Some Encryption Algorithms

by David G. Messerschmitt

Supplementary section for Understanding Networked Applications: A First Course, Morgan Kaufmann, 1999.

The encryption algorithm lies at the foundation of confidentiality, authentication, and non-repudiation. It is useful to take a closer look at representative encryption algorithms to gain a sense of how they work, as well as their vulnerabilities.

Some basic terminology of encryption algorithms is illustrated in Figure 1.. The encryption algorithm, working within the infrastructure, assumes that whatever information the application wants encrypted (for confidentiality or integrity) is represented as data. For purposes of encryption, the data is typically fragmented into fixed-size n-bit *blocks*, which are similar to packets in the network (except they are fixed length and packets may not be). (There is an alternative approach called stream ciphers not discussed in this book.)

Example: If the encryption algorithm works on blocks of $n = 64$ bits each, and a message with 610 bits is to be encrypted, then the message must be *padded* with 30 zero or randomized bits (to make it a multiple of 64) and then fragmented into 10 blocks of 64 bits each. Each of those plaintext blocks is encrypted using a block-encryption algorithm. At the recipient, the ciphertext blocks are decrypted and reassembled into the original messages. This process is similar to how a message is fragmented into packets and then reassembled for communication through the network.

An interesting point, emphasized in Figure 1., is that the encryption algorithm can assign any interpretation to a block it chooses, independent of the representation used by the application.

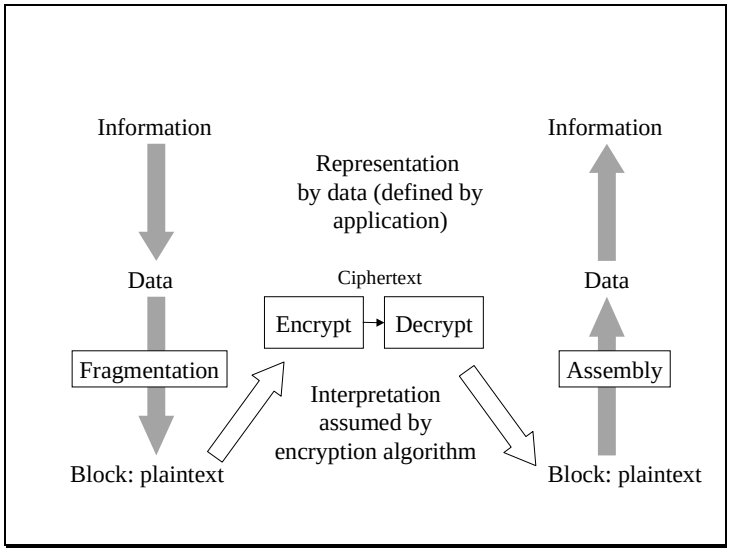

Figure 1. An encryption algorithm assumes its own interpretation of data, which may be different from the application.

This is because the decryption preserves the integrity of the plaintext blocks, so the representation it uses in its own encryption/decryption is irrelevant. For example, the RSA algorithm displayed shortly assumes each plaintext block represents an integer value (interpreting the bits in the block according to base-two arithmetic), and transforms it using arithmetic operations.

Block Substitution Table

The basic properties of a block algorithm can be appreciated by writing it in tabular form, known as a block substitution table. (This table is similar to the letter substitution algorithm in "Simplistic Encryption Algorithms" in Chapter 13, except that it substitutes for blocks of bits rather than letters.) An example of such a table for a very small block size $n = 3$ is given in Table 1. The important properties of this table, which apply to any blocksize n and to any block encryption algorithm include:

- There are 2^n rows, one for each possible plaintext block of n bits, so that every feasible plaintext block has a ciphertext correspondence.
- The encryption must be reversible, so the original plaintext can be recovered from the ciphertext. Thus, each possible ciphertext block must appear exactly once. The ciphertext column is merely a rearrangement of the plaintext column, called a *permutation*.
- Ideally there is no discernible relationship between the plaintext and ciphertext. For example, the ciphertext column could be obtained a random experiment, like a sequence of coin tosses.

