# Modern Cryptology: an Introduction

Bart Preneel*

September 2, 1998

**Abstract**

This paper provides an overview of the state of the art in the design of cryptographic algorithms. It reviews the different type of algorithms for encryption and authentication. The principles are explained of stream ciphers, block ciphers, hash functions, public-key encryption algorithms, and digital signature schemes. Subsequently the design and evaluation procedures for cryptographic algorithms are discussed.

*F.W.O. postdoctoral researcher, sponsored by the Fund for Scientific Research – Flanders (Belgium).

# Contents


**1 Introduction** **3**

**2 Encryption for secrecy protection** **3**
2.1 Conventional encryption . . . . . 5
2.1.1 Additive stream ciphers . . . . . 5
2.1.2 Self-synchronizing stream ciphers . . . . . 7
2.1.3 Block ciphers . . . . . 7
2.2 Security of conventional algorithms . . . . . 10
2.3 Public-key encryption . . . . . 11

**3 Hashing and signatures for authentication** **13**
3.1 Symmetric authentication . . . . . 14
3.1.1 MDCs . . . . . 14
3.1.2 MACs . . . . . 15
3.2 Digital signatures . . . . . 17

**4 Analysis and design of conventional cryptographic algorithms** **18**
4.1 Three approaches in cryptography . . . . . 18
4.2 Life cycle of a cryptographic algorithm . . . . . 19
4.3 Brute force attacks versus shortcut attacks . . . . . 20
4.3.1 Brute force attacks . . . . . 20
4.3.2 Shortcut attacks . . . . . 21
4.4 Public versus secret algorithms . . . . . 21
4.5 Insecure versus secure algorithms . . . . . 21

**5 Concluding remarks** **22**

**A The Data Encryption Standard (DES)** **25**

# 1 Introduction

The expansion of worldwide communications and the increased digitalization of our society can make information more vulnerable to misuse. This misuse can take many forms: eavesdropping of sensitive information (for example, company secrets), unauthorized modification of information (changing money transfers between banks, or introducing viruses into computer software), stealing information (images or audiovisual recordings), use of electronic services without paying for them, repudiating electronic orders, bringing down computer systems or networks, etc. The risk for misuse has increased considerably, as potential attackers can operate from all over the globe. Moreover, if someone gains access to an electronic information system, the scale and impact of the abuse can be much larger than in a paper-based system.

These risks require adequate security measures to protect electronic information systems. It is clear that in an electronic world physical security or personnel security by itself will not be sufficient. An important component of every secure system is formed by advanced digital cryptographic techniques.

In this paper we discuss the principles underlying the design of cryptographic algorithms; we distinguish between confidentiality protection (the protection against passive eavesdroppers) and authentication (the protection against active eavesdroppers, who try to modify information). Then we review the different issues that arise when selecting, designing, and evaluating a cryptographic algorithm. Finally we present some concluding remarks.

# 2 Encryption for secrecy protection

The use of cryptography for protection the secrecy of information is as old as writing itself [16]. The basic idea consists of applying a 'complicated' transformation to the information to be protected. When the sender (usually called Alice in cryptography) wants to send a message to the recipient (Bob), she will apply to the *plaintext* $P$ the mathematical transformation $E()$. This transformation $E()$ is called the encryption algorithm; the result of this transformation is called the *ciphertext* or $C = E(P)$. Bob will decrypt $C$ by applying the inverse transformation $D = E^{-1}$; in this way he recovers $P$ or $P = D(C)$. For a secure algorithm $E$, the ciphertext $C$ does not make sense to outsiders: Eve, who is tapping the connection, can obtain $C$, but she cannot obtain (partial information on) the corresponding plaintext $P$.

This approach only works when Bob can keep the transformation $D$ secret. While this is ok for a person-to-person exchange, it is not feasible for large scale use. Bob needs a software or hardware implementation of $D$: either he has to program it himself, or he has to trust someone to write the program for him. Moreover, he will need a different transformation (and program) for each correspondent, which is not very practical. Bob and Alice always have to face the risk that somehow Eve will obtain $D$ (or $E$), for example by bribing the author of the software or their system manager, or by breaking into their computer system.

This problem can be solved by introducing into the encryption algorithm $E()$ a secret parameter, the key $K$. Typically such a key is a binary string of 40 to a few thousand bits. A corresponding key $K^*$ is used for the decryption algorithm $D$. One has thus $C = E_K(P)$ and $P = D_{K^*}(C)$ (see also Figure 1, which assumes that $K^* = K$). The transformation has to depend strongly (and in a very complicated way) on the keys: if one uses a wrong key $K^{*\prime} \neq K^*$, one does not obtain the plaintext $P$ but a 'random' plaintext $P'$. Now it is possible to publish the encryption algorithm $E()$ and the decryption algorithm $D()$; the security of

the system relies only on the secrecy of two short keys. This implies that $E()$ and $D()$ can be evaluated publicly and distributed on a commercial basis. One can think of the analogy with a mechanical lock: everyone knows how such a lock works, but to open a particular lock, one needs to know the key or the secret combination. The assumption that the algorithm is known to the opponent is known in cryptography as "Kerckhoffs's principle"; Kerckhoffs was a 19th century Dutch cryptographer who was the first to write down this approach.

A simple example of an encryption algorithm is the so-called 'Caesar cipher,' after the Roman emperor who used it. The plaintext is encrypted letter by letter; the ciphertext is obtained by shifting the letters over a fixed number of positions in the alphabet. The secret key indicates the number of positions. It is claimed that Caesar always used the value of three, such that "AN EXAMPLE" would be encrypted to "DQ HADPSOH". Another example is the name of the computer "HAL" from S. Kubrick's "A Space Odyssey (2001)", which was obtained by replacing the letters of "IBM" by their predecessor in the alphabet. This corresponds to a shift over 25 positions. It is clear that such a system is currently completely insecure.

A problem which has not yet been addressed is how Alice and Bob exchange the secret key. The easy answer is that cryptography does not solve this problem; cryptography only makes problems easier. In this case the secrecy of a (large) plaintext has been reduced to that of two *short* keys, which can be exchanged on beforehand. The problem of exchanging keys is studied in more detail in an area of cryptography that is called 'key management'. We will not discuss it in further detail here.

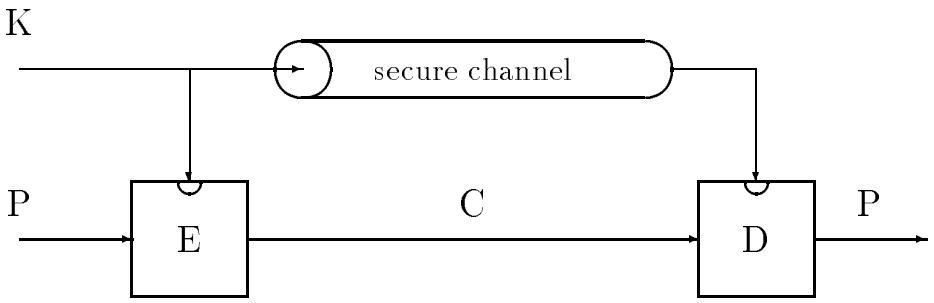


Figure 1: Model for conventional or symmetric encryption

The branch of science which studies the encryption of information is called *cryptography*. A related branch tries to 'break' encryption algorithms, by recovering the plaintext without knowing the key or by deriving the key from the ciphertext and parts of the plaintext; it is called *cryptanalysis.* The term *cryptology* covers both aspects. For more extensive introductions to cryptography, the reader is referred to [3, 17, 21, 22, 28, 29].

Thus far we have assumed that the key for decryption $K^*$ is equal to the encryption key $K$, or that it is easy to derive $K^*$ from $K$. This type of algorithms are called *conventional* or *symmetric* ciphers. In *public-key* or *asymmetric* ciphers, $K^*$ and $K$ are always different; moreover, it should be difficult to compute $K^*$ from $K$. This has the advantage that one can make $K$ public, which has important implications to the key management problem. The remainder of this section mainly discusses conventional algorithms; public-key algorithms are

treated briefly as well.

## 2.1 Conventional encryption

When designing or selecting an encryption algorithm, several criteria are taken into account:

**security:** related to the value of the information to be protected;

**performance:** in hardware or software;

**cost:** of design or licensing, implementation, and key management;

**availability:** this aspect is often related to commercial constraints or national security constraints (certain countries restrict the export of strong encryption algorithms);

**standardization:** this allows for maximal reuse of equipment and hopefully provides some security guarantees;

**error propagation and synchronization:** these aspects are very important for encryption in lower layers of a communications protocol stack, where channels are unreliable.

Encryption algorithms can be classified into three types based on their error propagation and synchronization properties:

- additive stream ciphers;
- self-synchronizing stream ciphers;
- block ciphers.

