances in Cryptology, Proceedings Crypto'03, LNCS 2729, D. Boneh, Ed., Springer, Heidelberg, 2003, pp. 600{616.

[4]. M. Bellare, New proofs for NMAC and HMAC: Security without collisionresistance, Advances in Cryptology, Proceedings Crypto'06, LNCS 4117, C. Dwork, Ed., Springer, Heidelberg, 2006, pp. 602{619.

[5]. M. Bellare, R. Canetti, H. Krawczyk, Keying hash functions for message authentication, Advances in Cryptology, Proceedings Crypto'96, LNCS 1109, N. Koblitz, Ed., Springer, Heidelberg, 1996, pp. 1{15.

[6]. M. Bellare, C. Namprempre, Authenticated encryption: Relations among notions and analysis of the generic composition paradigm, Advances in Cryptology, Proceedings Asiacrypt'00, LNCS 1976, T. Okamoto, Ed. (Springer, Heidelberg, 2000) 531{545.

[7]. M. Bellare, P. Rogaway, Random oracles are practical: A paradigm for designing efficient protocols, Proceedings ACM Conference on Computer and Communications Security (ACM Press 1993) 62{73.

[8]. M. Bellare, P. Rogaway, The exact security of digital signatures { How to sign with RSA and Rabin, Advances in Cryptology, Proceedings Eurocrypt'96, LNCS 1070, U. Maurer, Ed., Springer, Heidelberg, 1996, pp. 399{416.

[9]. D.J. Bernstein, The Poly1305-AES message-authentication code, Fast Software Encryption, LNCS 3557, H. Gilbert and H. Handschuh, Eds. (Springer, Heidelberg, 2005) 32{49.

[10]. D.J. Bernstein, Cache-timing attacks on AES, preprint, 2005, http://cr.yp.to/ papers.html#cachetiming

[11]. A. Biryukov, D. Khovratovich, \Related-key cryptanalysis of the full AES-192 and AES-256," Advances in Cryptology, Proceedings Asiacrypt'09, LNCS 5912, M. Matsui, Ed., Springer, Heidelberg, 2009, pp. 1{18.

[12]. J. Black, Authenticated encryption, Encyclopedia of Cryptography and Security H. van Tilborg, Ed., Springer, Heidelberg, 2005, pp. 11{21.

[13]. J. Black, S. Halevi, H. Krawczyk, T. Krovetz, P. Rogaway, UMAC: Fast and secure message authentication, Advances in Cryptology, Proceedings Crypto'99, LNCS 1666, M.J. Wiener, Ed., Springer, Heidelberg, 1999, pp. 216{233.

[14]. J. Black, P. Rogaway, CBC-MACs for arbitrary length messages, Advances in Cryptology, Proceedings Crypto'00, LNCS 1880, M. Bellare, Ed. (Springer, Heidelberg, 2000) 197{215.

[15]. D. Bleichenbacher, Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1, Advances in Cryptology, Proceedings Crypto'98, LNCS 1462, H. Krawczyk, Ed., Springer, Heidelberg, 1998, pp. 1{12.

[16]. D. Bleichenbacher, Forging some RSA signatures with pencil and paper, Presented at the Rump Session of Crypto 2006.

[17]. D. Boneh, R. DeMillo, R. Lipton, On the importance of checking cryptographic protocols for faults, Advances in Cryptology, Proceedings Eurocrypt'97, LNCS 1233, W. Fumy, Ed., Springer, Heidelberg, 1997, pp. 37{51.

[18]. D. Boneh, A. Joux, P.Q. Nguyen, Why textbook ElGamal and RSA encryption are insecure, Advances in Cryptology, Proceedings Asiacrypt'00, LNCS 1976, T. Okamoto, Ed. (Springer, Heidelberg, 2000) 30{43.

[19]. A. Bosselaers, H. Dobbertin, B. Preneel, The RIPEMD-160 cryptographic hash function, Dr. Dobb's Journal, Vol. 22, No. 1, January 1997, pp. 24{28.

[20]. B. Canvel, A.P. Hiltgen, S. Vaudenay, M. Vuagnoux, \Password interception in a SSL/TLS Channel," Advances in Cryptology, Proceedings Crypto'03, LNCS 2729, D. Boneh, Ed., Springer, Heidelberg, 2003, pp. 583{599.

[21]. S. Contini, Y.L. Lin, Forgery and partial key recovery attacks on HMAC and NMAC using hash collisions Advances in Cryptology, Proceedings Asiacrypt'06, LNCS 4284, X. Lai and K. Chen, Eds., Springer, Heidelberg, 2006, pp. 37{53.

[22]. J.-S. Coron, Y Dodis, C. Malinaud, and P. Puniya, \Merkle-Damgard revisited: how to construct a hash function," Advances in Cryptology, Proceedings Crypto'05, LNCS 3621, V. Shoup, Ed., Springer, Heidelberg, 2005, pp. 430{448.

[23]. N. Courtois, W. Meier, Algebraic attacks on stream ciphers with linear feedback, Advances in Cryptology, Proceedings Eurocrypt'03, LNCS 2656, E. Biham, Ed. (Springer, Heidelberg, 2003) 345{359.

[24]. J. Daemen, V. Rijmen, The Design of Rijndael. AES { The Advanced Encryption Standard, Springer, Heidelberg (2001). J.P. Degabriele, K.G. Paterson, \Attacking the IPsec standards in encryptiononly con gurations," in IEEE Symposium on Security and Privacy, IEEE, 2007, pp. 335{349.

[25]. B. den Boer, A. Bosselaers, Collisions for the compression function of MD5, Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765, T. Helleseth, Ed., Springer, Heidelberg, 1994, pp. 293{304.

[26]. T. Dierks, E. Rescorla, \The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[27]. 30. H. Dobbertin, The status of MD5 after a recent attack, CryptoBytes, Vol. 2, No. 2, Summer 1996, pp. 1{6.

[28]. 31. O. Dunkelman, N. Keller, A. Shamir, \A practical-time attack on the KASUMI cryptosystem used in GSM and 3G telephony," Advances in Cryptology, Proceedings Crypto'10, LNCS, T. Rabin, Ed., Springer, Heidelberg, 2010, in print.