Chapter 7

Block Ciphers

Contents in Brief

7.1 Introduction and overview

Symmetric-key block ciphers are the most prominent and important elements in many cryptographic systems. Individually, they provide confidentiality. As a fundamental building block, their versatility allows construction of pseudorandom number generators, stream ciphers, MACs, and hash functions. They may furthermore serve as a central component in message authentication techniques, data integrity mechanisms, entity authentication protocols, and (symmetric-key) digital signature schemes. This chapter examines symmetric-key block ciphers, including both general concepts and details of specific algorithms. Public-key block ciphers are discussed in Chapter 8.

No block cipher is ideally suited for all applications, even one offering a high level of security. This is a result of inevitable tradeoffs required in practical applications, including those arising from, for example, speed requirements and memory limitations (e.g., code size, data size, cache memory), constraints imposed by implementation platforms (e.g., hardware, software, chipcards), and differing tolerances of applications to properties of various modes of operation. In addition, efficiency must typically be traded off against security. Thus it is beneficial to have a number of candidate ciphers from which to draw.

Of the many block ciphers currently available, focus in this chapter is given to a subset of high profile and/or well-studied algorithms. While not guaranteed to be more secure than other published candidate ciphers (indeed, this status changes as new attacks become known), emphasis is given to those of greatest practical interest. Among these, DES is paramount; FEAL has received both serious commercial backing and a large amount of independent cryptographic analysis; and IDEA (originally proposed as a DES replacement) is widely known and highly regarded. Other recently proposed ciphers of both high promise and high profile (in part due to the reputation of their designers) are SAFER and RC5. Additional ciphers are presented in less detail.

Chapter outline

Basic background on block ciphers and algorithm-independent concepts are presented in §7.2, including modes of operation, multiple encryption, and exhaustive search techniques. Classical ciphers and cryptanalysis thereof are addressed in §7.3, including historical details on cipher machines. Modern block ciphers covered in chronological order are DES (§7.4), FEAL (§7.5), and IDEA (§7.6), followed by SAFER, RC5, and other ciphers in §7.7, collectively illustrating a wide range of modern block cipher design approaches. Further notes, including details on additional ciphers (e.g., Lucifer) and references for the chapter, may be found in §7.8.

7.2 Background and general concepts

Introductory material on block ciphers is followed by subsections addressing modes of operation, and discussion of exhaustive key search attacks and multiple encryption.

7.2.1 Introduction to block ciphers

Block ciphers can be either symmetric-key or public-key. The main focus of this chapter is symmetric-key block ciphers; public-key encryption is addressed in Chapter 8.

(i) Block cipher definitions

A block cipher is a function (see §1.3.1) which maps n-bit plaintext blocks to n-bit ciphertext blocks; n is called the *blocklength*. It may be viewed as a simple substitution cipher with large character size. The function is parameterized by a k-bit key K,[1] taking values from a subset $\mathcal{K}$ (the *key space*) of the set of all k-bit vectors V_k. It is generally assumed that the key is chosen at random. Use of plaintext and ciphertext blocks of equal size avoids data expansion.

To allow unique decryption, the encryption function must be one-to-one (i.e., invertible). For n-bit plaintext and ciphertext blocks and a fixed key, the encryption function is a bijection, defining a permutation on n-bit vectors. Each key potentially defines a different bijection. The number of keys is $|\mathcal{K}|$, and the *effective key size* is $\lg|\mathcal{K}|$; this equals the key length if all k-bit vectors are valid keys ($\mathcal{K} = V_k$). If keys are equiprobable and each defines a different bijection, the *entropy* of the key space is also $\lg|\mathcal{K}|$.

7.1 Definition An n-bit *block cipher* is a function $E : V_n \times \mathcal{K} \to V_n$, such that for each key $K \in \mathcal{K}$, $E(P, K)$ is an invertible mapping (the *encryption function* for K) from V_n to V_n, written $E_K(P)$. The inverse mapping is the *decryption function*, denoted $D_K(C)$. $C = E_K(P)$ denotes that ciphertext C results from encrypting plaintext P under K.

