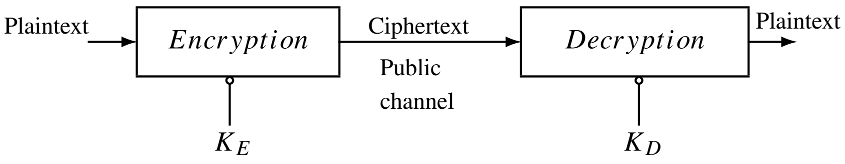
1

Introduction to cryptography, codes, Boolean, and vectorial functions

1.1 Cryptography

A fundamental objective of *cryptography* is to enable two persons to communicate over an insecure channel (a public channel such as the internet) in such a way that any other person is unable to recover their messages (constituting the *plaintext*) from what is sent in its place over the channel (the *ciphertext*). The transformation of the plaintext into the ciphertext is called *encryption*, or enciphering. It is ensured by a *cryptosystem*. Encryption–decryption is the most ancient cryptographic activity (ciphers already existed in the fourth century BC) but its nature has deeply changed with the invention of computers, because the *cryptanalysis* (the activity of the third person, the eavesdropper, who aims at recovering the message, or better, the secret data used by the algorithm – which is assumed to be public) can use their power. Another important change will occur (see *e.g.*, [70, 360, 832]), at least for public-key cryptography (see the definition below), when quantum computers become operational.

The encryption algorithm takes as input the plaintext and an encryption key K_E, and it outputs the ciphertext. The *decryption* (or deciphering) algorithm takes as input the ciphertext and a private[1] decryption key K_D. It outputs the plaintext.



For being considered robust, a cryptosystem should not be cryptanalyzed by an attack needing less than 2^{80} elementary operations (which represent thousands of centuries of computation with a modern computer) and less than billions of plaintext–ciphertext pairs. In particular, an exhaustive search of the secret parameters of the cryptosystem (consisting in trying every possible value of them until the data given to the attacker match the computed data) should not be feasible in less than 2^{80} elementary operations. In fact, we most often even want that there is no faster cryptanalysis than exhaustive search.

[1] According to principles already stated in 1883 by A. Kerckhoffs [688], who cited a still more ancient manuscript by R. du Carlet [207], only the key(s) need absolutely to be kept secret – the confidentiality should not rely on the secrecy of the encryption method – and a cipher cannot be considered secure if it can be decrypted by the designer himself without using the decryption key.

https://doi.org/10.1017/9781108606806.003 Published online by Cambridge University Press

Note that the term of cryptography is often used indifferently for naming the two activities of designing cryptosystems and of cryptanalyzing them, while the correct term when dealing with both is *cryptology*.

1.1.1 Symmetric versus public-key cryptosystems

If the encryption key is supposed to be secret, then we speak of *conventional cryptography* or of *private-key cryptography*. We also speak of *symmetric cryptography* since the same key can then be used for K_E and K_D. In practice, the principle of conventional cryptography relies then on the sharing of a private key between the sender of a message (often called Alice) and its receiver (often called Bob). Until the late 1970s, only symmetric ciphers existed.

If the encryption key can be public, then we speak of *public-key cryptography* (or *asymmetric cryptography*), which is preferable to conventional cryptography, since it makes it possible to securely communicate without having previously shared keys in a secure way: every person who wants to receive secret messages can keep secret a decryption key and publish an encryption key; if n persons want to secretly communicate pairwise using a public-key cryptosystem, they need n encryption keys and n decryption keys, when conventional cryptosystems will need $\binom{n}{2} = \frac{n(n-1)}{2}$ keys. Of course, it must be impossible to deduce in reasonable time, even with huge computational power, the private decryption key from the public encryption key. Such requirement is related to the problem of building *one-way function*s, that is, functions such that computing the image of an element is fast (*i.e.*, is a problem of polynomial complexity), while the problem of computing the preimage of an element has exponential complexity.

All known public-key cryptosystems, such as RSA, which uses operations in large rings [846], allow a much lower data throughput; they also need keys of sizes 10 times larger than symmetric ciphers for ensuring the same level of security. Some public-key cryptosystems, such as those of McEliece and Niederreiter (based on codes) [846], are faster, but have drawbacks, because the ciphertext and the plaintext have quite different lengths, and the keys are still larger than for other public-key cryptosystems.[2] Private-key cryptosystems are then still needed nowadays for ensuring the confidential transfer of large data. In practice, they are widely used for confidentiality in the internet, banking, mobile communications, etc., and their study and design are still an active domain of research. Thanks to public-key cryptosystems, the share-out of the necessary secret keys for the symmetric cipher can be done without using a secure channel (the secret keys for conventional cryptosystems are strings of a few hundreds of bits only and can then be encrypted by public-key cryptosystems). The protagonists can then exchange safely, over a public channel such as the internet, their common private encryption–decryption key, called a *session key*. Protocols specially devoted to key exchange can also be used.

The change caused by the intervention of quantum computers will be probably much less important for symmetric than for public-key cryptography. Most current symmetric ciphers

[2] Code-based, lattice-based, and other "postquantum" cryptosystems are, however, actively studied, mainly because they would be alternatives to RSA and to the cryptosystems based on the discrete logarithm, in case an efficient quantum computer could be built in the future, which would break them.

seem secure against attacks by quantum computers (Grover's algorithm [576], which, given a black box with N possible inputs and some output, deduces with high probability from the results of $\mathcal{O}(\sqrt{N})$ evaluations the supposedly unique input,[3] will probably have as an impact the necessity to double the length of the keys).

1.1.2 Block ciphers versus stream ciphers

The encryption in a symmetric cipher can be treated block by block in a so-called *block cipher* (such as the Advanced Encryption Standard, AES [403, 404]). The binary plaintext is then divided into blocks of the same size, several blocks being encrypted with the same key (and a public data called initial vector being changed more often). It can also be treated in a *stream cipher* [463], through the addition, most often mod 2, of a *keystream* of the same size as the plaintext, output by a pseudorandom generator (PRG) parameterized by a secret key (the keystream can be produced symbol by symbol, or block by block when the PRG uses a block cipher in a proper mode[4]). A quality of stream ciphers is to avoid error propagation, which gives them an advantage in applications where errors may occur during the transmission.

The ciphertext can be decrypted in the case of block ciphers by inverting the process and in the case of stream ciphers by the same bitwise addition of the keystream, which gives back the plaintext. Stream ciphers are also meant to be faster and to consume less electric power (which makes them adapted to cheap embedded devices). The triple constraint of being lightweight and fast while ensuring security is a difficult challenge for stream ciphers, all the more since they do not have the advantage of involving several rounds like block ciphers (their security is dependent on the PRG only). And the situation is nowadays still more difficult because modern block ciphers such as the AES are very fast. This difficulty has been illustrated by the failure of all six stream ciphers submitted to the 2000–2003 NESSIE project (New European Schemes for Signatures, Integrity and Encryption) [901], whose purpose was to identify secure cryptographic primitives. NESSIE has then been followed by the contest eSTREAM [495] organized later, between 2004 and 2008, by the European Union (EU) ECRYPT network.

As mentioned in [242], the price to pay for these three constraints described above is that security proofs hardly exist for efficient stream ciphers as they do for block ciphers. This is a drawback of stream ciphers, compared to block ciphers.[5] The only practical possibility for verifying the security of efficient stream ciphers (in particular, the unpredictability of the keystream they generate) is to prove that they resist the known attacks. It is then advisable to include some amount of randomness in them, so as to increase the probability of resisting future attacks.[6]

[3] Or equivalently finds with high probability a specific entry in an unsorted database of N entries.
[4] Note, however, that stream ciphers are often supposed to be used on lighter devices than block ciphers (typically not needing cryptoprocessors, for instance).
[5] However, the security of block ciphers is actually proved under simplifying hypotheses, and it has been said by Lars Knudsen that "what is provably secure is probably not."
[6] Some stream cipher proposals, such as the Toyocrypt, LILI-128 and SFINKS ciphers, learned this at their own expense; see [387].

Proving the security of a cipher consists of reducing it to the intractability of a hard problem (a problem that has been extensively addressed by the academic community, and for which only algorithms of exponential or subexponential complexity could be found), implying that any potential attack on it could be used for designing an efficient algorithm (whose worst-case complexity would be polynomial in the size of its input) solving the hard problem.

Note that provably secure stream ciphers do exist (some proposals are even unconditionally secure, that is, are secure even if the attacker has unlimited computational power, but limited storage or access); see for instance the proposals by Alexi–Chor–Goldreich–Schnorr (whose security is reducible to the intractability of the RSA problem) or Blum–Blum–Shub [98] (whose security is reducible to the intractability of the quadratic residue problem modulo pq, where p and q are large primes), or the stream cipher QUAD [61] (based on the iteration of a multivariate quadratic system over a finite field, and whose security is reducible to the intractability of the so-called multivariate quadratic (MQ) polynomial problem). But they are too slow and too heavy for being used in practice. Even in the case of QUAD, which is the fastest, the encryption speed is lower than for the AES. And this is still worse when security is ensured unconditionally. This is why the stream ciphers using Boolean functions (see below) are still much used and studied.

1.2 Error-correcting codes

The objective of error-detecting/-correcting codes in *coding theory* is to enable digital communication over a noisy channel, in such a way that the errors of transmission can be detected by the receiver and, in the case of error correcting codes, corrected. General references are [63, 780, 809]. Shannon's paper [1033] is also prominent.

Without correction, when an error is detected, the information needs to be requested again by the receiver and sent again by the sender (such procedure is called an Automatic Repeat reQuest, ARQ). This is what happened with the first computers: working with binary words, they could detect only one error (one bit) in the transmission of $(x_1, \dots, x_k)$, by adding a parity bit $x_{k+1} = \bigoplus_{i=1}^{k} x_i$ (this transformed the word of length k into a word of length $k+1$ having even *Hamming weight*, *i.e.*, an even number of nonzero coordinates, which was then sent over the noisy channel; if an error occurred in the transmission, then, assuming that only one could occur, this was detected by the fact that the received word had odd Hamming weight).