This classification will be followed in the rest of this section. One can also classify ciphers based on the plaintext alphabet. For manual encryption one often uses the conventional alphabet (A-Z) (as in the Caesar cipher), but for electronic applications a binary alphabet is more appropriate; plaintext and ciphertext are then strings of ones and zeroes. This will be the case for the rest of this paper.

### 2.1.1 Additive stream ciphers

Additive stream ciphers are ciphers for which the encryption consists of a modulo 2 addition (exclusive or, exor) of a key stream to the plaintext (see Figure 2). The plaintext and ciphertext are divided into words of $m$ bits ($m$ is typically 1 or 8), and the $i$th word of the plaintext, ciphertext, and key stream is denoted with $p_i$, $c_i$, and $k_i$, respectively. The encryption operation can then be written as:

$$c_i = p_i \oplus k_i \,.$$

Here $\oplus$ denotes addition modulo 2 or $0 \oplus 0 = 0$, $0 \oplus 1 = 1 \oplus 0 = 1$, and $1 \oplus 1 = 0$. The decryption operation is identical to the encryption (in mathematical terms, the cipher is an involution): indeed, $p_i = c_i \oplus k_i = (p_i \oplus k_i) \oplus k_i = p_i \oplus (k_i \oplus k_i) = p_i \oplus 0 = p_i$. It is clear that the $m$-bit key stream word $k_i$ cannot be a constant (in that case a cryptanalyst can compute the key stream word from a single ciphertext word and the corresponding plaintext word; also repetitions in the plaintext would be visible in the ciphertext). One can show that for a strong cipher the sequence of $k_i$ has to consist of randomly looking strings (see also §2.2). It is necessary but not sufficient that the values are uniformly distributed; another condition is that there should be no correlations between (part of) successive words (note

that cryptanalytic attacks exist which exploit correlations of less than 1 in 1 million; finding such correlations with 'blind' statistical tests is not feasible).

In practice one computes the words $k_i$ with a finite state machine $F_K$ with state $s_i$ at time $i$. The initial state $s_0$, the next state function $f_2()$, and the output function $f_3()$ can all depend on the secret key $K$ (see Figure 3):

$$s_0 = f_1(K, IV) \qquad s_{i+1} = f_2(K, s_i) \qquad k_i = f_3(K, s_i) \; .$$

Such a machine stretches a short secret key $K$ into a much longer key stream sequence $k_i$; in cryptography this is called a *pseudo-random* string generator. The sequence $k_i$ is eventually periodic. One important (but again not sufficient) design criterion for the finite state machine is that the period has to be sufficient long ($2^{64}$ is a typical lower bound).

Often the initial state will also depend on an additional parameter, the initial value $IV$; by varying the $IV$ one can ensure that the initial state $s_0$ and hence the sequence $k_i$ varies, even if the key $K$ is unchanged. Sender and receiver have to agree somehow on the value of $IV$ (for example, it can be a serial number, or it can be sent as the first part of the ciphertext).

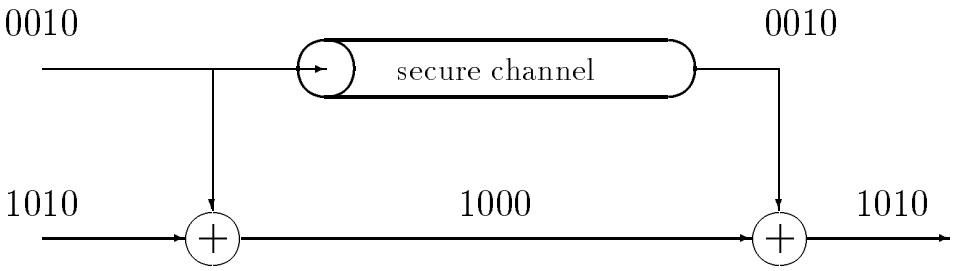


Figure 2: An additive stream cipher

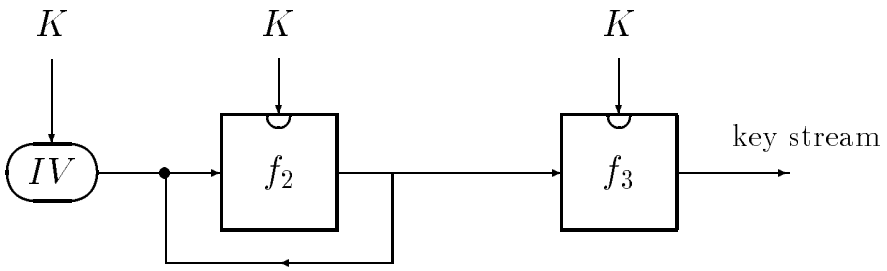


Figure 3: Internal details of an additive stream cipher; $IV$ is the Initial Value, $f_2$ is the next state function, and $f_3$ is the output function

An important advantage of additive stream ciphers is that they do not cause any error propagation: when a single bit of the ciphertext is modified (by a transmission error), the

corresponding plaintext bit will be in error. This disadvantage is that active attacks are easy: an opponent can easily flip one bit of the plaintext, without disturbing the rest of the plaintext (cf. §3). An important practical problem is the synchronization issue: if one or more bits are lost during transmission, the wrong key word is added to the ciphertext and the recovered plaintext becomes useless (it will be random). This can only be resolved by re-synchronizing with a new $IV$ at predetermined instances or if errors are detected; what is often overlooked is that this can have major security implications.

### 2.1.2 Self-synchronizing stream ciphers

Self-synchronizing stream ciphers avoid the re-synchronization problem. In these ciphers the key word $k_i$ depends on the $M$ previous ciphertext words, which are stored in the internal memory of the encryption algorithm. The usual form of this memory is a shift register, which then forms the input of the encryption function $f_K()$:

$$c_i = p_i \oplus k_i \qquad \text{with} \qquad k_i = f_K(c_{i-M}, \ldots c_{i-1}) \; .$$

Again the decryption operation is equal to the encryption operation. A single error in the ciphertext is copied to the corresponding plaintext bit; this error will be present in the shift register for the next $M$ steps, which implies that in total $M + 1$ words of the plaintext will contain errors. This corresponds to an error propagation with a factor of $M + 1$. Note that one cannot choose $M$ too small, since this has security implications (cf. §4.3.1); a good compromise for $m = 1$ is $M = 64 \ldots 128$.

The main advantage of these ciphers is their self-synchronizing property: if $M$ successive ciphertext words are received correctly, the next ciphertext word will be deciphered correctly, no matter what happened before (the only restriction is that the word boundaries have to be preserved). If the plaintext is enciphered bit by bit (or $m = 1$), one can recover automatically from loosing an arbitrary number of ciphertext bits. This has also an advantage for starting up the communication session: it is sufficient to send an initial value of $M$ random ciphertext words; if these are received correctly, sender and receiver will be synchronized.

These properties do not come for free: an opponent who can choose ciphertext and obtain the corresponding plaintext can influence the decryption function. This will make it easier to recover the secret key. To preclude such an attack, the encryption function of a self-synchronizing stream cipher needs stronger cryptographic properties than that of an additive stream cipher.

### 2.1.3 Block ciphers

Block ciphers take a different approach to encryption: the plaintext is divided into larger words of $n$ bits, which are denoted *blocks*. Every block is enciphered in the same way; the encryption operation does *not* depend on the location in the ciphertext as is the case for additive stream ciphers, or on the previous plaintexts, as is the case for self-synchronizing stream ciphers. One can envisage the following attack on a block cipher: the cryptanalyst collects ciphertext blocks and their corresponding plaintext blocks (this is possible as part of the plaintext is often predictable); this is used to build a large table (this attack is called a tabulation attack). With such a table, one can deduce information on other plaintexts encrypted under the same key. In order to preclude this attack, two conditions have to be satisfied:

- the value of $n$ has to be quite large; typical values are 64 or 128;
- the plaintext should not contain any patterns, as these will be leaked to the ciphertext.

Note that the latter condition is a condition on the plaintext; however, there is an easy way to satisfy this condition for all plaintexts: one uses the block cipher in a special way, which is called a *mode of operation.* Theses modes have been standardized [6, 14].

The simplest mode is the ECB (Electronic CodeBook) mode. The plaintext is divided into $n$-bit blocks, and is encrypted block by block; the decryption also operates on individual blocks:

$$c_i = E_K(p_i) \qquad \text{and} \qquad p_i = D_K(c_i)\,.$$

Errors in the ciphertext do not propagate beyond the block boundaries (as long as these can be recovered). However, the ECB mode does not hide patterns (such as repetitions) in the plaintext), as these will be copied to the ciphertext. Therefore this mode can only be used in very special cases, where the plaintext is already random, such as the encryption of cryptographic keys.