Table 1 An example of a block substitution table for $n = 3$.

Plaintext	Ciphertext
000	011
001	111
010	101
011	010
100	000
101	100
110	001
111	110

What is the key corresponding to an algorithm represented by a block substitution table? The key must exactly represent the table—in fact, it is only necessary to represent the right column, since the left column follows a predictable pattern. For example, the key could be just the data in the right column,

Key = “011,111,101,010,000,100,001,110”

where the commas have been added for easy interpretation. If there is no discernible pattern to the ciphertext column, then this 24-bit key is almost the smallest key that can represent the table (but not quite, see the exercises). The number of possible 24-bit keys is $2^{24} = 16.7$ million. (It turns

out that only 16 bits are required to represent the key, and thus there are only 40,320 keys.)

The goal is for someone observing the cipertext output but not knowing the key—that is, not knowing the table—to require unreasonable effort to recover the key or the plaintext. The third property—that there is no discernible pattern between the two columns of the table—helps prevent cryptanalysis. However, it is not sufficient—to have any hope of security, the block size has to be much larger than in Table 1. The reason is that for such a small table, every possible key could be tried in a very short time on a fast computer. A more subtle problem is that such a small block size would be easily susceptible to a statistical analysis, based for example on the relative frequency of letters in the alphabet for "typical" plaintext.

For much larger (and hence more secure) block sizes, this tabular approach quickly becomes impractical. A block size of n requires a table with $n \cdot 2^n$ bits in the ciphertext column.

Example: For $n = 64$, the number of bits in the table is 10^{21}. A typical desktop computer might have a total disk space of about 10 gigabytes, or 8×10^{10} bits, which would be grossly inadequate to store this key.

The shortcomings of the tabular approach illustrate the need for an encryption *algorithm*. That is, rather than a brute-force approach of the block substitution table, a computational algorithm can achieve a permutation with desirable properties, but using a much smaller key.

Example: An example of an algorithm would be a bit-permutation. That is, each ciphertext is obtained by permuting the n bits in the corresponding plaintext in a standard way. This encryption algorithm would have as a key a representation of the permutation pattern, which would require many fewer bits to represent than the block substitution. (A 64-bit permutation, for example, can be represented by only 296 bits, which is easily stored.)

While this bit-permutation algorithm illustrates the algorithmic approach to encryption, it too is very insecure. What is needed is a more sophisticated algorithmic approach, such as the widely used DES.

Data Encryption Standard

The data encryption standard (DES) is widely used for symmetric encryption of large amounts of data. It is standardized, and thus very attractive for symmetric encryption in uncoordinated environments, often using a random session key communicated confidentially by a digital envelope.

DES is a symmetric encryption block algorithm with block size $n = 64$ bits. As just shown, the straightforward block substitution table would require an impracticably large key. DES uses an algorithm that requires a dramatically smaller key with only 56 bits—the reduction in key size from 2^{21} to 56 bits is truly dramatic—called the *Feistel algorithm* [Fei73] and is illustrated in Figure 2.. The 64-bit plaintext is split in two 32-bit blocks, and passed through sixteen stages of a *round function*. One half the ciphertext is transformed by a round function f, which also depends on a subset of the key, and is then combined with the other half using a "bit-by-bit addition", where each addition bit is specified by this table (which represents base-two arithmetic addition):