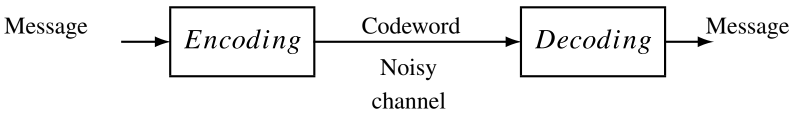
With correction, the ARQ is not necessary, but this requires in practice that fewer errors have occurred than for detection (see below). Hybrid coding techniques exist then that make a trade-off between the two approaches.

The aim of error detection/correction is achieved by using an encoding algorithm that transforms the information (assumed to be a sequence over some alphabet $\mathcal{A}$) before sending it over the channel. In the case of block coding,[7] the original sequence (the *message*) is treated as a list of vectors (words) of the same length – say k – called *source vectors* which are encoded into *codewords* of a larger length – say[8] n. If the alphabet with which the words

[7] We shall not address convolutional coding here.
[8] When dealing with Boolean functions, the symbol n will be often devoted to their number of variables; the length of the codes they will constitute will then not be n but $N = 2^n$. See Section 1.3.

are built is the field $\mathbb{F}_2$ of order 2, we say that the code is binary. If the code is not binary, then the symbols of the alphabet will have to be transformed into binary vectors before being sent over a binary channel.

Thanks to the length extension, called *redundancy*, the codewords sent over the channel are some of all possible vectors of length n. The set C of all codewords is called the *code* (for instance, in the case of the detecting codes using a parity bit as indicated above, the code is made of all binary words of length $n = k + 1$ and of even Hamming weights; it is called the *parity code*). The only information the receiver has, concerning the sent word, is that it belongs to C.



1.2.1 Detecting and correcting capacities of a code

The decoding algorithm of an error-detecting code is able to recognize if a received vector is a codeword. This makes possible to detect errors of transmission if (see [585]) denoting by d the minimum *Hamming distance* between codewords, *i.e.*, the minimum number of positions at which codewords differ (called the *minimum distance* of the code), no more than $d - 1$ coordinates of the received vector differ from those of the sent codeword (condition for having no risk that a codeword different from the sent one can be received and then accepted). In the case of an error-correcting code, the decoding algorithm can additionally correct the errors of transmission, if their number is smaller than or equal to the so-called *correction capacity* of the code. This capacity equals $e = \left\lfloor \frac{d-1}{2} \right\rfloor$, where "$\lfloor\ \rfloor$" denotes the integer part (and so, roughly, a code can detect twice as many errors than it can correct), since the condition for having no risk that a vector corresponds, as received vector, to more than one sent codeword with at most t errors of transmission in each case is that $2t < d$. Indeed, in order to be always able (theoretically) to recover the correct codeword, we need that, for every word y at distance at most t from a codeword x, there does not exist another codeword x' at distance at most t from y, and this is equivalent to saying that the Hamming distance between any two different codewords is larger than or equal to $2t + 1$:

- If there exist a vector y and two codewords x and x' at Hamming distance at most t from y, then we have $d \le 2t$ by the triangular inequality on distances.
- Conversely, if there exist two codewords x and x' at Hamming distance $\delta \le 2t$ from each other, then there exists a vector y such that $d_H(x, y) \le t$ and $d_H(x', y) \le t$ (let I be the set of positions where x and x' coincide; take $y_i = x_i$ when $i \in I$ and among the δ others, take for instance $\lfloor \frac{\delta}{2} \rfloor$ coordinates of y equal to those of x and the $\lceil \frac{\delta}{2} \rceil$ others equal to those of x').

In practice, determining d and then $e = \left\lfloor \frac{d-1}{2} \right\rfloor$ and showing that they are large is not sufficient. We still need to have an efficient decoding algorithm to recover the sent codeword. The naive method consisting in visiting all codewords and keeping the nearest one from the received word is inefficient because the number 2^k of codewords is too large, in general.

Determining the nearest codeword from a received vector is called *maximum likelihood decoding*.

The correction capacity e is limited by the *Hamming bound* (or *sphere-packing bound*): since all the balls $B(x,e) = \{y \in \mathcal{A}^n; d_H(x,y) \leq e\}$, of radius e and centered in codewords are pairwise disjoint, and since there are $|C|$ of them, the size of their union equals $|C| \sum_{i=0}^{e} \binom{n}{i}(q-1)^i$, where q is the size of the alphabet. This union is a subset of $\mathcal{A}^n$. This implies the following:

$$|C| \sum_{i=0}^{e} \binom{n}{i}(q-1)^i \leq q^n.$$

The codes that achieve this bound with equality are called *perfect code*s.

Puncturing, shortening, and extending codes

The *punctured code* of a code C is the set of vectors obtained by deleting the coordinate at some fixed position i in each codeword of C; we shall call such transformation *puncturing at position* i. This operation can be iterated, and we shall still speak of puncturing a code when deleting the codeword coordinates at several positions.

The *shortened code* of a code C is the set of vectors obtained by keeping only those codewords whose ith coordinate is null and deleting this ith coordinate.

The *extended code* of a code C over an additive group is the set of vectors, say, $(c_0, c_1, \dots, c_n)$, where $(c_1, \dots, c_n) \in C$ and $c_0 = -(c_1 + \cdots + c_n)$. Note that the extended code of C equals the intersection of the code $\{(c_0, c_1, \dots, c_n) \in \mathbb{F}_q;\ (c_1, \dots, c_n) \in C\}$ and of the parity code $(c_0, c_1, \dots, c_n) \in \mathbb{F}_q;\ \sum_{i=0}^{n} x_i = 0\}$.

1.2.2 Parameters of a code

Sending words of length n over the channel instead of words of length k slows down the transmission of information in the ratio of $\frac{k}{n}$. This ratio, called the *transmission rate*, must be as high as possible, for a given correction capacity, to make possible fast communication. As we see, the three important parameters of a code C are n, k, d (or equivalently $n, |C|, d$ since if q is the alphabet's size, we have $|C| = q^k$), and the first aim[9] of algebraic coding is to find codes minimizing n, maximizing k, and maximizing d, for diverse ranges of parameters corresponding to the needs of communication (see tables of best-known codes in [570]). It is easily seen that $k \leq n - d + 1$ (this inequality, valid for any code over any alphabet, is called the *Singleton bound*) since erasing the coordinates of all codewords at $d-1$ fixed positions gives a set of q^k distinct vectors of length $n - d + 1$, where q is the size of the alphabet, and the number of all vectors of length $n - d + 1$ equals q^{n-d+1}. Codes achieving the Singleton bound with equality are called *maximum distance separable* (MDS). In the case of binary linear codes (see below), it can be shown by using the Pless identities (see, *e.g.*, [348]) that MDS codes have dimension at most 1 or at least $n-1$ and, except for such codes, the bound becomes then $k \leq n - d$.

Another important parameter is the *covering radius*, which is the smallest integer ρ such that the spheres of (Hamming) radius ρ centered at the codewords cover the whole space. In

[9] The second aim is to find decoding algorithms for the codes found.

other words, it is the minimal integer ρ such that every vector of length n lies at Hamming distance at most ρ from at least one codeword, that is, the maximum number of errors to be corrected when maximum likelihood decoding (see page 6) is used. The book [375] is devoted to its study.

The sphere-covering bound is the lower bound on the covering radius ρ, which expresses that, by definition, the balls $B(x,\rho) = \{y \in \mathcal{A}^n; d_H(x,y) \le \rho\}$, of radius ρ and centered in codewords, cover the whole space $\mathcal{A}^n$:

$$|C| \sum_{i=0}^{\rho} \binom{n}{i} (q-1)^i \ge q^n.$$

1.2.3 Linear codes

The general class of linear codes gives a simple and wide example of codes and how they can be used in error correction.

Definition 1 *A code is called a* linear code *if its alphabet is a finite field $\mathbb{F}_q$ (where q is the power of a prime) and if it has the structure of an $\mathbb{F}_q$-linear subspace of $\mathbb{F}_q^n$, where n is its length (see [809]).*

A code that is not necessarily linear is called an *unrestricted code*. The minimum distance of a linear code equals the minimum Hamming weight of all nonzero codewords, since the Hamming distance between two vectors equals the Hamming weight of their difference. We shall write that a linear code[10] over $\mathbb{F}_q$ is an $[n,k,d]_q$-*code* (and if the value of q is clear from the context, an $[n,k,d]$-*code*) if it has length n, dimension k, and minimum distance d. The translates of a linear code are called its *coset*s and the elements of minimum Hamming weights in these cosets are called *coset leader*s (there may exist several in some cosets).

Generator matrix

Any linear code can be described by a *generator matrix* G, obtained by choosing a basis of this vector space and writing its elements as the rows of this matrix. The code equals the set of all the vectors of the form $u \times G$, where u ranges over $\mathbb{F}_q^k$ (and $\times$ is the matrix product) and a possible encoding algorithm is therefore the mapping $u \in \mathbb{F}_q^k \mapsto u \times G \in \mathbb{F}_q^n$. When the codeword corresponding to a given source vector u is obtained by inserting so-called *parity check coordinates* in the source vector (whose coordinates are then called *information coordinates*), the code is called *systematic* (it equals then the *graph* $\{(x, F(x), x \in \mathbb{F}_q^k\}$ of a function, up to coordinate permutation). The corresponding generator matrix is then called a *systematic generator matrix* and has the form $[I_k : M]$, where I_k is the $k \times k$ identity matrix, up to column permutation. It is easily seen that every linear code has such a generator matrix: any generator matrix (of rank k) has k linearly independent columns, and if we place these columns at the k first positions, we obtain $G = [A : M]$, where A is a nonsingular $k \times k$ matrix; then $A^{-1} \times G = [I_k : A^{-1} \times M]$ is a systematic generator matrix of the permuted

[10] The square brackets around n, k, d specify that the code is linear, contrary to standard parentheses.

code (since the multiplication by A^{-1} transforms a basis of the permuted code into another basis of the permuted code).