The standard mode of operation of a block cipher is the CBC (Cipher Block Chaining) mode. In this mode the different blocks are coupled by adding modulo 2 to a plaintext block the previous ciphertext block before the encryption operation:

$$c_i = E_K(p_i \oplus c_{i-1}) \qquad \text{and} \qquad p_i = D_K(c_i) \oplus c_{i-1}\,.$$

Note that this 'randomizes' the plaintext, and hides patterns. To enable the encryption of the first plaintext block ($i = 1$), one defines $c_0$ as the initial value $IV$. By varying this value, one can ensure that the same plaintext is encrypted into a different ciphertext under the same key. As for additive stream ciphers, sender and receiver have to agree on the value of $IV$. The CBC mode has a limited error propagation: errors in the $i$th ciphertext block will garble the $i$th plaintext block completely, and will be copied into the $i+1$th plaintext block. The CBC mode allows for random access on decryption: if necessary, one can decrypt only a small part of the ciphertext.

Two other modes allow for the use of a block cipher as an additive stream cipher and as a self-synchronizing stream cipher respectively. These modes use only the encryption operation, and have the same properties as the corresponding stream ciphers. The first stream mode is the OFB (Output FeedBack) mode. The key word is computed as follows (see also Figure 5):

$$k_0 = IV \qquad k_i = E_K(k_{i-1})\,.$$

Sometimes one uses only $m$ ($< n$) bits of the key word; this increases the security level at the cost of a lower speed.

The CFB (Cipher FeedBack) mode is a self-synchronizing stream cipher, where the function $f_K()$ consists of the block $E_K()$ (see Figure 6). The length of a plaintext word determined the number $m$ of bits used from the output of $E_K()$. The value of $M$ is equal to $n/m$. In practice one often uses $m = 1$ and $m = 8$. The CFB mode is $M$ times slower than the CBC mode; for small values of $m$ the CFB mode is very slow.

This section has illustrated that a block cipher forms a very flexible building block. The price paid for this is that it has to satisfy very strong security requirements. One very famous block cipher is the Data Encryption Standard (or DES). It is described in Annex A.

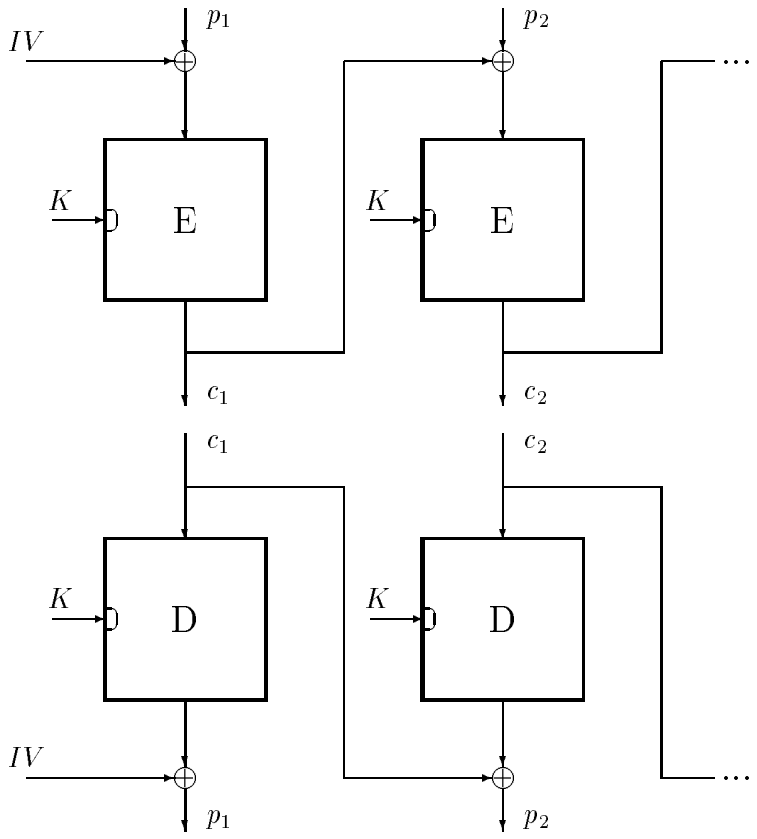


Figure 4: The CBC mode of a block cipher

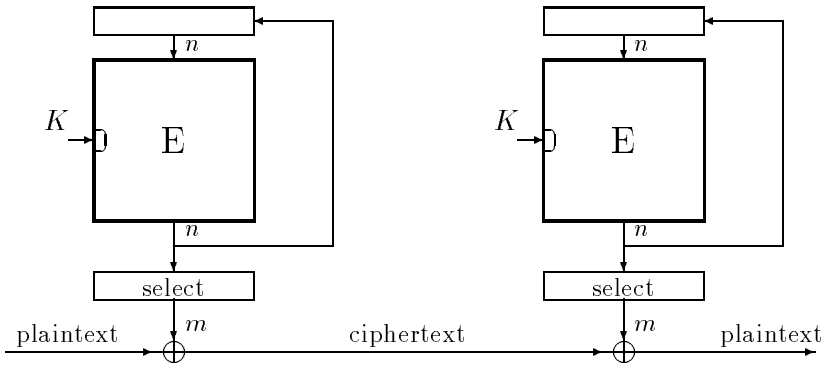


Figure 5: The $m$-bit OFB mode of an $n$-bit block cipher

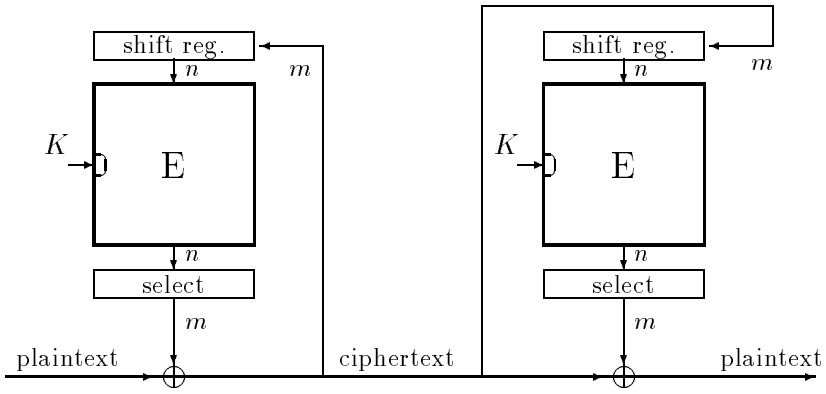


Figure 6: The $m$-bit CFB mode of an $n$-bit block cipher

## 2.2 Security of conventional algorithms

An essential aspect in the choice of an encryption algorithm is the security level. In 1926 G.S. Vernam has published a simple encryption algorithm for telegraphic messages [30]. The cipher is an additive stream cipher, where the key stream consists of a completely random sequence, generated by a binary symmetric source (all bits are uniformly and identically distributed). In 1949 C. Shannon, the father of information theory, was able to prove mathematically that this scheme offers perfect security, i.e., from observing the ciphertext, the opponent cannot obtain any information on the plaintext, no matter how much computing power he has [27]. The main disadvantage of this scheme is that the secret key is exactly as long as the message (one should never reuse a key stream); C. Shannon also showed that this the best one can do if one wants perfect security. In spite of the long key, the Vernam algorithm is still used by diplomats and spies; it has been used for the 'red telephone' between Washington and Moscow. Spies used to carry key pads with random characters (it is easy to generalize the scheme to arbitrary alphabets). The security of the scheme relies on the fact that every page of the pad is used only once, which explains the name "one-time pad".

In most commercial applications one cannot afford to distribute keys which are as long as the plaintext. Therefore one uses encryption algorithms which do not offer perfect security; this implies that it is *in principle* possible to recover the plaintext and/or the secret key from the ciphertext, in the sense that one has sufficient information to do this. This does not mean that it is also possible *in practice.* For example, additive stream ciphers try to mimic the approach of the Vernam scheme by replacing the random key stream sequence by a pseudo-random sequence.

In order to determine the security of an encryption algorithm, one has to make assumptions on the knowledge available to the cryptanalyst.

- In the most restrictive case, the cryptanalyst only knows the ciphertext (a *ciphertext-only* attack). The cryptanalyst is only able to break the cipher if the plaintext has some redundancy (for example if he knows it consists of an English text in ASCII). If the plaintext is randomized perfectly (by the cipher), the cipher will resist a ciphertext only attack; such a cipher is called ideally secure. Note that perfect randomization is not very practical, as it requires a perfect model of the source.

- A more realistic assumption is that the cryptanalyst knows a number of ciphertext and corresponding plaintexts (for example, he knows the beginning of a letter, or has some information on the telecommunication protocol). This is called a *known plaintext* attack.