First bit	Second bit	Sum
0	0	0
0	1	1
1	0	1

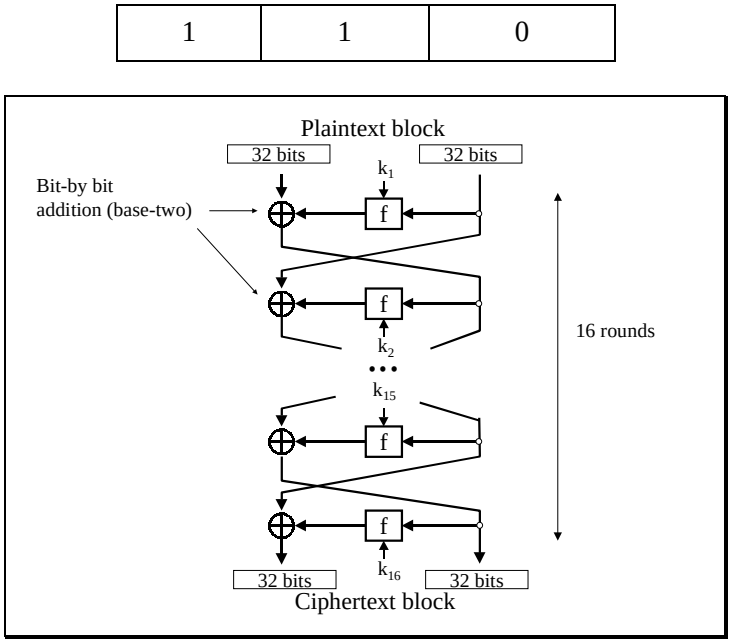

Figure 2. The sixteen rounds of the DES encryption algorithm.

The results of each stage are swapped before being applied to the following round function stage—sixteen in all.

The DES algorithm uses the two basic techniques of encryption [Sch96]:

- *Confusion*. The relationship between plaintext and ciphertext is obscured—in DES, this is achieved by a key-dependent substitution in the round function.
- *Diffusion*. The plaintext is diffused throughout the ciphertext—making every part of the ciphertext dependent on every part of the plaintext—by the swapping between stages.

Many algorithmic-based block encryption algorithms depend on some combination of confusion and diffusion.

RSA Encryption

One of the earliest and still a popular asymmetric encryption algorithm is RSA. While DES treats the data block in its more basic form—a collection of bits—RSA presumes that the plaintext block is a representation of an integer using base-two arithmetic. Thus, the algorithm assumes the plaintext P and the ciphertext C are integers represented by the data in the plaintext and ciphertext blocks. Further, the algorithm presumes these two integers fall in the range $0 \le P < n,\ 0 \le C < n$, where the so-called *modulus* n is presumed to be the product of two prime numbers (integers with no factors other than themselves and one).

Example: Suppose the plaintext is four bits, which can represent a range of eight values. This is consistent with $n = 2 \cdot 5 = 10$ if only 10 of those 16 values is allowed. In practice, the blocks are much larger than this—on the order of 1000 or 2000 bits—so that n is a huge number.

The basic operation of the algorithm, as shown in Figure 3., assumes that the integers represented by the plaintext block are themselves represented by n equally spaced points on a circle. A particular plaintext block P is represented by one point on the circle. The encryption algorithm rotates

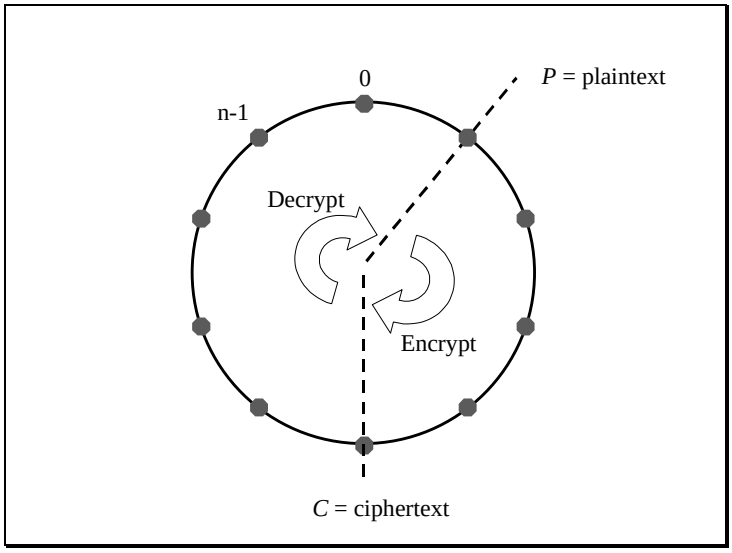

Figure 3. The RSA algorithm works with integers represented as points on a circle.

to a different point C, based on the value of both P and the key. The decryption algorithm does another rotation from C, based on a different by coordinated key, landing back at the original P.