Dual code and parity check matrix

The generator matrix is well suited for generating the codewords, but it is not for checking if a received word of length n is a codeword or not. A characterization of the codewords is obtained thanks to the generator matrix H of the *dual code* $C^\perp = \{x \in \mathbb{F}_q^n;\ \forall y \in C,\ x \cdot y = \sum_{i=1}^n x_i\, y_i = 0\}$ (such a matrix is called a *parity check matrix* and "$\cdot$" is called the *usual inner product*, or scalar product, in $\mathbb{F}_q^n$): we have $x \in C$ if and only if $x \times H^t$ is the null vector. Consequently, the minimum distance of any linear code equals the minimum number of $\mathbb{F}_q$-linearly dependent columns in one of its parity check matrices (any one). For instance, the binary *Hamming code* of length $n = 2^m - 1$, which has by definition for parity check matrix the $m \times (2^m - 1)$ binary matrix whose columns are all the nonzero vectors of $\mathbb{F}_2^m$ in some order, has minimum distance 3. This code, which by definition is unique up to equivalence, has played an important historical role since it is the first perfect code found. It still plays a role since many computers use it to detect errors in their internal communications. It is the basis on which BCH and Reed–Muller codes were built (see pages 10 and 151). It depends on the choice of the order, but we say that two codes over $\mathbb{F}_q$ are *equivalent codes* if they are equal, up to some permutation of the coordinates of their codewords (and, for nonbinary codes, to the multiplication of each coordinate in each codeword by a nonzero element of $\mathbb{F}_q$ depending only on the position of this coordinate). Note that such codes have the same parameters.

The dual of the binary Hamming code is called the *simplex code*. A generator matrix of this code being the parity check matrix of the Hamming code described above, and the rows of this matrix representing then the *coordinate functions* in $\mathbb{F}_2^m$ (sometimes called dictator functions), on which the order chosen for listing the values is given by the columns of the matrix, the codewords of the simplex code are the lists of values taken on $\mathbb{F}_2^m \setminus \{0_m\}$ by all linear functions.

Note that the dual of a linear code C permuted by some bijection over the indices equals $C^\perp$ permuted by this same bijection, and that, if $G = [I_k : M]$ is a systematic generator matrix of a linear code C, then $[-M^t : I_{n-k}]$ is a parity check matrix of C, where M^t is the transposed matrix of M.

The linear codes that are supplementary with their duals (or equivalently that have trivial intersection with their duals since the dimensions of a code and of its dual are complementary to n) are called complementary dual codes (LCD) and will play an important role in Subsection 12.1.5.

The advantages of linearity

Linearity allows considerably simplifying some main issues about codes. Firstly, the minimum distance being equal to the minimum nonzero Hamming weight, computing it (if it cannot be determined mathematically) needs only to visit $q^k - 1$ codewords instead of $\frac{q^k(q^k-1)}{2}$ pairs of codewords. Secondly, the knowledge of the code is provided by a $k \times n$ generator matrix and needs then the description of k codewords instead of all q^k codewords.

Thirdly, a general decoding algorithm is valid for every linear code. This algorithm is not efficient in general, but it gives a framework for the efficient decoding algorithms that will have to be found for each class of linear codes. The principle of this algorithm is as follows: let y be the (known) received vector corresponding to the (unknown) sent codeword x. We assume that there has been at most $d-1$ errors of transmission, where d is the minimum distance, if the code is used for error detection, and at most e errors of transmission, where $e = \lfloor (d-1)/2 \rfloor$ is the correcting capacity of the code, if the code is used for error correction. The error detection is made by checking if the so-called *syndrome* $s = y \times H^t$ is the zero vector. If it is not, then denoting by ϵ the so-called (unknown) *error vector* $\epsilon = y - x$, correcting the errors of transmission is equivalent to determining ϵ. This can be done by visiting all vectors z of Hamming weight at most e in $\mathbb{F}_q^n$ and checking if $z \times H^t = s$ (indeed, by linearity of matrix multiplication, the syndrome of the error vector equals the syndrome of the received vector, which is known). There exists a unique z of Hamming weight at most e in $\mathbb{F}_q^n$ such that $z \times H^t = s$; this unique z equals ϵ.

Concatenating codes

Given an $\mathbb{F}_q$-linear $[n,k,d]$ code C (where n is the length, k is the dimension, and d is the minimum distance), where $q = 2^e$, $e \geq 2$, a binary $[n', e, d']$ code C' and an $\mathbb{F}_2$-isomorphism $\phi : \mathbb{F}_q \mapsto C'$, the *concatenated code* C'' equals the $[nn', ke, d'' \geq dd']$ binary code $\{(\phi(c_1), \ldots, \phi(c_n));\ (c_1, \ldots, c_n) \in C\}$. Codes C and C' are respectively called outer code and inner code for this construction.

MDS linear codes

Let C be an $[n,k,d]$ code over a field K, let H be its parity check matrix, and G its generator matrix. Then $n-k$ is the rank of H, and we have then $d \leq n - k + 1$ since $n-k+1$ columns of H are always linearly dependent and therefore any set of indices of size $n-k+1$ contains the support of a nonzero codeword. This proves again the Singleton bound: $d \leq n-k+1$.

Recall that C is called *MDS* if $d = n-k+1$. The following are the properties of MDS linear codes:

1. C is MDS if and only if each set of $n-k$ columns of H has rank $n-k$.
2. If C is MDS, then $C^\perp$ is MDS.
3. C is MDS if and only if each set of k columns of G has rank k (and their positions constitute then an information set; see page 161).

Other properties of linear codes

Puncturing, shortening, and extending codes preserve their linearity. Puncturing preserves the MDS property (if $n > k$).

The following lemma will be useful when dealing with Reed–Muller codes in Chapter 4.

Lemma 1 *Let C be a linear code of length n over $\mathbb{F}_q$ and $\hat{C}$ its extended code. We have* $\hat{C}^\perp = \{(y_0, \ldots, y_n) \in \mathbb{F}_q^{n+1};\ (y_1 - y_0, \ldots, y_n - y_0) \in C^\perp\}$.

Proof We have $\hat{C}^\perp = \{(y_0,\dots,y_n) \in \mathbb{F}_q^{n+1};\ \forall (x_1,\dots,x_n) \in C,\ y_0(-\sum_{i=1}^n x_i) + \sum_{i=1}^n x_i y_i = 0\} = \{(y_0,\dots,y_n) \in \mathbb{F}_q^{n+1};\ \forall (x_1,\dots,x_n) \in C,\ \sum_{i=1}^n x_i(y_i - y_0) = 0\} = \{(y_0,\dots,y_n) \in \mathbb{F}_q^{n+1};\ (y_1 - y_0,\dots,y_n - y_0) \in C^\perp\}$. □

Uniformly packed codes: These codes will play a role with respect to almost perfect nonlinear (APN) functions, at page 381.

Definition 2 *[50] Let C be any binary code of length N, with minimum distance $d = 2e+1$ and covering radius ρ. For any $x \in \mathbb{F}_2^N$, let us denote by $\zeta_j(x)$ the number of codewords of C at distance j from x. The code C is called a* uniformly packed code*, if there exist real numbers $h_0,\ h_1,\dots,\ h_\rho$ such that, for any $x \in \mathbb{F}_2^N$, the following equality holds:*

$$\sum_{j=0}^{\rho} h_j\ \zeta_j(x) = 1.$$

As shown in [51], this is equivalent to saying that the covering radius of the code equals its external distance (*i.e.*, the number of different nonzero distances between the codewords of its dual).

1.2.4 Cyclic codes

Two-error correcting Bose–Chaudhuri–Hocquenghem (BCH) codes

The binary Hamming code of length $n = 2^m - 1$ has dimension $n - m$ and needs m parity check bits for being able to correct 1 error. It happens that 2-error binary correcting codes can be built with $2m$ parity check bits. Let us denote by $W_1,\dots,W_n$ the nonzero binary vectors of length m written as columns in some order. The parity check matrix of the Hamming code of length $n = 2^m - 1$ is as follows:

$$H = [W_1,\dots,W_n].$$

To find a 2-error correcting code C of the same length, we consider the codes whose parity check matrices H' are the $2m \times n$ matrices whose m first rows are those of H. These codes being subcodes of the binary Hamming code, they are at least 1-error correcting. For each such matrix H', there exists a function F from $\mathbb{F}_2^m$ to itself such that:

$$H' = \begin{bmatrix} W_1 & W_2 & \dots & W_n \\ F(W_1) & F(W_2) & \dots & F(W_n) \end{bmatrix}.$$

Note that, when F is a *permutation* (*i.e.*, is bijective), the code of generator matrix H' is a so-called *double simplex code* (and plays a central role in [136]); it is the direct sum of two simplex codes: the standard one and its permutation by F.

Going back to general F, assume that two errors are made in the transmission of a codeword of C, at indices $i \neq j$. The *syndrome* of the received vector equals that of the error vector, that is,

$$\begin{bmatrix} S_1 \\ S_2 \end{bmatrix} = \begin{bmatrix} W_i \\ F(W_i) \end{bmatrix} + \begin{bmatrix} W_j \\ F(W_j) \end{bmatrix},$$

with $S_1 \neq 0_m$ (where 0_m is the length m all-zero vector) since $i \neq j$. We have then the following:

$$\begin{cases} W_i + W_j &= S_1 \neq 0_m \\ F(W_i) + F(W_j) &= \quad S_2 \quad . \end{cases}$$

The code is then 2-error correcting if and only if, for every $S_1, S_2 \in \mathbb{F}_2^m$ such that $S_1 \neq 0_m$, this system of equations has either no solution (i, j) (which happens when $\begin{bmatrix} S_1 \\ S_2 \end{bmatrix}$ is not the syndrome of an error vector of Hamming weight 2) or only two solutions (one solution if we impose $i < j$).