- The most powerful setting allows a cryptanalyst to carry out a *chosen plaintext (ciphertext)* attack: this means that he can obtain ciphertexts (plaintexts) corresponding to plaintexts (ciphertexts) of this choice. While this attack is not very realistic, it provides a good idea of the strength of the algorithm. Note that for an additive stream cipher, a chosen plaintext attack and a known plaintext attack are equivalent, as the opponent has no control over the key.

In §4 we will revisit the problem of how to evaluate the security of a conventional cryptographic algorithm.

### 2.3 Public-key encryption

The main problem which is left unsolved by conventional cryptography is the key distribution problem. Especially in a large network it is not feasible to distribute keys between all user pairs (in a network with $t$ users there are $t(t-1)/2$ such pairs). An alternative is to store all keys in a central location, but this may then become a single point of failure. Public-key cryptography offers a much more elegant solution to this problem.

The concept of public-key cryptography has been invented by in 1976, independently by W. Diffie and M. Hellman [4] and by R. Merkle [20]. The basic idea behind public-key cryptography is the concept of *trapdoor one-way functions*. A *one-way function* is a function which is easy to compute, but which is hard to invert. For example, in a conventional block cipher, the ciphertext has to be a one-way function of the plaintext and the key: it is easy to compute the ciphertext from the plaintext and the key, but given the plaintext and the ciphertext it should be hard to recover the key (otherwise the block cipher would not be secure). Similarly one can show that the existence of additive stream ciphers (pseudo-random string generators) implies the existence of one-way functions. Trapdoor one-way functions are one-way function with an additional property: given some additional information (the trapdoor), it becomes possible to invert the one-way function.

With such functions Bob can send a secret message to Alice without the need for prior arrangement of a secret key. Alice chooses a trapdoor one-way function with public parameter $P_A$ (Alice's public key) and with secret parameter $S_A$ (Alice's secret key). Alice makes her public key widely available (she can put it on her home page, but it can also be included in special directories). Anyone who wants to send some confidential information to Alice, computes the ciphertext as the image of the plaintext under the trapdoor one-way function using the parameter $P_A$. Upon receipt of this ciphertext, Alice recovers the plaintext by using her trapdoor information $S_A$ (see Figure 7). An attacker, who does not know $S_A$, sees only the image of the plaintext under a one-way function, and will not be able to recover the plaintext. This assumes that it is infeasible to compute $S_A$ from $P_A$. Note that if one wants to send a message to Alice, one has to know Alice's public key $P_A$, and one has to be sure that this key really belongs to Alice (and not to Eve), since it is only the owner of the corresponding secret key who will be able to decrypt the ciphertext. Public keys do not need a secure channel for their distribution, but they do need an authentic channel. As the keys

for encryption and decryption are different, and Alice and Bob have different information, public-key algorithms are also known as asymmetric algorithms.

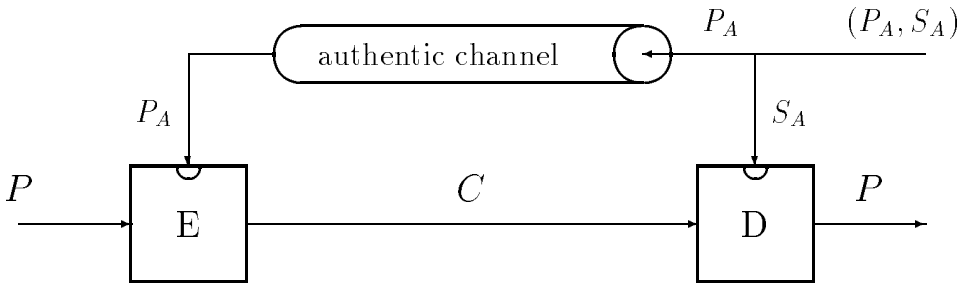


Figure 7: Model for public-key or asymmetric encryption

The principles behind public-key cryptography can be clarified with two analogies. The first one is a mechanical one: public-key encryption is like a mailbox, in which everyone can deposit a letter, but only the owner of the mailbox can retrieve the letter as this requires the key of the mailbox. In this case it is important that one is sure of the name of the person on the mailbox. A second analogy is based on dictionaries; it emphasizes the equivalence between public and secret keys and the difficulty of computing the secret key from the public key. The plaintext is written in English, and the ciphertext is the translation into Latin. The encryption requires the public key, namely a dictionary English-Latin. The decryption uses the secret key, a dictionary Latin-English. Even if one has the public key (an English-Latin dictionary), decrypting a Latin ciphertext is quite hard (if one does not know Latin). Moreover, while it is in principle feasible to derive the secret key from the public key, this requires quite some effort.

The conditions which a public-key encryption algorithm has to satisfy are:

- the generation of a key pair $(P_A, S_A)$ has to be easy;
- encryption and decryption have to be easy operations;
- it should be hard to compute the public key $P_A$ from the corresponding secret key $S_A$;
- $S_A(P_A(P)) = P$.

Designing a secure public-key encryption algorithm is apparently a very difficult problem. From the large number of proposals, only a few have survived (for example, almost all knapsack-based systems have been broken). The most popular algorithm is the RSA algorithm [25], which was named after its inventors (R.L. Rivest, A. Shamir, and L. Adleman). The security of RSA is based on the fact that it is relatively simple to find two large prime numbers (in 1997 large means 115 decimal digits or more) and to multiply these, while factoring their product (of 230 decimal digits) is not feasible with the current algorithms and computers.

**key generation:** Find 2 prime numbers $p$ and $q$ with at least 115 digits and compute their product, the modulus $n = p \cdot q$. Compute the Carmichael function $\lambda(n)$, the least common multiple of $p-1$ and $q-1$. Choose an encryption exponent $e$ (at least 32 to 64 bits long), which is relatively prime to $\lambda(n)$ and compute the decryption exponent as

$d = e^{-1} \bmod \lambda(n)$ (with Euclid's algorithm). The public key consists of the pair $(e, n)$, and the secret key consists of the decryption exponent $d$ or the pair $(p, q)$;

**encryption:** represent the plaintext as an integer in the interval $[0, n-1]$ and compute the ciphertext as $C = P^e \bmod n$;

**decryption:** $P = C^d \bmod n$.

Without explaining the mathematical background of the algorithm, one can observe that decryption requires the extraction of modular $e$th roots; no algorithm is known for this problem which does not use the prime factors of $n$; finding the decryption exponent requires knowledge of $\lambda(n)$ and hence of the factors of $n$. On the other hand, this knowledge is not required for the encryption operation. For the practical use of RSA, one has to take into account many technical details: for example, $p$ and $q$ have to be 'strong' primes [10], and the plaintext $P$ has to be padded to avoid trivial attacks (e.g., the extraction of natural $e$th roots when $P^e < n$).

The more complex properties of public-key cryptography seem to require some 'high level' mathematical structure; most public-key algorithms are based on number theoretic problems (such as factoring and discrete logarithm in certain groups). One exception is a public-key system based on a cellular automaton [11]. While these number theoretic problems are believed to be difficult, it should be noted that the invention of public-key cryptography has led to significant progress in algorithms to solve these problems: the factorization record in 1975 was 39 decimal digits; in 1985 this was increased to 65 digits, and in 1994 a 130-digit modulus was factored. This evolution is due to a combination of more sophisticated factoring algorithms with progress in hardware and parallel processing. The cryptographer should take this into account by selecting sufficiently large keys for public-key algorithms.

The main advantage of public-key algorithms is the simplified key management. The main disadvantages are the larger keys (typically 64 to 256 bytes) and the slow performance: both in software and hardware public-key encryption algorithms are two to three orders of magnitude slower than conventional algorithms. For example, a 1024-bit exponentiation requires about 0.3 seconds on a 90 MHz Pentium, which corresponds to 3.4 kbit/s. An optimization of the RSA decryption uses the Chinese remainder theorem: this can increase the speed to 13.5 kbit/s. For RSA encryption, a small exponent $e$ (for example $e = 2^{32} + 1$) results in an encryption speed of about 100 kbit/s on the same machine). Because of the large difference in performance and the larger block length (which influences error propagation), one always employs *hybrid* systems: the public-key encryption scheme is used to distribute a secret key, which is then used in a fast conventional algorithm.

# 3 Hashing and signatures for authentication

Information authentication includes two main aspects:

- *data origin authentication*, or who has originated the information;
- *data integrity*, or has the information been modified.

Other aspects which can be important are the timeliness of the information, the sequence of messages, and the destination of information. These aspects can be accounted for by using sequence numbers and time stamps in the messages and by including addressing information in the data. In data communications, the implicit authentication created by recognition of the

handwriting, signature, or voice disappears. The reason is that information becomes much more vulnerable to falsification as the physical coupling between information and its bearer is lost.