Whereas block ciphers generally process plaintext in relatively large blocks (e.g., $n \geq 64$), stream ciphers typically process smaller units (see Note 6.1); the distinction, however, is not definitive (see Remark 7.25). For plaintext messages exceeding one block in length, various modes of operation for block ciphers are used (see §7.2.2).

The most general block cipher implements every possible substitution, as per Definition 7.2. To represent the key of such an n-bit (true) random block cipher would require

[1] This use of symbols k and K may differ from other chapters.

$\lg(2^n!) \approx (n - 1.44)2^n$ bits, or roughly 2^n times the number of bits in a message block. This excessive bitsize makes (true) random ciphers impractical. Nonetheless, it is an accepted design principle that the encryption function corresponding to a randomly selected key should *appear* to be a randomly chosen invertible function.

7.2 Definition A (*true*) *random cipher* is an n-bit block cipher implementing all $2^n!$ bijections on 2^n elements. Each of the $2^n!$ keys specifies one such permutation.

A block cipher whose block size n is too small may be vulnerable to attacks based on statistical analysis. One such attack involves simple frequency analysis of ciphertext blocks (see Note 7.74). This may be thwarted by appropriate use of modes of operation (e.g., Algorithm 7.13). Other such attacks are considered in Note 7.8. However, choosing too large a value for the blocksize n may create difficulties as the complexity of implementation of many ciphers grows rapidly with block size. In practice, consequently, for larger n, easily-implementable functions are necessary which *appear* to be random (without knowledge of the key).

An encryption function per Definition 7.1 is a deterministic mapping. Each pairing of plaintext block P and key K maps to a unique ciphertext block. In contrast, in a randomized encryption technique (Definition 7.3; see also Remark 8.22), each (P, K) pair is associated with a set $C_{(P,K)}$ of eligible ciphertext blocks; each time P is encrypted under K, an output R from a random source non-deterministically selects one of these eligible blocks. To ensure invertibility, for every fixed key K, the subsets $C_{(P,K)}$ over all plaintexts P must be disjoint. Since the encryption function is essentially one-to-many involving an additional parameter R (cf. homophonic substitution, §7.3.2), the requirement for invertibility implies data expansion, which is a disadvantage of randomized encryption and is often unacceptable.

7.3 Definition A *randomized encryption* mapping is a function E from a plaintext space V_n to a ciphertext space V_m, $m > n$, drawing elements from a space of random numbers $\mathcal{R}$ $= V_t$. E is defined by $E : V_n \times \mathcal{K} \times \mathcal{R} \to V_m$, such that for each key $K \in \mathcal{K}$ and $R \in \mathcal{R}$, $E(P, K, R)$, also written $E_K^R(P)$, maps $P \in V_n$ to V_m; and an inverse (corresponding decryption) function exists, mapping $V_m \times \mathcal{K} \to V_n$.

(ii) Practical security and complexity of attacks

The objective of a block cipher is to provide confidentiality. The corresponding objective of an adversary is to recover plaintext from ciphertext. A block cipher is *totally broken* if a key can be found, and *partially broken* if an adversary is able to recover part of the plaintext (but not the key) from ciphertext.

7.4 Note (*standard assumptions*) To evaluate block cipher security, it is customary to always assume that an adversary (i) has access to all data transmitted over the ciphertext channel; and (ii) (*Kerckhoffs' assumption*) knows all details of the encryption function except the secret key (which security consequently rests entirely upon).

Under the assumptions of Note 7.4, attacks are classified based on what information a cryptanalyst has access to in addition to intercepted ciphertext (cf. §1.13.1). The most prominent classes of attack for symmetric-key ciphers are (for a fixed key):

1. *ciphertext-only* – no additional information is available.
2. *known-plaintext* – plaintext-ciphertext pairs are available.

3. *chosen-plaintext* – ciphertexts are available corresponding to plaintexts of the adversary's choice. A variation is an *adaptive chosen-plaintext* attack, where the choice of plaintexts may depend on previous plaintext-ciphertext pairs.

Additional classes of attacks are given in Note 7.6; while somewhat more hypothetical, these are nonetheless of interest for the purposes of analysis and comparison of ciphers.