Note that since $\{W_1, \ldots, W_n\}$ equals $\mathbb{F}_2^m \setminus \{0_m\}$ and these vectors are all distinct, it is equivalent to consider the system

$$\begin{cases} x + y &= S_1 \neq 0_m \\ F(x) + F(y) &= \quad S_2 \quad , \end{cases}$$

where x and y range over $\mathbb{F}_2^m \setminus \{0_m\}$. This is where finite fields of orders larger than 2 played a historical role in coding theory (see Appendix, page 480, for a description of finite fields): considering such functions F and such systems of equations is easier when we have a structure of field (even though the equations do not involve multiplications). This allows us indeed to take $F(x)$ in a polynomial form, and the first polynomials to be tried are of course monomials. The monomials x and x^2, being linear functions, do not satisfy the condition needed for the code to be 2-error correcting, but the next monomial x^3 does satisfy it (this is easily seen since $x^3 + y^3 = (x+y)^3 + x\,y\,(x+y)$ implies that the system is equivalent to $\begin{cases} x + y = S_1 \neq 0 \\ \;\; x\,y \;\;\; = \frac{S_2 + S_1^3}{S_1} \end{cases}$ and such an equation results in an equation of degree 2, which has at most two solutions over a finite field).

The condition on F (or more precisely on its extension by taking $F(0) = 0$) is equivalent to saying that it is an APN function. This notion plays a very important role in cryptography; see Chapter 11, page 369.

We need here the notion of *primitive element*; see page 487. Such element α satisfies that $\mathbb{F}_{2^n} = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^n-2}\}$ and exists for every n.

Definition 3 *Let α be a primitive element of $\mathbb{F}_{2^m}$. The binary 2-error correcting BCH code of length $n = 2^m - 1$ is the $[n, n-2m, 5]$ code due to Bose, Chaudhuri, and Hocquenghem, of the following parity check matrix:*

$$H' = \begin{bmatrix} \alpha & \alpha^2 & \ldots & \alpha^n \\ \alpha^3 & \alpha^6 & \ldots & \alpha^{3n} \end{bmatrix}.$$

Ordering the elements of $\mathbb{F}_{2^n}^*$ as $\alpha, \alpha^2, \ldots, \alpha^{n-1}, \alpha^n = 1$ (we could have also chosen $1, \alpha, \alpha^2, \ldots, \alpha^{n-1}$) implies a property that does not seem so important at first glance but which played a central role in the history of codes and still plays such role nowadays: the code is (globally) invariant under cyclic permutations of the codeword coordinates. This property, when added to the linearity of the code, confers to them a structure of principal ideal, with very nice theoretical and practical consequences.

General cyclic codes

A linear code C of length n is a *cyclic code* if it is (globally) invariant under cyclic shifts of the codeword coordinates (see [809, page 188]). For this, it is enough that it is invariant under one of the primitive cyclic shifts, for instance:

$$(c_0, \dots, c_{n-1}) \mapsto (c_{n-1}, c_0, \dots, c_{n-2}).$$

Cyclic codes have been extensively studied in coding theory, because of their strong properties.

Representation of codewords

Each codeword $(c_0, \dots, c_{n-1})$ is represented by the polynomial $c_0 + c_1X + \dots + c_{n-1}X^{n-1}$, viewed as an element of the quotient algebra $A = \mathbb{F}_q[X]/(X^n - 1)$ (each element of this algebra is an equivalence class modulo $X^n - 1$, which will be always represented by its unique element of degree at most $n - 1$, equal to the common rest in the division by $X^n - 1$ of the polynomials constituting the class). We shall call $c_0 + c_1X + \dots + c_{n-1}X^{n-1}$ the *polynomial representation of codeword* $(c_0, \dots, c_{n-1})$. Then it is easily shown that C is cyclic if and only if it is an ideal of $\mathbb{F}_q[X]/(X^n - 1)$, that is, it satisfies $fC \subseteq C$ for every nonzero $f \in A$ (C being assumed linear, it is a subgroup of A).

Generator polynomial

The algebra $\mathbb{F}_q[X]/(X^n - 1)$ is a principal domain. It is easily shown that any (linear) cyclic nontrivial[11] code has a unique monic element $g(X)$ (whose leading coefficient equals 1) having minimal degree, which generates the ideal and is called the *generator polynomial* of the code. In fact, $g(X)$ is a generator of the code in the strong sense that every polynomial of degree at most $n - 1$ is a codeword if and only if it is a multiple of $g(X)$ in $\mathbb{F}_q[X]$ (which implies that it is a multiple of $g(X)$ in $\mathbb{F}_q[X]/(X^n - 1)$). The code equals then the set of all those polynomials that include the zeros of $g(X)$ (in the splitting field of $g(X)$) among their own zeros. It is also easily seen that $g(X)$ is a divisor of $X^n - 1$.

Zeros of the code

In our framework, the length will have the form $n = q^m - 1$ (we call such length a *primitive length*). In such a case, since $g(X)$ divides $X^n - 1$, the zeros of $g(X)$ all belong to $\mathbb{F}_{q^m}^*$. The generator polynomial having all its coefficients in $\mathbb{F}_q$, its zeros are of the form $\{\alpha^i, i \in I\}$ (where α is a primitive element of $\mathbb{F}_{q^m}$), where $I \subseteq \mathbb{Z}/n\mathbb{Z}$ is a union of cyclotomic classes of q modulo $n = q^m - 1$ (and vice versa). The set I is called the *defining set* of the code. The elements $\alpha^i, i \in I$ are called the *zeros of the cyclic code*, which has dimension $n - |I|$. The elements $\alpha^i, i \in \mathbb{Z}/n\mathbb{Z} \setminus I$ are called the *nonzeros of the cyclic code*. The generator polynomial of $C^\perp$ is the reciprocal of the quotient of $X^n - 1$ by $g(X)$, and its defining set therefore equals $\{n - i; i \in \mathbb{Z}/n\mathbb{Z} \setminus I\}$.

[11] That is, it is different from $\{0_n\}$; in fact, we shall consider that the trivial cyclic code has also a generator polynomial: $X^n - 1$ itself.

McEliece's theorem [833] states that a binary cyclic code is exactly 2^l-divisible (that is, l is the maximum such that all codeword Hamming weights are divisible by 2^l) if and only if l is the smallest number such that $l+1$ nonzeros of C (with repetitions allowed) have product 1 (and recall that $\alpha^j = 1$ if and only if $2^n - 1$ divides j).

Generating all cyclic codes of some primitive length

Since a polynomial over $\mathbb{F}_q$ is the generator polynomial of a cyclic code of length n if and only if it divides $X^n - 1$, we obtain all cyclic codes from all the divisors of $X^n - 1$ in $\mathbb{F}_q$. Any such divisor is the product of some irreducible factors of $X^n - 1$ in $\mathbb{F}_q$. These irreducible factors are the polynomials of the form $\prod_{j \in \mathcal{C}} (X - \alpha^j)$, where $\mathcal{C}$ is a cyclotomic class of q modulo n. The number of cyclic codes of length n over $\mathbb{F}_q$ is then 2^r, where r is the number of these cyclotomic classes (including the trivial cyclic code $\{0_n\}$ and the full one $\mathbb{F}_q^n$). The *Hamming code* has for generator polynomial the irreducible polynomial corresponding to the cyclotomic class containing 1. Its dual, the *simplex code*, has then for generator polynomial the polynomial corresponding to all cyclotomic classes except that of $n-1$.

Nonprimitive length

If the length is not primitive, the zeros of $X^n - 1$ live in its *splitting field* $\mathbb{F}_{q^m}$ (where n divides $q^m - 1$, and m is minimal). If n and q are coprime, the zeros of $X^n - 1$ are simple since the derivative nX^{n-1} of this polynomial does not vanish on them, and the same theory applies by replacing $\mathbb{F}_{q^m}$ by the group of nth roots of unity in $\mathbb{F}_{q^m}$ and α by a primitive nth root of unity.

BCH bound

A very efficient bound on the minimum distance of cyclic codes is the *BCH bound* [809, page 201]: if I contains a "string" $\{l+1, \ldots, l+\delta-1\}$ of length $\delta - 1$ of consecutive[12] elements of $\mathbb{Z}/n\mathbb{Z}$, then the cyclic code has minimum distance larger than or equal to δ (which is then called the *designed distance* of the cyclic code). A proof of this bound (in the framework of Boolean functions) is given in the proof of Theorem 23, page 337.

BCH codes

Let n be coprime with q and $\delta < n$, the BCH codes of length n and designed distance δ are the cyclic codes that have such string of length $\delta - 1$ in their zeros (and have then minimum distance at least δ, according to the BCH bound) and maximal dimension (*i.e.*, minimal number of zeros) with such constraint.

Reed–Solomon codes

When $n = q - 1$, the cyclotomic classes of q modulo n are singletons and the set of zeros of a cyclic code can then be any set of nonzero elements of the field (the generator polynomial

[12] Considering of course that 0 is the successor of $n-1$ in $\mathbb{Z}/n\mathbb{Z}$.

can be any divisor of $X^n - 1$); when it is constituted of consecutive powers of a primitive element, this particular case of a BCH code is called a *Reed–Solomon (RS) code*. Such codes are important because they achieve the Singleton bound with equality (*i.e.*, they are *maximum distance separable MDS*). Indeed, the BCH bound gives $\delta \le d \le n - (n - (\delta - 1)) + 1 = \delta$, and the Singleton bound is then achieved with equality.