Until recently it was widely believed that encryption of information (with a conventional algorithm) was sufficient for protecting its authenticity. The reasoning was that if a certain ciphertext resulted after decryption in a *meaningful* plaintext, it had to be created by someone who knows the key, and therefore it must be authentic. Two counterexample are sufficient to refute this claim: if a block cipher is used in ECB mode, an attacker can always reorder the blocks. And it was already pointed out that for any additive stream cipher (including the Vernam scheme), an opponent can always modify any plaintext bit (without knowing whether a 0 has been changed to a 1 or vice versa). The concept 'meaningful' information implicitly assumes that the information contains redundancy, which allows to distinguish genuine information from an arbitrary plaintext. However, one can envisage applications where the plaintext contains very little or no redundancy. The separation between secrecy and authentication has also been clarified by public-key cryptography: anyone who knows Alice's public key can send her a confidential message, and therefore Alice has no idea who has actually sent this message.

Two different levels of information authentication can be distinguished. If two parties trust each other and want to protect themselves against malicious outsiders, the term 'conventional message authentication' is used. In this setting, both parties are at equal footing (for example, they share the same secret key). If however a dispute arises between them, a third party (such as a judge) will not be able to resolve it (for example a judge cannot tell whether a message has been created by Alice or by Bob). If protection between two mutually distrustful parties is required (which is often the case in commercial relationships), an electronic equivalent of a manual signature is needed. In cryptographic terms this is called a digital signature.

## 3.1 Symmetric authentication

The underlying idea is similar to that for encryption, where the secrecy of a large amount of information is replaced by the secrecy of a short key. In the case of authentication, one replaces the authenticity of the information by the protection of a short string, which is a unique 'fingerprint' of the information. Such a 'fingerprint' is computed as a hash result. This can also be interpreted as adding a special form of redundancy to the information. This process consists of two components. First one compresses the information to a string of fixed length, with a (cryptographic) hash function. Then the resulting string (the hash result) is protected as follows:

- either the hash result is communicated over an authentic channel. For example, the hash result can be read over the phone or sent through conventional mail. It is then sufficient to use a hash function without a secret parameter, which is called a Manipulation Detection Code or MDC;

- or the hash function uses a secret parameter (the key); it is then called a Message Authentication Code or MAC.

### 3.1.1 MDCs

If an additional (authentic) channel is available, MDCs can provide authenticity without requiring secret keys. Moreover an MDC is a flexible primitive, which can be used for a variety

of other cryptographic applications. An MDC has to satisfy the following conditions:

- it should be hard to find an input with a given hash result (preimage resistance);
- it should be hard to find a second input with the same hash result as a given input (2nd preimage resistance);
- it should be hard to find two different inputs with the same hash result (collision resistance).

An MDC satisfying these three conditions is called a *collision resistant* hash function. For a strong hash function with an $n$-bit result, solving one of the first two problems requires about $2^n$ evaluations of the hash function. This implies that $n = 64 \ldots 80$ is sufficient. However, finding collisions is much easier: one will find with probability close to 1 a collision in a set of hash results corresponding to $2^{n/2}$ inputs. This implies that collision resistant hash functions need a hash result of 128 to 160 bits. Not all applications need collision resistant hash functions; sometimes (2nd) preimage resistance is sufficient.

> This last property is also known as the birthday paradox: within a group of 24 persons the probability that there are two persons with the same birthday is about 50%. The reason is that a group of this size contains 276 different pairs of persons, which is a large fraction of the 365 days in a year. Note that the birthday paradox plays an essential role in the security of many cryptographic primitives (cf. §4.3.1).

The details of how to design collision resistant hash functions are beyond the scope of this paper. As these are the only unkeyed cryptographic primitives, they require design criteria which are somewhat different from that of other cryptographic algorithms. Note that it is possible to construct hash functions based on block ciphers, which illustrates that block ciphers form a powerful tool.

### 3.1.2 MACs

MACs have been used for more than twenty years in electronic transactions in the banking environment. They require the exchange of a secret key between the communicating parties. The MAC corresponding to a message is a complex function of every bit of the message and every bit of the key; it should be infeasible to derive the key from observing a number of text/MAC pairs, or to compute or predict a MAC without knowing the secret key.

A MAC is used as follows (see also Figure 8): Alice computes for her message $P$ the value $\mathrm{MAC}_K(P)$ and appends this MAC to the message (here MAC denotes both the function and its result). Bob recomputes the value of $\mathrm{MAC}_K(P)$ based on the received message $P$, and verifies whether it matches the received MAC. If so, he accepts the message as authentic, i.e., as a genuine message from Alice. Eve, the active eavesdropper, can modify the message $P$ to $P'$, but she is not able to compute the corresponding MAC value $\mathrm{MAC}(P')$, as she is not privy to the secret key $K$. For a secure MAC, the best Eve can do is guessing the MAC. In that case, Bob can detect the modification with high probability: for an $n$-bit MAC Eve's probability of success is only $1/2^n$. The value of $n$ lies typically between 32 and 64. Note that if encryption and authentication are combined, the key for encryption and authentication need to be different. Moreover, the preferred option is to compute the MAC on the plaintext.

A popular way to compute a MAC is to encrypt the message with a block cipher using the CBC mode (yet another use of a block cipher), and to keep only part of the bits of the last

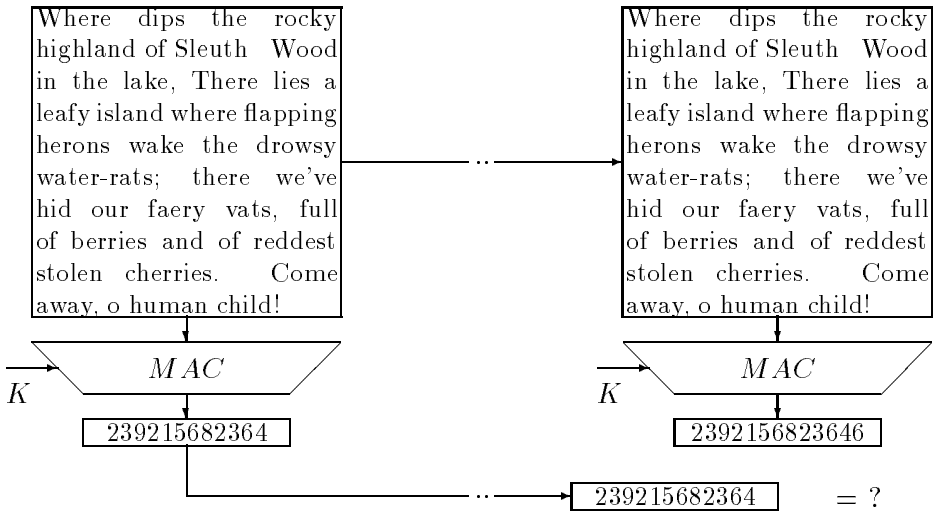


Figure 8: Using a Message Authentication Code for information authentication

block as the MAC. However, recent research has indicated that this approach is less secure than previously believed [24]; again, the birthday paradox plays a role in this work.

For a MAC, the equivalent of the Vernam scheme exists. This implies that one can design a MAC algorithm which is unconditionally secure, in the sense that the security of the MAC is independent of the computing power of the opponent. The requirement is again that the secret key is used only once. The basic idea of this approach is due to G.J. Simmons and dates back to the seventies (see for example [28]). It turns out that these algorithms can be computationally very efficient, since the properties required from this primitive are combinatorial rather than cryptographic. Recent constructions are therefore one order of magnitude faster than other cryptographic primitives (encryption algorithms, hash functions), and achieve speeds up to 1 Gbit/s on fast processors [12]. A simple example is described here, which is derived from Reed-Solomon codes for error-correction [15]. The key consists of two $n$-bit words denoted with $K_1$ and $K_2$. The plaintext is divided into $t$ $n$-bit words, denoted with $p_1$ through $p_t$. The MAC, which consists of a single $n$-bit word, is computed based on a simple polynomial evaluation:

$$\mathrm{MAC}_{K_1,K_2}(x) = K_1 + \sum_{i=1}^{t} p_i \cdot (K_2)^i ,$$

where addition and multiplication are to be computed in the finite field with $2^n$ elements. It can be proved that the probability of creating another valid message/MAC pair is upper bounded by $t/2^n$. A practical choice is $n = 64$, which results in a 128-bit key. For messages up to 1 Mbyte, the success probability of a forgery is then less than $1/2^{47}$. Note that it turns out to be possible to reuse $K_2$; however, for every message a new key $K_1$ is required. This key could be generated from a short initial key using an additive stream cipher, but then the unconditional security is lost. However, one can argue that it is easier to understand the security of this scheme than that of a computationally secure MAC.