The actual encryption algorithm is remarkably simple,

$$C = P^s \bmod n, \qquad \text{EQ 1}$$

where the key consists of (n, s). It simply multiplies P by itself $s - 1$ times, with the complication that the result is forced to fall in the range between 0 and $n - 1$ by interpreting it on the circle (this is what the modulo, or mod, function means).

Example: If $n = 10$, then the result of exponentiation must fall in the range zero to nine. For example, $3^8 = 6561$ which carries around the circle 656 times with one left over, so the result is equal to one.

The decryption algorithm is identical, except that a different exponent t and key (n, t) is used. All the sophistication of the algorithm lies in the determination of s and t such that encryption followed by decryption recovers the plaintext. (If you are interested and mathematically inclined, see the book homepage for a derivation of the RSA algorithm.)

The asymmetric algorithm security depends on the computational impracticality of inferring the secret key from the public key. For RSA, this depends in turn on the difficulty of factoring n, which is known to be the product of two prime numbers. Factorization can be made computationally difficult by choosing n to be very large (representation of n by 1024 bits is typical).

Discussion

D1 Encryption technology is based in part of centuries of research on number theory, research not motivated by any practical application. Use this observation to discuss the importance of fundamental research in mathematics and other fields.

Exercises

E1. Consider the possibility of using RSA as a message digest. You can do this by finding a key pair,

throwing away the secret key, and encrypting the message using RSA and the public key.

a. List again the properties of a message digest.

a. Are there any properties of a message digest that are not satisfied by using RSA in this way?

b. How can the algorithm be modified or enhanced to turn it into a message digest?

E2. Describe where you might place a firewall and the policies you might implement in the firewall to prevent each of the following bad things from happening:

a. Preventing an intruder or unauthorized employee from gaining access to your payroll database on a mainframe

b. Preventing an employee sending undetected harassing electronic mail messages to someone in another department

c. Preventing employees authorized to examine but not change payroll records from actually being able to change them (without impacting the ability of other authorized employees to change the records)

d. Preventing employees from sending large documents or program code as email attachments while allowing them to send short-length messages

E3. For the block substitution table of Table 1, generalized to an n-bit plaintext:

a. If the bits in the right column are used as the key, what is the required number of bits in the key?

b. Considering that the ciphertext rows are just a permutation of the plaintext rows, what is the absolute minimum number of bits in the key?

c. Approximate b. using Stirling's formula, and show that a. and b. are roughly equal for large n.

d. Using the results of a. and c., show that the number of bits required for the key is about 10^{21} for $n = 64$.

E4. Consider the set of all possible words with eight characters. How many such words are there? Assuming that all possible pairs of successive words are possible, how many pairs of eight-character words are there? Discuss the implications of this for the career prospects of novelists.

E5. The human genome (genetic definition of an individual) is estimated to consist of three billion base pairs, where each base pair has four possibilities. It is also estimated that only about 20% of these base pairs have any meaning; with the remainder unused. Assuming that all possible sequences of base pairs in the genome are possible, how many possible viable human beings are there? Compare this to the population of the world, which is roughly six billion people.

E6. Design an interface specification for each of the following object classes, where each class is allowed to inherit other classes, and objects of these classes can be passed as parameters or returns. Be sure to benefit from inheritance and polymorphism.

a. `class Block_of_data`

b. `class Key`

c. `class Encryption`

d. `class Decryption`

E7. In anticipation of constructing a lot of server objects that need to be authenticated by their clients, you define a superclass `Authenticated_server`. All server objects that need to be authenticated by client objects will inherit this superclass. As part of this authentication, a random `session_key` is generated by the server object and is available to the client object for use in symmetric encryption of

subsequent messages. Define the interface to this class for:

a. Randomized challenge/response authentication
b. Certificate authentication