Remark. There exists another equivalent definition of Reed–Solomon codes; see the remark on page 45. RS codes are widely used in consumer electronics (CD, DVD, Blu-ray), data transmission technologies (DSL, WiMAX), broadcast systems, computer applications, and deep-space communications. □

Extended Reed–Solomon codes

A cyclic code C of length n being given, recall that the extended code of C is the set of vectors $(c_\infty, c_0, \dots, c_{n-1})$, where $c_\infty = -(c_0 + \dots + c_{n-1})$. It is a linear code of length $n+1$ and of the same dimension as C. When C is a Reed–Solomon code whose defining set has the form $\{1, 2, \dots, \delta - 1\}$, its extended code is also MDS, because when $(c_0, \dots, c_{n-1})$ is a codeword of C of minimal Hamming weight δ, we have $c_\infty \neq 0$ (again according to the BCH bound: if $c_\infty = 0$, then the polynomial $c_0 + c_1X + \dots + c_{n-1}X^{n-1}$ has also $\alpha^0 = 1$ for zero and has then Hamming weight at least $\delta + 1$, thanks to the BCH bound applied with the string $\{0, \dots, \delta - 1\}$). Hence, either $(c_0, \dots, c_{n-1})$ is a codeword of C of minimal Hamming weight δ and then $(c_\infty, c_0, \dots, c_{n-1})$ has Hamming weight $\delta + 1$ or $(c_0, \dots, c_{n-1})$ has Hamming weight at least $\delta + 1$ and $(c_\infty, c_0, \dots, c_{n-1})$ has a fortiori Hamming weight at least $\delta + 1$. Hence the minimum distance of the extended code is $\delta + 1 = (n + 1) - (n - \delta + 1) + 1$. The extended code is MDS.

Cyclic codes and Boolean functions

Cyclic codes over $\mathbb{F}_2$ and of length $2^m - 1$ can be viewed as sets of m-variable Boolean functions. Indeed, any codeword in such cyclic code with defining set I can be represented in the form $\sum_{i=1}^{l} tr_n(a_i x^{-u_i}), a_i \in \mathbb{F}_{2^m}$, where $u_1, \dots, u_l$ are representatives of the cyclotomic classes lying outside I (see Relation (2.20) in Subsection 2.2.2, page 45).

1.2.5 The MacWilliams identity and the notion of dual distance

Linear codes

A nice relationship, due to F. J. MacWilliams [809, page 127], exists between the Hamming weights in every binary linear code[13] and those in its dual: let C be any binary linear code of length n; consider the polynomial $W_C(X, Y) = \sum_{i=0}^{n} A_i X^{n-i} Y^i$, where A_i is the number of codewords of Hamming weight i. This polynomial is called the *weight enumerator* of C and describes[14] the *weight distribution* $(A_i)_{0 \le i \le n}$ of C. Then

$$W_C(X + Y, X - Y) = |C|\, W_{C^\perp}(X, Y). \qquad (1.1)$$

[13] It exists for every linear code over a finite field and even for more general codes, but we shall need it only for binary codes.

[14] W_C is a homogeneous version of classical generating series for the weight distribution of C.

We give a sketch of proof[15] of this *MacWilliams' identity*: we observe first that $W_C(X,Y) = \sum_{x\in C}\prod_{i=1}^n X^{1-x_i}Y^{x_i}$; substituting X by $X+Y$ and Y by $X-Y$, we deduce that $W_C(X+Y, X-Y) = \sum_{x\in C}\prod_{i=1}^n (X+(-1)^{x_i}Y)$. We apply then the classical relation making possible to expand products of sums: for every $\lambda_1,\dots,\lambda_n,\mu_1,\dots,\mu_n$, we have $\prod_{i=1}^n(\lambda_i+\mu_i) = \sum_{b\in\mathbb{F}_2^n}\prod_{i=1}^n(\lambda_i^{1-b_i}\mu_i^{b_i})$ (indeed, choosing λ_i in the ith factor when $b_i = 0$ and μ_i when $b_i = 1$ provides when b ranges over $\mathbb{F}_2^n$ all the possible terms in the expansion). This gives here $W_C(X+Y,X-Y) = \sum_{x\in C}\sum_{b\in\mathbb{F}_2^n}\prod_{i=1}^n\left(X^{1-b_i}((-1)^{x_i}Y)^{b_i}\right)$. We obtain then $W_C(X+Y,X-Y) = \sum_{b\in\mathbb{F}_2^n}\left(X^{n-w_H(b)}Y^{w_H(b)}\sum_{x\in C}(-1)^{b\cdot x}\right)$, where "$\cdot$" is the *usual inner product* in $\mathbb{F}_2^n$, and we conclude by observing that, if $b\notin C^\perp$, then the linear form $b\cdot x$ over the vector space C is nonzero, and takes then values 0 and 1 on two complementary hyperplanes, that is, the same number of times (we will find again this in Relation (2.38), page 58). This proves Relation (1.1). Of course, we deduce that $W_C(X,Y) = \frac{1}{|C^\perp|}W_{C^\perp}(X+Y,X-Y)$ and the same method shows, as observed in [37], that for every coset $a+C$, we have $W_{a+C}(X,Y) = \frac{1}{|C^\perp|}\left(2\,W_{C^\perp\cap\{0_n,a\}^\perp}(X+Y,X-Y) - W_{C^\perp}(X+Y,X-Y)\right)$.

Remark. We have $|C| = \sum_{i=0}^n A_i = W_C(1,1)$. The fact that the polynomial $\frac{1}{W_C(1,1)}W_C(X+Y,X-Y)$ has nonnegative integer coefficients is very specific (among all homogeneous polynomials $P(X,Y)$ whose coefficients are nonnegative integers). As far as we know, the characterization of all homogeneous polynomials $P(X,Y)$ over $\mathbb{N}$ such that $\frac{1}{P(1,1)}P(X+Y,X-Y)$ has nonnegative integer coefficients has never been investigated in a paper. $\square$

Remark. The average Hamming weight of the codewords of a linear binary code C equals $(W_C)'_Y(1,1)$ (the value at $(1,1)$ of the partial derivative of $W_C(X,Y)$ with respect to Y), divided by $|C|$. MacWilliams' identity writes $W_C(X,Y) = \frac{1}{|C^\perp|}W_{C^\perp}(X+Y,X-Y)$. Differentiating with respect to Y gives $(W_C)'_Y(X,Y) = \frac{1}{|C^\perp|}(W_{C^\perp})'_X(X+Y,X-Y) - \frac{1}{|C^\perp|}(W_{C^\perp})'_Y(X+Y,X-Y)$ and thus $(W_C)'_Y(1,1) = \frac{1}{|C^\perp|}(W_{C^\perp})'_X(2,0) - \frac{1}{|C^\perp|}(W_{C^\perp})'_Y(2,0) = \frac{n2^{n-1}}{|C^\perp|} - \frac{1}{|C^\perp|}(W_{C^\perp})'_Y(2,0)$, and the average Hamming weight of codewords equals $\frac{n}{2} - 2^{-n}(W_{C^\perp})'_Y(2,0)$, which depends on the number of words of Hamming weight 1 in $C^\perp$ (see more in [809, page 131] on the moments of the weight distribution of codes) and is bounded above by $\frac{n}{2}$. In fact, it is easily seen directly that the average Hamming weight of codewords equals $\frac{n-r}{2}$, where r is the number of positions where all codewords are null, since if there is a codeword with 1 at position i, the average value of codewords at position i equals $\frac{1}{2}$. $\square$

Remark. Some authors call weight enumerator of C the univariate polynomial $A_C(Z) = \sum_{i=0}^n A_iZ^i$. MacWilliams' identity writes then $(1+Z)^nA_C\left(\frac{1-Z}{1+Z}\right) = |C|\,W_{C^\perp}(Z)$, where n is the length of the binary code C. $\square$

[15] The classical proof uses Fourier–Hadamard transform; since this transform will be addressed later in this book, in Section 2.3, we give a proof more coding theory oriented.

The MacWilliams identity gives information on self-dual codes (*i.e.*, codes equal to their duals) through the *Gleason theorem*, which says that the weight enumerator of a self-dual code is in the ring generated by $X^2 + Y^2$ and $XY - Y^2$ (see [809, page 602]).

Unrestricted codes

The principle of MacWilliams' identity can also be applied to unrestricted codes. When C is not linear, the weight distribution of C has no great relevance. The distance distribution has more interest. We consider the *distance enumerator* of C:

$$D_C(X,Y) = \frac{1}{|C|}\sum_{i=0}^{n} B_i X^{n-i} Y^i,$$

where B_i is the size of the set $\{(x,y) \in C^2;\ d_H(x,y) = i\}$. Note that, if C is linear, then $D_C = W_C$. Similarly as above, we see the following:

$$D_C(X,Y) = \frac{1}{|C|}\sum_{(x,y)\in C^2}\prod_{i=1}^{n} X^{1-(x_i\oplus y_i)} Y^{x_i\oplus y_i};$$

we deduce as follows:

$$D_C(X+Y, X-Y) = \frac{1}{|C|}\sum_{(x,y)\in C^2}\prod_{i=1}^{n}(X + (-1)^{x_i\oplus y_i} Y).$$

Expanding these products by the same method as above, we obtain the following:

$$D_C(X+Y, X-Y) = \frac{1}{|C|}\sum_{(x,y)\in C^2}\sum_{b\in\mathbb{F}_2^n}\prod_{i=1}^{n}\left(X^{1-b_i}((-1)^{x_i\oplus y_i} Y)^{b_i}\right);$$

that is,

$$D_C(X+Y, X-Y) = \frac{1}{|C|}\sum_{b\in\mathbb{F}_2^n} X^{n-w_H(b)} Y^{w_H(b)}\left(\sum_{x\in C}(-1)^{b\cdot x}\right)^2. \qquad (1.2)$$

Hence, $D_C(X+Y, X-Y)$ has nonnegative coefficients (but $D_C(X,Y)$ is not necessarily the weight enumerator of a code; note, however, that it is one in the case of distance-invariant codes, such as Kerdock codes; see Section 6.1.22).