### 3.2 Digital signatures

A digital signature is the electronic equivalent of a manual signature on a document. It provides a strong binding between the document and a person, and in case of a dispute, a third party can decide whether or not the signature is valid. Of course a digital signature will not bind a person and a document, but will bind a key and a document. Additional measures are then required to couple the person to his or her key. Note that for a MAC, both Alice and Bob can compute the MAC, hence a third party cannot distinguish between them. While block ciphers (and even one-way functions) can be used to construct digital signatures, the most elegant and efficient constructions for digital signature rely on public-key cryptography.

If Alice wants to sign some information $P$ intended for Bob, she adds some redundancy to the information, resulting in $\tilde{P}$, and decrypts the resulting text with her secret key. This operation can only be carried out by Alice. Upon receipt of the signature, Bob encrypts it using Alice's public key, and verifies that the information $\tilde{P}$ has the prescribed redundancy. If so, he accepts the signature on $P$ as valid. Such a digital signature (which is a signature 'giving message recovery') imposes an additional condition on the public-key system: $P_A(S_A(P)) = P$. Note that anyone who knows Alice's public key can verify the signature. The RSA public-key encryption scheme is a bijection (a trapdoor one-way permutation), and thus it allows for the construction of digital signatures with message recovery. For RSA, digital signatures and encryption can be combined, as indicated in Figure 9.

Note that without redundancy, the following problem occurs: Eve can pick a random string, and send this to Bob. Bob encrypts the string using Alice's public key, and obtains another random string $P'$. In this way Eve has 'forged' a signature on the text $P'$ which may or may not be useful (depending on the context). However, if one imposes that the result of the encryption has to have a certain redundancy, Bob is sure that only Alice can have created the signature (by starting from $\tilde{P}$), as Eve can only 'sign' random strings $P'$.

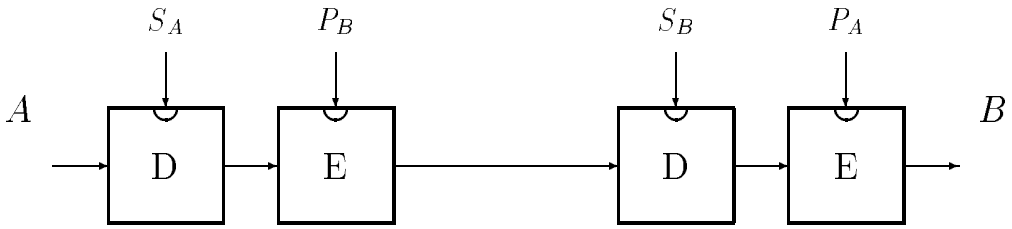


Figure 9: Public-key cryptography: secrecy and authentication

If Alice wants to sign very long messages (without encrypting them), this approach results in signatures that are as long as the message. Moreover, signing with a public-key system is a relatively slow operation. In order to solve these problems, Alice does not sign the information itself, but the hash result of the information computed with an MDC. The signature now consists of a single block, which is appended to the information (this is called a digital signature 'with appendix'). In order to verify such a signature, Bob recomputes the MDC of the message and encrypts the signature with Alice's public key. If both operations give the same result, Bob accepts the signature as valid. MDCs used in this way need to be collision resistant: otherwise Alice can sign a message $P$, and later be held accountable for a fraudulent message $P'$ with the same MDC (and thus with the same signature).

Note that there exist other signature schemes with appendix (such as the DSA [7]), which are not derived immediately from a public-key encryption scheme. For these schemes one can define a 'signing operation' (using the secret key) and a 'verification operation' (using the public key), without referring to 'decryption' and 'encryption' operations.

# 4 Analysis and design of conventional cryptographic algorithms

In this section we compare three approaches to the design of cryptographic algorithms. Next we describe the typical phases in the life of an algorithm. Then we contrast brute force and shortcut attacks, public and secret algorithms, and weak and strong algorithms.

## 4.1 Three approaches in cryptography

Present day cryptology tries to develop provably secure and efficient cryptographic algorithms. Often such algorithms are not available; therefore cryptographic algorithms are studied following three approaches: the information theoretic approach, the complexity theoretic approach, and the system based approach. These approaches differ in the assumptions about the capabilities of an opponent, in the definition of a cryptanalytic success, and in the notion of security.

The most desirable from the viewpoint of the cryptographer are unconditionally secure algorithms; this design approach is also known as the *information theoretic* approach. However, few such schemes exist: examples are the Vernam scheme (§2.2), and the MAC based on Reed-Solomon codes (§3.1.2). While they are computationally very efficient, the cost in terms of key material may be prohibitively large (certainly for the Vernam scheme). For most applications one has to live with schemes which offer only conditional security.

A second approach is to reduce the security of his scheme to that of other well known difficult problems, or to that of other cryptographic primitives. The *complexity theoretic* approach starts from an abstract model for computation, and assumes that the opponent has limited computing power within this model [8]. This approach has many positive sides:

- It forces the formulation of exact definitions, and to state clearly the security properties and assumptions.
- Once the proofs are written down, anyone can verify them and decide whether or not they are correct.

However, this approach has also some limitations:

- Many cryptographic applications need building blocks, such are one-way functions, one-way permutations, or block ciphers, which cannot be reduced to other primitives. In terms of the existence of such primitives, complexity theory has only very weak results: in non-uniform complexity (Boolean circuits) the best proved thus far is that there exist functions which are twice as hard to invert as to compute, which is far too weak to be of any use in cryptography [13].
- Sometimes the resulting scheme is not very efficient (for example, a conventional scheme which is as slow as a public-key scheme but provably as secure as factoring is not very attractive from a practical standpoint), or the security reduction is quite loose: for example, the correct properties are proved, but the exact security level is hard to obtain from the result.

- The assumptions might be too strong.
- The security property proved might not be the one needed for a particular application.

This implies that for many instances, the cryptographer has to rely on the *system-based* or *practical approach.* This approach tries to produce practical solutions; the security estimates are based on the best algorithm known to break the system and on realistic estimates of the necessary computing power or dedicated hardware to carry out the algorithm. By trial and error procedures, several *cryptanalytic principles* have emerged, and it is the goal of the designer to avoid attacks based on these principles. The second aspect is to design *building blocks with provable properties*, and to assemble such basic building blocks to design cryptographic primitives.

## 4.2 Life cycle of a cryptographic algorithm

A cryptographic algorithm usually starts with a new idea of a cryptographer. A first step should always consist of an evaluation of the resulting algorithm, in which the cryptographer tries to determine whether or not the scheme is secure. If the scheme is unconditionally secure, he has to write the proofs, and to convince himself that the model is correct and matches the application. For computational security, it is again very important to write down security proofs, and to check these for subtle flaws. Moreover, one has to assess whether the assumptions behind the proofs are realistic. For the system-based approach, it is important to prove partial results, and to write down arguments which should convince others of the security of the algorithm. One (obvious) step which is often overlooked is to write down a complete specification of the scheme; it is not attractive to evaluate a vague collection of design principles. Often such cryptographic algorithms have security parameters (the number of steps, the size of the key, ...); it is then very important to give lower bounds for these parameters, and to indicated the value of the parameters which corresponds to a certain security level. Another aspect is to explain in which environments the algorithm can offer practical (or theoretical) advantages.

The next step is then the publication of the algorithm at a conference, in a journal, or in an Internet Request for Comment (RFC). This (hopefully) results in an *independent* evaluation of the algorithm. Often more or less subtle flaws are then discovered by other researchers. This can vary from small errors in proofs, to complete security breaks. Depending on the outcome, this can lead to a small fix of the scheme or to abandoning the idea altogether. Sometimes such weaknesses can be found 'in real-time' when the author is presenting his ideas at a conference, but often evaluating a cryptographic algorithm is a very time consuming task; for example, the design effort of the Data Encryption Standard (DES) (cf. appendix A) has been more than 17 man-years, and the open academic evaluation since has taken a multiple of this effort. Cryptanalysis is quite destructive; in this respect it differs from usual scientific activities, even when proponents of competing theories criticize each other.

Few algorithms survive the evaluation stage; ideally, this stage should last for several years. The survivors can be integrated into products and find their way to the market. Sometimes they are standardized by organizations such as NIST (National Institute of Standards and Technology, US), ANSI, IEEE, IETF, or ISO.

As will be explained below, even if no new security weaknesses are found, the security of a cryptographic algorithm degrades over time; if the algorithm is not modular, the moment will come when it has to be take out of service.

This scheme does not work perfectly, as the evaluation effort is usually much larger than the design effort, and there is a clear shortage of cryptanalysts. Cryptanalysts will prefer to analyze algorithms for which they can maximize their credit: this includes algorithms from well known cryptographers or companies, and algorithms which are already widely used (but this contradicts the ideal process as outlined above). Of course they will also look at algorithms which they can break very quickly, as this offers a quick return on 'investment.'