Definition 4 *The smallest nonzero exponent of Y with nonzero coefficient in the polynomial $D_C(X+Y, X-Y)$, that is, the number*

$$\min\left\{w_H(b);\ b \neq 0_n, \sum_{x\in C}(-1)^{b\cdot x} \neq 0\right\},$$

often denoted by $d^\perp(C)$, is called the dual distance *of C.*

The dual distance of C is strictly larger than an integer t if and only if the restriction to C of any sum of at least one and at most t coordinate functions in $\mathbb{F}_2^n$ is *balanced* (*i.e.*, has uniform distribution), that is, any of the punctured codes of length t of C equals the whole vector space $\mathbb{F}_2^t$ and each vector in $\mathbb{F}_2^t$ is matched the same number of times.[16] Hence, as we shall see again at page 88, the size of a code of dual distance d is divisible by 2^{d-1}; note that for linear codes, this tells more than the Singleton bound applied to the dual.

This notion will play an important role with Boolean functions (see Definition 21, page 86; this is why we include Lemma 2 below) and with a recent kind of cryptanalysis that plays an important role nowadays: side channel attacks (see Section 12.1, page 425).

Lemma 2 *1. Any coset $a + C$ of a binary unrestricted code has the same dual distance as C. Any union of cosets of a linear code C has at least the same dual distance as C.*

2. *The dual distance of a punctured code is larger than or equal to the dual distance of the original code (assuming that the latter has minimum distance at least 2).*
3. *The dual distance of the Cartesian product of two binary unrestricted codes equals the minimum of their dual distances.*
4. *Let C_1 and C_2 be binary unrestricted codes of the same length n and*

$$C'' = \{(c_1, c_1 + c_2); c_1 \in C_1, c_2 \in C_2\},$$

then $d''^{\perp} = \min(d_1^{\perp}, 2\, d_2^{\perp})$.

The proof of this lemma is also an easy consequence of the properties of the Fourier–Hadamard transform that we shall see in Section 2.3.

Remark. When C is linear, $d^{\perp}$ equals the minimum distance of the *dual code* $C^{\perp}$. Hence, since the minimum distance of a linear code over $\mathbb{F}_q$ equals the minimum nonzero number of $\mathbb{F}_q$-linearly dependent columns in its parity check matrix, its dual distance equals the minimum nonzero number of $\mathbb{F}_q$-linearly dependent columns in its generator matrix. $\square$

1.3 Boolean functions

We call *Boolean functions* (and sometimes we specify *n-variable* Boolean functions or *Boolean functions in dimension n*) the (single-output) functions from the n-dimensional vector space $\mathbb{F}_2^n$ over $\mathbb{F}_2$, to $\mathbb{F}_2$ itself. Their set is denoted by $\mathcal{BF}_n$. Number n will be named the number of variables, or of input bits. More generally,[17] we call n-variable *pseudo-Boolean* functions the functions from $\mathbb{F}_2^n$ to $\mathbb{R}$.

Boolean functions will also be viewed in some cases as taking their input in the field $\mathbb{F}_{2^n}$. Indeed, this field is an n-dimensional vector space over $\mathbb{F}_2$ and it can then be identified with the vector space $\mathbb{F}_2^n$ through the choice of a basis.

Boolean functions play roles in both cryptographic and error-correcting coding activities in *information protection*:

[16] This is a consequence of the properties of the Fourier–Hadamard transform that we shall see in Section 2.3, applied to the indicator of C; see Corollary 6, page 88, and Theorem 5.

[17] When we will consider Boolean functions as particular pseudo-Boolean functions, by viewing their output values 0 and 1 as elements of $\mathbb{Z}$ rather than $\mathbb{F}_2$ (for instance, when defining their numerical normal form in Subsection 2.2.4 or their Fourier–Hadamard transform in Section 2.3), adding their values will be made in $\mathbb{Z}$, with notation $+$; otherwise, it will be made modulo 2, with notation $\oplus$.

Table 1.1 Number of n-variable Boolean functions.

n	4	5	6	7	8
$\lvert\mathcal{BF}_n\rvert$	2^{16}	2^{32}	2^{64}	2^{128}	2^{256}
$\approx$	$6 \cdot 10^4$	$4 \cdot 10^9$	10^{19}	10^{38}	10^{77}

- Every binary unrestricted code of length 2^n, for some positive integer n, can be interpreted as a set of Boolean functions, since every n-variable Boolean function can be represented by its truth table (an ordering of the set of binary vectors of length n being first chosen) and thus associated with a binary word of length 2^n, and vice versa; important codes (Reed–Muller, Kerdock codes; see Sections 4.1 and 6.1.22) can be defined this way as sets of Boolean functions.
- The role of Boolean functions in conventional cryptography is even more important: cryptographic transformations can be designed by appropriate composition of Boolean functions.[18]

In both frameworks, n is rarely large, in practice:

- The error-correcting codes derived from n-variable Boolean functions have length 2^n; so, taking $n = 10$ already gives codes of length 1024.
- For reason of efficiency, the Boolean functions used in stream ciphers had about 10 variables until algebraic attacks were invented in 2003, and the number of variables is now most often limited to at most 20, except when the functions are particularly fast to compute.

Despite their low numbers of variables, the Boolean functions used in cryptography and satisfying the desired conditions (see Section 3.1 below) cannot be determined or studied by an exhaustive computer investigation: the number $|\mathcal{BF}_n| = 2^{2^n}$ of n-variable Boolean functions is too large when $n \geq 6$. We give in Table 1.1 below the values of this number for n ranging between 4 and 8.

Assume that visiting an n-variable Boolean function, and determining whether it has the desired properties, requires one nanosecond (10^{-9} seconds); then it would need millions of hours to visit all functions in six variables, and about 100 billions times the age of the universe to visit all those in seven variables. The number of eight-variable Boolean functions approximately equals the number of atoms in the whole universe! We see that trying to find functions satisfying the desired conditions by simply picking up functions at random is also impossible for these values of n, since visiting a nonnegligible part of all Boolean functions in seven or more variables is not feasible, even when parallelizing. The study of Boolean functions for constructing or studying codes or ciphers is essentially mathematical. But clever computer investigation is very useful to imagine or to test conjectures, and sometimes to generate interesting functions.

[18] Boolean functions play also a role in hash functions, but we shall not develop this aspect, for lack of space, and in the inner protection of some chips.

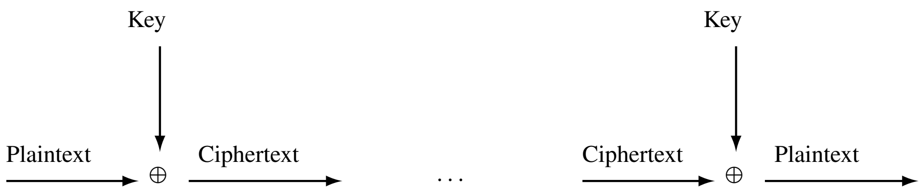


Figure 1.1 Vernam cipher.

Remark. Boolean functions play an important role in computational complexity theory, with the notion of NP-complete decisional problem (where "NP" stands for nondeterministic polynomial time), for which satisfiability problems (in particular, the 3-SAT problem) are central. These problems are related to representations of Boolean functions by disjunctive and conjunctive normal forms, which do not ensure uniqueness and are not much used in cryptography and error-correcting coding. We refer the reader interested in satisfiability problems and in the related complexity theory of Boolean functions to [31, 81, 1117]. □

A nice site under construction at the moment this book is written can be found at the URL http://boolean.h.uib.no/mediawiki.

1.3.1 Boolean functions and stream ciphers

Stream ciphers are based on the so-called *Vernam cipher* (see Figure 1.1) in which the plaintext (a binary string of some length) is bitwise added to a (binary) secret key of the same length, in order to produce the ciphertext. The Vernam cipher is also called the *one time pad* because a new random secret key must be used for every encryption. Indeed, the bitwise addition of two ciphertexts corresponding to the same key equals the addition of the corresponding plaintexts, which gives much information on these plaintexts when they code for instance natural language (it is often enough to recover both plaintexts, even when one of them is reversed; some secret services and spies learned this at their own expense).

The Vernam cipher, which is the only known cipher offering unconditional security (see [1034]) if the key is truly random and if it is changed for every new encryption, was used for the communication between the heads of the USA and the USSR during the cold war (the keys being carried by diplomats) and by some secret services.

In practice (except in the very sensitive situations indicated above), since in the Vernam cipher, the length of the private key must be equal to the length of the plaintext (which is impractical), a so-called *pseudorandom generator* (*PRG*) is used for producing a long *pseudorandom sequence* (the *keystream*, playing the role of the private key in the Vernam cipher) from the short random secret key. Only the latter is actually shared.[19] The unconditional security is then no longer ensured (this is the price to pay for making the cipher lighter). If the keystream only depends on the key (and not on the plaintext), the

[19] The PRG is supposed to be public since taking a part of the secret for describing it would reduce in practice the length of the key.

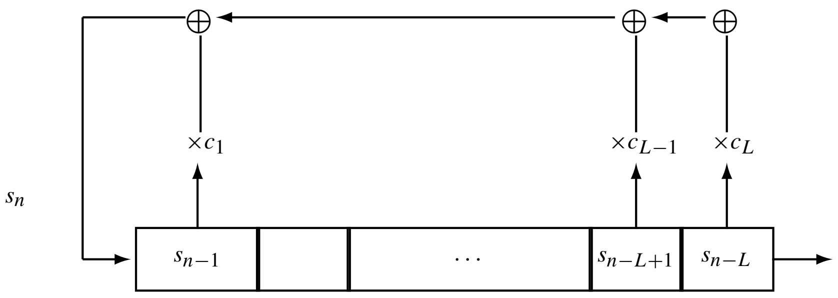


Figure 1.2 LFSR.

cipher is called *synchronous*.[20] Stream ciphers, because they operate on data units as small as a bit or a few bits, are suitable for fast telecommunication applications. Having also a very simple construction, they are easily implemented both in hardware and software. They need to resist all known attacks (see in Section 3.1 those that are known so far). The so-called *attacker model* for these attacks (that is, the description of the knowledge the attacker is supposed to have) is as follows: some knowledge on the plaintext may be unavoidable and it is then assumed that the attacker has access to a small part of it. Since the keystream equals the XOR of the plaintext and the ciphertext, the attacker is then assumed to have access to a part of the keystream, and he/she needs to reconstruct the whole sequence.

A first method for generating pseudorandom sequences from secret keys has used *linear feedback shift register*s (*LFSR*) [550]. In such an LFSR (see Figure 1.2, where $\times$ means multiplication), at every clock cycle, the bits $s_{n-1}, \dots, s_{n-L}$ contained in the flip-flops of the LFSR move to the right. The right-most bit is the current output (a keystream of length N will then be produced after N clock cycles) and the leftmost flip-flop is fed with the linear combination $\bigoplus_{i=1}^{L} c_i s_{n-i}$, where the c_is are bits. Thus, such an LFSR outputs a recurrent sequence satisfying the relation

$$s_n = \bigoplus_{i=1}^{L} c_i s_{n-i}.$$

Such a sequence is always ultimately periodic[21] (if $c_L = 1$, then it is periodic; we shall assume that $c_L = 1$ in the sequel, because otherwise the same sequence can be output by an LFSR of a shorter length, except for its first bits, and this can be exploited in attacks) with period at most $2^L - 1$. The generating series $s(X) = \sum_{i \geq 0} s_i X^i$ of the sequence can be expressed in a nice way (see the chapter by Helleseth and Kumar in [959] and Section 10.2, "LFSR sequences and maximal period sequences", by Niederreiter in [890]): $s(X) = \frac{G(X)}{F(X)}$, where $G(X) = \sum_{i=0}^{L-1} X^i \left(\bigoplus_{j=0}^{i} c_{i-j} s_j \right)$ is a polynomial of degree smaller than L and $F(X) = 1 + c_1 X + \cdots + c_L X^L$ is the *feedback polynomial* (an equivalent representation

[20] There also exist self-synchronizing stream ciphers, in which each keystream bit depends on the n preceding ciphertext bits, which makes possible resynchronizing after n bits if an error of transmission occurs between Alice and Bob.

[21] Conversely, every ultimately periodic sequence can be generated by an LFSR.

uses the *characteristic polynomial*, which is the reciprocal of the feedback polynomial). The minimum length of the LFSR producing a sequence is called the *linear complexity* of the sequence (and sometimes its *linear span*). It equals L if and only if the polynomials F and G above are coprime and is equal in general to $N - deg\,(\gcd(X^N + 1, S(X)))$, where N is a period and $S(X)$ is the generating polynomial $S(X) = s_0 + s_1 X + \cdots + s_{N-1} X^{N-1}$. An *m-sequence* (or *maximum length sequence*) is a sequence of period $2^{\mathcal{L}} - 1$, where $\mathcal{L}$ is the linear complexity. Assuming that $L = \mathcal{L}$, this corresponds to taking a primitive feedback polynomial (see page 488). The sequence can then be represented in the form $s_i = tr_n(a\alpha^i)$, where α is a primitive element of $\mathbb{F}_{2^n}$ (see page 487) and tr_n is the trace function from $\mathbb{F}_{2^n}$ to $\mathbb{F}_2$ (see pages 42 and 489). The m-sequences have very strong properties; see the chapter by Helleseth and Kumar in [959].

The initialization $s_0, \ldots, s_{L-1}$ of the LFSR and the values of the *feedback coefficients* c_i must be kept secret (they are then computed from the secret key); if the feedback coefficients were public, the observation of L consecutive bits of the keystream would allow recovering all the subsequent sequence.

Berlekamp–Massey attack

The use of LFSRs as pseudorandom generators is cryptographically weak because of an attack found in the late 1970s called the *Berlekamp–Massey (BM) algorithm* [826]: let $\mathcal{L}$ be the linear complexity of the sequence, assumed to be unknown from the attacker; if he knows at least $2\mathcal{L}$ consecutive bits of the sequence, the BM algorithm allows him to recover the values of $\mathcal{L}$ and of the feedback coefficients of an LFSR of length $\mathcal{L}$ generating the sequence, as well as the initialization of this LFSR. The BM algorithm has quadratic complexity, that is, works in $\mathcal{O}(\mathcal{L}^2)$ elementary operations. Improvements of the algorithm exist, which have lower complexity: the main idea[22] is to use the extended Euclidean (EE) algorithm (or its variants). The way to use this algorithm is shown in the section "Linearly recurrent sequences" (Section 12.3) of the book *Modern Computer Algebra* by J. von zur Gathen and J. Gerhard [533] (Algorithm 12.9 in this book is essentially an EE algorithm). The complexity of an EE algorithm being $\mathcal{O}(M(\mathcal{L})\log(\mathcal{L}))$, where $M(\mathcal{L})$ is the cost of the multiplication between two polynomials of degree $\mathcal{L}$, and this latter cost being quasilinear, the complexity of finding the retroaction polynomial of an LFSR is roughly $\mathcal{O}(\mathcal{L}\ \log(\mathcal{L}))$. The data complexity is still $2\mathcal{L}$, but these $2\mathcal{L}$ bits of the sequence do not need to be strictly consecutive: having k strings of $2\mathcal{L}/k$ consecutive bits is enough, thanks to a matrix version of the BM algorithm found by Coppersmith, coupled with an algorithm due to Beckerman and Labahn, or with a simpler (and implemented) one due to Thomé; see more in [1085].

The role of Boolean functions

Many keystream generators still use LFSRs, and to resist the Berlekamp–Massey attack, either combine several LFSRs (and possibly some additional memory) as in the case of E_0, the keystream generator that is part of the Bluetooth standard, or use Boolean functions; see [1006]. The first model that appeared in the literature for such use is the *combiner model* (see Figure 1.3).

[22] We thank Pierrick Gaudry for his kind explanations.

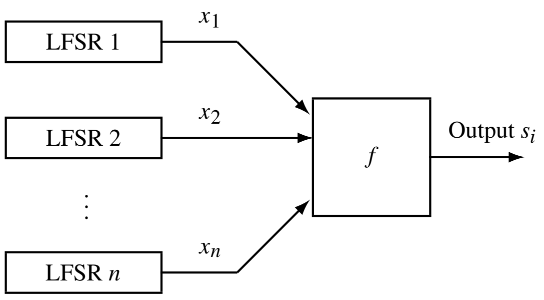


Figure 1.3 Combiner model.

Notice that the feedback coefficients of the n LFSRs used in such a generator can be public. The Boolean function is also public, in general, and the (short) secret key is necessary only for the initialization of the n LFSRs (also depending on an initial vector, which being public can be changed more often than the key): if we want to use for instance a 128-bit-long secret key, this makes possible using n LFSRs of lengths $L_1, \ldots, L_n$ such that $L_1 + \cdots + L_n = 128$.

Such system clearly outputs a periodic sequence whose period is at most the LCM of the periods of the sequences output by the n LFSRs (assuming that $c_L = 1$ in each LFSR; otherwise, the sequence is ultimately periodic and the period is shorter). So, this sequence satisfies a linear recurrence and can therefore be produced by a single LFSR. However, as we shall see, well-chosen Boolean functions allow the linear complexity of the sequence to be much larger than the sum of the lengths of the n LFSRs. Nevertheless, choosing LFSRs producing sequences of large periods, choosing these periods pairwise co-prime in order to have the largest possible global period, and choosing f such that the linear complexity is large enough too are not sufficient. As we shall see, the combining function should also not leak information about the individual LFSRs and behave as differently as possible from affine functions, in several different ways.

The combiner model is only a model, useful for studying attacks and related criteria. In practice, the systems are more complex (see for instance at URL www.ecrypt.eu.org/stream/ to see how the stream ciphers of the *eSTREAM Project* [495] are designed).

A more recent model is the *filter model*, which uses a single LFSR (of a longer length). A filtered LFSR outputs $f(x_1, \ldots, x_n)$, where f is some n-variable Boolean function, called a filtering function, and where $x_1, \ldots, x_n$ are the bits contained in some flip-flops of the LFSR; see Figure 1.4.

Such a system is equivalent to the combiner model using n copies of the LFSR. However, the attacks, even when they apply to both systems, do not work similarly (a first obvious difference is that the lengths of the LFSRs are different in the two models). Consequently, the criteria that the involved Boolean functions must satisfy to allow resistance to these attacks need to be studied for each model (we shall see that they are in practice not so different, except for one criterion that will be necessary for the combiner model but not for the filter model).

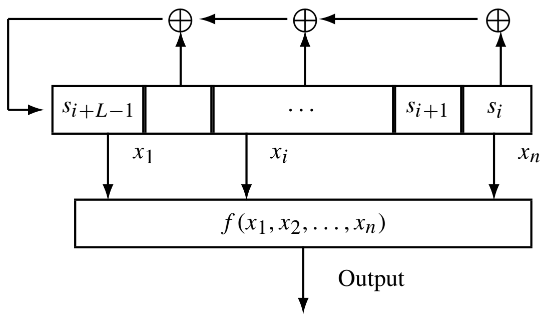


Figure 1.4 Filter model.