## 4.3 Brute force attacks versus shortcut attacks

A detailed description of the evaluation procedures for cryptographic algorithms is beyond the scope of this paper. We restrict ourselves to explaining the difference between brute force attacks and shortcut attacks.

### 4.3.1 Brute force attacks

Brute force attacks are attacks which exist against any cryptographic algorithm which is conditionally secure, no matter how it works internally. These attack only depend on the size of the external parameters of the algorithm, such as the block length of a block cipher, or the key length of any encryption algorithm or MAC. It is the task of the designer to choose the external parameters in such a way that brute force attack are infeasible.

A typical brute force attack against an encryption algorithm or a MAC is an exhaustive key search; it is equivalent to breaking into a safe by trying all the combinations of the lock. The lock should be designed such that this is not feasible in a reasonable amount of time. This attack is very realistic, as it requires only a few known plaintext/ciphertext (or plaintext/MAC) pairs. It can be precluded by increasing the key length: adding one bit to the key doubles the time for exhaustive key search. One should also ensure that the key is selected uniformly at random in the key space.

On a standard PC, trying a single key for a typical algorithm requires a few microseconds. For example, a 40-bit key (which is at present the maximum value allowed by the US government for export) will be recovered after a few hundred hours. If a LAN with 100 machines can be used, one can find the key in a few hours. Therefore IBM has called their DES variant with 40-bit keys a 'Commercial Data Masking Facility,' rather than an encryption algorithm. For a 56-bit key such as DES (which can be exported from the US under restrictive conditions), a key search requires a few months if several thousand machines are available (as has been demonstrated in the first half of 1997).

However, if dedicated hardware is used, a different picture emerges. With an investment of 1 million US$, a 56-bit key can be found in 1 hour, and if the machine is used continuously, the cost per key is under 100 US$ [31].

In selecting the key size, one has to take into account that some information (such as medical data) requires long term protection (20 to 50 years), while other information (such as economic data) can loose its value in a few hours or a few days. This influences the required security level.

One should also take into account "Moore's law" [26], which states that computers double their speed every 18 months (for the same cost). This implies that a 64-bit key, which offers a reasonable security level for the time being, is probably not sufficient for data which needs to be protected for 10 years. Such applications will need keys of at least 80 bits. As the cost

of increasing the key size is quite low, it is advisable to design new algorithms with variable key size up to 128...256 bits.

There exist many other brute force attacks. For example, it turns out the security of a block cipher in the CBC or CFB-mode is decreased by what is called the 'matching ciphertext' attack. As a consequence of the birthday paradox, after $2^{n/2}$ encryptions with a single key, information starts to leak on the plaintext (due to matches in the internal memory, which correspond to matching ciphertexts). This attack can be a problem for present day block ciphers with a 64-bit block length. It can only be prevented by designing new block ciphers with larger block lengths (128 or more), or by changing the key frequently. This attack also affects the choice for the internal memory of self-synchronizing stream ciphers (the parameter $M$). A related but less serious problem can occur for additive stream ciphers (or in the OFB mode).

### 4.3.2 Shortcut attacks

Many algorithms are less secure than suggested by the size of their external parameters. It is often possible to find more effective attacks than trying all keys. Assessing the strength of an algorithm requires cryptanalytic skills and experience, and often hard work. During the last 10 years powerful new tools have been developed: this includes differential cryptanalysis [1], which analyzes the propagation of differences through cryptographic algorithms, linear cryptanalysis [18], which is based on the propagation of bit correlations, and fast correlation attacks on stream ciphers [19].

The design of new algorithms according to the system-based approach is not a memoryless process: when new cryptanalytic techniques are developed, the cryptographers invent new designs which provide complete (or at least improved) resistance against these new attacks. In this way cryptology develops by trial and error procedures.

## 4.4 Public versus secret algorithms

The open and independent evaluation process described in §4.2 offers a strong argument for publishing all details of a cryptographic algorithm. Publishing the algorithm opens it up for public scrutiny, and is the best way to guarantee that it is as strong as claimed. (Note that a public algorithm should not be confused with a public-key algorithm.) Published algorithms can be standardized, and will be available from more than one source.

Nevertheless, certain governments and organizations prefer to keep their algorithms secret. They argue (correctly) that obtaining the algorithm raises an additional barrier for the attacker. Moreover, governments want to protect their know-how on the design of cryptographic algorithms. However, obtaining a description of the algorithm is often not harder than just bribing one person. This approach is acceptable, provided that sufficient experience and resources are available for *independent* evaluation and re-evaluation of the algorithm.

## 4.5 Insecure versus secure algorithms

In spite of the fact that secure cryptographic algorithms are available, which offer good performance, one encounters in many applications very insecure cryptographic algorithms. For example, popular software sometimes 'encrypts' data by adding a constant key word to all data words. Several reasons can be indicated for this:

- one excuse is performance: while it is true that adding a constant will always be faster than strong encryption, it should be noted that in software, current encryption algorithms achieve between 10 and 100 Mbit/s; this is sufficient for many applications;
- legal and/or export restrictions: as indicated above, for national security reasons, certain countries (such as the USA) do not allow the export of strong encryption algorithms; some countries (such as France) do not allow for strong encryption within their territory (unless the keys are handed over to the government);
- commercial pressure: companies often rush their security solutions to market, without allowing for sufficient time for the slow evaluation process;
- evolution of computing power: the strength of a cryptographic algorithm erodes over time because of Moore's law; often there exists a large inertia to replace or upgrade an algorithm. A typical example is the DES: its 56-bit key is no longer sufficient for most applications, but it is still widely in use;
- evolution of cryptanalysis: if designers are not aware of the latest developments in cryptanalysis, it is quite likely that their algorithms will not resist these attacks. For example, the FEAL block cipher with 8 rounds, which was published in 1987, can now be broken with only 10 chosen plaintexts.

# 5 Concluding remarks

Securing an application should be based on a careful analysis of the risks and vulnerabilities, and of the requirements for the cryptographic algorithms.

Before starting the design of an algorithm, one should decide which type of algorithm is needed: for encryption one has to choose between a block cipher (which is a flexible tool that can also be used to construct MACs and MDCs), and an additive stream cipher or a self-synchronizing stream cipher. Each of these choices will lead to different security requirements and corresponding constraints. One also has to decide in how far the algorithm needs to be tuned towards a specific application (for example, image processing).

Designing a very slow but secure cryptographic algorithm is not very difficult for an experienced cryptographer. However, if optimal performance has to be achieved, such a design requires a combination of cryptologic know-how (design and cryptanalysis) and deep insight in the technology which is going to be used. This can vary from the specific properties of a CPU (instruction set, cache memory, parallelism) to the limitations and constraints of certain hardware environments (FPGA, standard cell, cellular neural networks, etc.). Examples of such new designs can be found in recent proceedings of the "Fast Software Encryption" workshop [2, 9, 23].

As a rule of thumb, an algorithm should not be used in an application before a design and evaluation effort of at least 1 person-year has been spent on it. Given this effort, it is advisable that the algorithm is as *modular* as possible, which implies that it can offer different security levels, and allows for a variable-size key. Implementors should be prepared to upgrade their algorithm as a consequence of some progress in cryptanalysis. However, in order to attract evaluation, it is also important to fix a single 'nominal' scheme as an evaluation target.

## References


[1] E. Biham, A. Shamir, *"Differential Cryptanalysis of the Data Encryption Standard,"* Springer-Verlag, 1993.

[2] *"Fast Software Encryption," LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997.

[3] D.W. Davies, W.L. Price, *"Security for Computer Networks. An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer,"* (Second Edition), Wiley, 1989.

[4] W. Diffie, M.E. Hellman, "New directions in cryptography," *IEEE Trans. on Information Theory*, Vol. IT–22, No. 6, 1976, pp. 644–654.

[5] FIPS 46, *"Data Encryption Standard,"* Federal Information Processing Standard (FIPS), Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C., January 1977.

[6] FIPS 81, *"DES Modes of Operation,"* Federal Information Processing Standard (FIPS), Publication 81, National Bureau of Standards, US Department of Commerce, Washington D.C., December 1980.

[7] FIPS 186, *"Digital Signature Standard,"* Federal Information Processing Standard (FIPS), Publication 186, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., May 1994.

[8] M.R. Garey, D.S. Johnson, *"Computers and Intractability: A Guide tot the Theory of NP-Completeness,"* W.H. Freeman and Company, San Francisco, 1979.

[9] *"Fast Software Encryption," LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996.

[10] J.A. Gordon, "Strong RSA keys," *Electronic Letters*, Vol. 20, No. 12, 1984, pp. 514–516.

[11] P. Guam, "Cellular automaton public key cryptosystem," *Complex Systems*, Vol. 1, pp. 51–56, 1987.

[12] S. Halevi, H. Krawczyk, "MMH: software message authentication in the Gbit/second rates," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 172–189.