Note that in both models, the PRG is made of a *linear part* (constituted by the LFSRs), the linearity allowing speed, and a *nonlinear part* (made of the combiner/filter function) providing confusion (see the meaning of this term in Section 3.1). Generalizations of the two models have been proposed with the same structure "linear part, nonlinear part" [495, 901]. In practice, models will not be used as is; we shall add memory and/or few combinatoric stages and/or initialization registers; a high level security is ensured by the fact that the model, as is, is proved resistant to all known attacks, and the additional complexity will make the work of the attacker still more difficult.

Other kinds of pseudorandom generators exist that are not built on the same principle. A *feedback shift register* (FSR) has the same structure as an LFSR, but the leftmost flip-flop is feeded with $g(x_{i_1}, \ldots, x_{i_n})$, where $n \leq L$ and $x_{i_1}, \ldots, x_{i_n}$ are bits contained in the flip-flops of the FSR, and where g is some n-variable Boolean function called the feedback function (if g is not affine, then we speak of *NFSR*, where *N* stands for nonlinear). The linear complexity of the produced sequence can be near 2^L (see [640] for general FSRs and [344] for FSRs with a quadratic feedback function; see the definition of "quadratic" at page 36). Some finalists of the eSTREAM project [495] such as Grain and Trivium use NFSRs. But the theory of NFSRs is not completely understood. The linear complexity is difficult to study in general. Even the period is not easily determined, although some special cases have been investigated [630, 702, 1045, 1046]. Nice results similar to those on the m-sequences exist in the case of feedback with carry shift-registers (FCSRs); see [30, 559, 560, 703].

1.3.2 Boolean functions and error-correcting codes

As explained above, every binary *unrestricted code* whose length equals 2^n for some positive integer n can be interpreted as a set of Boolean functions. A particular class of codes has its very definition given by means of Boolean functions. This class is that of Reed–Muller codes. We shall see in Chapter 2 that an integer lying between 0 and n and called algebraic degree can be associated to every Boolean function over $\mathbb{F}_2^n$. The *Reed–Muller code* of order $k \in \{0, \ldots, n\}$ is made of all Boolean functions over $\mathbb{F}_2^n$ whose algebraic degree is bounded above by k; see Section 4.1. This linear code has length 2^n since each Boolean function is identified to the list of its values over $\mathbb{F}_2^n$, in some order. It is linear and has nice particularities, thanks to which Reed–Muller codes are still used nowadays, even if their parameters are not very good, except for the first-order Reed–Muller code. The second-order Reed–Muller code contains a nonlinear code, called the Kerdock code, which has minimum distance almost the same as that of the first-order Reed–Muller code of the same length and size roughly the square of its size. In fact, the parameters of the Kerdock code are so good that they are provably optimal among all unrestricted codes; see Section 6.1.22.

1.4 Vectorial functions

The functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ are called (n, m)-*functions*. Such function F being given, the Boolean functions $f_1, \ldots, f_m$ defined at every $x \in \mathbb{F}_2^n$ by $F(x) = (f_1(x), \ldots, f_m(x))$, are called the *coordinate functions* of F. When the numbers m and n are not specified, (n, m)-functions are called *multioutput Boolean functions* or *vectorial Boolean functions*. Those vectorial functions whose role is to ensure confusion[23] in a cryptographic system are called *substitution boxes* (*S-boxes*).

Note that (n, m)-functions can also be viewed as taking their input in $\mathbb{F}_{2^n}$ as we have seen with Boolean functions, and if m divides n, then we shall see that the output can then be expressed as a polynomial function of the input. We shall be in particular interested in *power functions* $F(x) = x^d$, $x \in \mathbb{F}_{2^n}$.

1.4.1 Vectorial functions and stream ciphers

In the pseudorandom generators of stream ciphers, (n, m)-functions can be used to combine the outputs of n LFSRs or to filter the content of a single one, generating m bits at each clock cycle instead of only one, which increases the speed of the cipher (but risks decreasing its robustness). The attacks described about Boolean functions are obviously also efficient on these kinds of ciphers. They are in fact often more efficient – see Section 3.3, page 129 – since the attacker can combine in any way the m output bits of the function.

1.4.2 Vectorial functions and block ciphers

Vectorial functions play mainly a role with block ciphers. All known block ciphers are iterative, that is, are the iterations of a transformation depending on a key over each block of plaintext. The iterations are called *rounds* and the key used in an iteration is called a *round key*. The round keys are computed from the secret key (called the *master key*) by a *key scheduling algorithm*. The rounds consist of vectorial Boolean functions combined in different ways and involve the round key.

Remark. Boolean functions also play an important role in block ciphers, each of which admits as input a binary vector $(x_1, \ldots, x_n)$ (a block of plaintext) and outputs a binary vector $(y_1, \ldots, y_m)$; the coordinates $y_1, \ldots, y_m$ are the outputs of Boolean functions (depending on the key) over $(x_1, \ldots, x_n)$; see Figure 1.5.

But the number n of variables of these Boolean functions being large (often more than 100), they are hardly analyzed precisely. □

[23] See Section 3.1 for the meaning of this term.

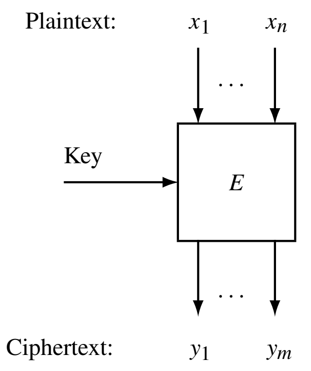


Figure 1.5 Block cipher.

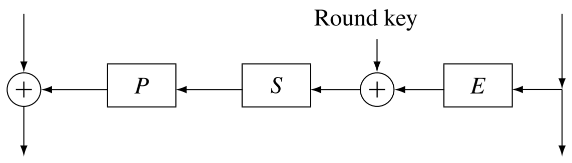


Figure 1.6 A DES round.

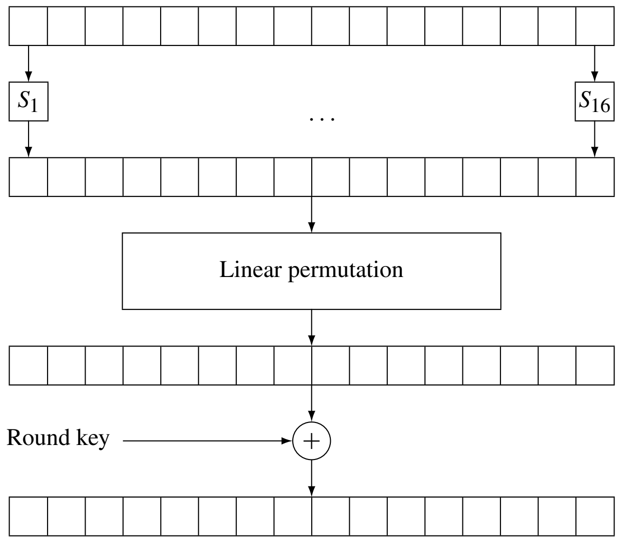


Figure 1.7 An AES round.

We give in Figures 1.6 and 1.7 a description of the rounds of the *Data Encryption Standard (DES)* [88] and of the *Advanced Encryption Standard (AES)* [404].

The input to a DES round is a binary string of length 64, divided into two strings of 32 bits each (in the figure, they enter the round, from above, on the left and on the right); confusion is achieved by the S-box, which is a nonlinear transformation of a binary string of 48 bits[24] into a 32-bit -long one. So, 32 Boolean functions on 48 variables are involved. But, in fact, this nonlinear transformation is the concatenation of eight sub-S-boxes, which transform binary strings of six bits into 4-bit-long ones. Before entering the next round, the two 32-bit-long halves of data are swapped. Such *Feistel cipher* structure does not need the involved vectorial functions (in particular the S-boxes) to be injective for the decryption to be possible. Indeed, any function of the form $(x, y) \mapsto (y, x + \phi(y))$ is a permutation. The number of output bits can be smaller than that of input bits like in the DES; it can also be larger, like in the CAST cipher [6], where input dimension is eight and output dimension is 32. However, if the S-boxes are not balanced (that is, if their output is not uniform), this represents a weakness against some attacks, and it obliges the designer to complexify the structure (for instance by including expansion boxes); see more in [957].

In the (standard) AES round, the input is a 128-bit-long string, divided into 16 strings of eight bits each; the S-box is the concatenation of 16 sub-S-boxes corresponding to 16×8 Boolean functions in eight variables. Such a *substitution permutation network* (SPN) needs the vectorial functions (in particular the S-boxes) to be bijective, so that decryption is possible. Then $n = m$. Another well-known example of such cipher is PRESENT [100]. A third general structure for block ciphers is ARX structure; see [708].

Remark. Klimov and Shamir [705] have identified a particular kind of vectorial functions usable in stream and block ciphers (and in hash functions), called *T-function*s. These are mappings F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ such that each ith bit of $F(x)$ depends only on $x_1, \dots, x_i$. For example, addition and multiplication in $\mathbb{Z}$, viewed in binary expansion, are T-functions; logical operations (XOR and AND, that is, addition and multiplication in $\mathbb{F}_2$) are T-functions too. Any composition of T-functions is a T-function as well. Their simplicity makes them appealing for lightweight cryptography. But they may be too simple to provide enough confusion; they have suffered attacks. $\square$

1.4.3 *Vectorial functions and error-correcting codes*

We shall see in Chapter 4 that interesting linear subcodes of the Reed–Muller codes and other (possibly nonlinear) codes can be built from vectorial functions.

[24] The E-box has expanded the 32-bit-long string into a 48-bit-long one.