[13] A.P.L. Hiltgen, "Construction of feebly-one-way families of permutations," *Advances in Cryptology, Proc. Auscrypt'92, LNCS 718*, J. Seberry and Y. Zheng, Eds., Springer-Verlag, 1993, pp. 422–434.

[14] ISO/IEC 10116, *"Information technology – Security techniques – Modes of operation of an n-bit block cipher algorithm,"* IS 10116, 1991.

[15] T. Johansson, G. Kabatianskii, B. Smeets, "On the relation between A-codes and codes correcting independent errors," *Advances in Cryptology, Proc. Eurocrypt'93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 1–11.

[16] D. Kahn, *"The Codebreakers. The Story of Secret Writing,"* MacMillan, New York, 1967.

[17] N. Koblitz, *"A Course in Number Theory and Cryptography,"* Springer-Verlag, 1987.

[18] M. Matsui, "The first experimental cryptanalysis of the Data Encryption Standard," *Advances in Cryptology, Proc. Crypto'94, LNCS 839*, Y. Desmedt, Ed., Springer-Verlag, 1994, pp. 1–11.

[19] W. Meier, O. Staffelbach, "Fast correlation attacks on stream ciphers," *Journal of Cryptology*, Vol. 1, 1989, pp. 159–176.

[20] R. Merkle, *"Secrecy, Authentication, and Public Key Systems,"* UMI Research Press, 1979.

[21] A.J. Menezes, P.C. van Oorschot, S. Vanstone, *"Handbook of Applied Cryptography,"* CRC Press, 1996.

[22] *"State of the Art and Evolution of Computer Security and Industrial Cryptography," Lecture Notes in Computer Science 741*, B. Preneel, R. Govaerts, and J. Vandewalle, Eds., Springer-Verlag, 1993.

[23] *"Fast Software Encryption," LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995.

[24] B. Preneel, P.C. van Oorschot, "MDx-MAC and building fast MACs from hash functions," *Proc. Crypto'95, LNCS 963*, D. Coppersmith, Ed., Springer-Verlag, 1995, pp. 1–14.

[25] R.L. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, Vol. 21, No. 2, 1978, pp. 120–126.

[26] R.R. Schaller, "Moore's law: past, present, and future," *IEEE Spectrum*, Vol. 34, No. 6, June 1997, pp. 53–59.

[27] C.E. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal*, Vol. 28, No. 4, 1949, pp. 656–715.

[28] *"Contemporary Cryptology: The Science of Information Integrity,"* G.J. Simmons, Ed., IEEE Press, 1991.

[29] D. Stinson, *"Cryptography. Theory and Practice,"* CRC Press, 1995.

[30] G.S. Vernam, "Cipher printing telegraph system for secret wire and radio telegraph communications," *Journal American Institute of Electrical Engineers*, Vol. XLV, 1926, pp. 109–115.

[31] M.J. Wiener, "Efficient DES key search," *Technical Report TR-244*, School of Computer Science, Carleton University, Ottawa, Canada, May 1994. Presented at the rump session of Crypto'93.

# A The Data Encryption Standard (DES)

It would be quite strange to write an article on (conventional) cryptographic algorithms without giving a single example. This appendix contains a brief description of the Data Encryption Standard, which has been and probably still is the worldwide de facto standard encryption algorithm. While the DES is nearing the end of its lifetime (it will be replaced by the Advanced Encryption Standard or AES in the next few years), it is clear that DES has played a crucial role in the development of present-day cryptology. One can expect that DES or extensions of DES will still be in use for a least 15 more years to come.

The Data Encryption Standard (DES) has been developed jointly by IBM and NSA (National Security Agency, USA); it was published as a standard by the NBS (National Bureau of Standards) in 1977 (NBS is now called NIST, National Institute for Standards and Technology). DES has a block length of 64 bits and a 56-bit key (the key is in fact 64 bits long, but the parity bits are ignored by the algorithm). DES consists of a bit permutation ($IP$ or Initial Permutation), 16 consecutive rounds, and a second bit permutation ($IP^{-1}$) (see Figure 10). Each round has a specific structure, which is called a Feistel structure. The 64 input bits at the beginning of round $t$ are divided into a left and a right half of 32 bits each, denoted with $L_t$ and $R_t$ respectively. The new left half is the old right half, and the new right half is the modulo 2 sum of the old left half and a function of the key and the old right half:

$$L_{t+1} = R_t, \qquad R_{t+1} = L_t \oplus f(R_t, K_t)\,.$$

Here $K_t$ denotes a selection of 48 key bits, and $f$ consists of an expansion $E$ (from 32 to 48 bits), followed by the addition modulo 2 of the key $K_t$, a non-linear substitution $S$ (with 48 input bits and 32 output bits) and a bit permutation $P$. The substitution $S$ contains 8 S-boxes which map 6 input bits to 4 output bits; this substitution forms the only operation in DES which is non-linear with respect to addition modulo 2; it forms thus an essential component of the DES algorithm.

**Note:**
*If the S-boxes would be linear w.r.t. addition modulo 2, the DES encryption operation could be written as an affine transformation of the form $AX + B$, where $A$ denotes a binary $64 \times 64$ matrix, and $B$ a binary $64 \times 1$ vector. Breaking such a weak DES variant would require about 65 plaintext/ciphertext pairs: these would allow for sufficient information to solve for $A$ and $B$.*

The decryption is also an iterative process, which is essentially identical to the encryption (the keys have to be used in the reverse order).

$$R_t = L_{t+1}, \qquad L_t = R_{t+1} \oplus f(L_{t+1}, K_t).$$

Note that this feature is very important to reduce the cost of hardware implementations. The detailed description of the different building blocks can be found in [5].

Currently, the main security weakness of the DES is the short key (cf. §4.3.1). After more than twenty years of intensive research, the best known shortcut attack is the linear attack of [18]: the key can be recovered in a relatively short time if the opponent knows $2^{43}$ plaintext/ciphertext pairs. However, this attack is only of theoretical value. The 2nd best attack is a differential attack: it requires $2^{47}$ *chosen* plaintext/ciphertext pairs [1]. The complexity of these attacks provides some evidence that the DES is a very strong cryptographic algorithm.

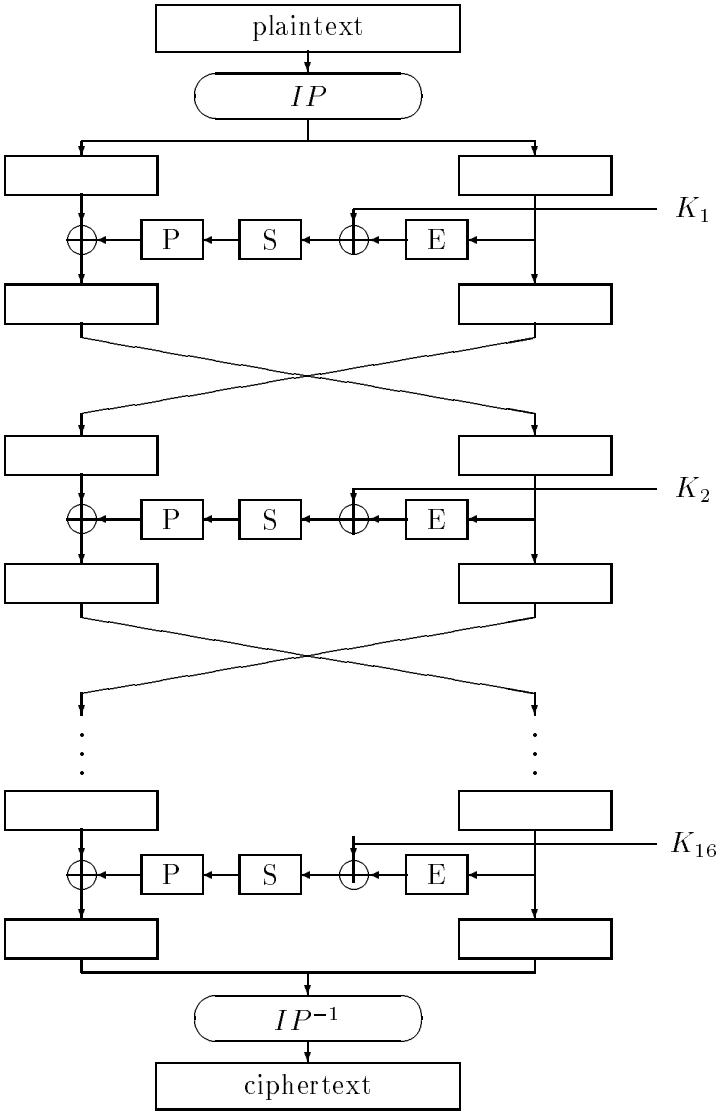


Figure 10: The Data Encryption Standard (DES).