7.5 Remark (*chosen-plaintext principle*) It is customary to use ciphers resistant to chosen-plaintext attack even when mounting such an attack is not feasible. A cipher secure against chosen-plaintext attack is secure against known-plaintext and ciphertext-only attacks.

7.6 Note (*chosen-ciphertext and related-key attacks*) A *chosen-ciphertext* attack operates under the following model: an adversary is allowed access to plaintext-ciphertext pairs for some number of ciphertexts of his choice, and thereafter attempts to use this information to recover the key (or plaintext corresponding to some new ciphertext). In a *related-key attack*, an adversary is assumed to have access to the encryption of plaintexts under both an unknown key and (unknown) keys chosen to have or known to have certain relationships with this key.

With few exceptions (e.g., the one-time pad), the best available measure of security for practical ciphers is the complexity of the best (currently) known attack. Various aspects of such complexity may be distinguished as follows:

1. *data complexity* – expected number of input data units required (e.g., ciphertext).
2. *storage complexity* – expected number of storage units required.
3. *processing complexity* – expected number of operations required to process input data and/or fill storage with data (at least one time unit per storage unit).

The *attack complexity* is the dominant of these (e.g., for linear cryptanalysis on DES, essentially the data complexity). When parallelization is possible, processing complexity may be divided across many processors (but not reduced), reducing attack time.

Given a data complexity of 2^n, an attack is always possible; this many different n-bit blocks completely characterize the encryption function for a fixed k-bit key. Similarly, given a processing complexity of 2^k, an attack is possible by exhaustive key search (§7.2.3). Thus as a minimum, the effective key size should be sufficiently large to preclude exhaustive key search, and the block size sufficiently large to preclude exhaustive data analysis. A block cipher is considered *computationally secure* if these conditions hold and no known attack has both data and processing complexity significantly less than, respectively, 2^n and 2^k. However, see Note 7.8 for additional concerns related to block size.

7.7 Remark (*passive vs. active complexity*) For symmetric-key block ciphers, data complexity is beyond the control of the adversary, and is *passive complexity* (plaintext-ciphertext pairs cannot be generated by the adversary itself). Processing complexity is *active complexity* which typically benefits from increased resources (e.g., parallelization).

7.8 Note (*attacks based on small block size*) Security concerns which arise if the block size n is too small include the feasibility of *text dictionary attacks* and *matching ciphertext attacks*. A text dictionary may be assembled if plaintext-ciphertext pairs become known for a fixed key. The more pairs available, the larger the dictionary and the greater the chance of locating a random ciphertext block therein. A complete dictionary results if 2^n plaintext-ciphertext pairs become known, and fewer suffice if plaintexts contain redundancy and a non-chaining mode of encryption (such as ECB) is used. Moreover, if about $2^{n/2}$ such pairs are known, and about $2^{n/2}$ ciphertexts are subsequently created, then by the birthday paradox one expects to locate a ciphertext in the dictionary. Relatedly, from ciphertext blocks alone, as the number of available blocks approaches $2^{n/2}$, one expects to find matching ciphertext blocks. These may reveal partial information about the corresponding plaintexts, depending on the mode of operation of the block cipher, and the amount of redundancy in the plaintext.

Computational and unconditional security are discussed in §1.13.3. Unconditional security is both unnecessary in many applications and impractical; for example, it requires as many bits of secret key as plaintext, and cannot be provided by a block cipher used to encrypt more than one block (due to Fact 7.9, since identical ciphertext implies matching plaintext). Nonetheless, results on unconditional security provide insight for the design of practical ciphers, and has motivated many of the principles of cryptographic practice currently in use (see Remark 7.10).

7.9 Fact A cipher provides *perfect secrecy* (unconditional security) if the ciphertext and plaintext blocks are statistically independent.

7.10 Remark (*theoretically-motivated principles*) The unconditional security of the one-time-pad motivates both additive stream ciphers (Chapter 6) and the frequent changing of cryptographic keys (§13.3.1). Theoretical results regarding the effect of redundancy on unicity distance (Fact 7.71) motivate the principle that for plaintext confidentiality, the plaintext data should be as random as possible, e.g., via data-compression prior to encryption, use of random-bit fields in message blocks, or randomized encryption (Definition 7.3). The latter two techniques may, however, increase the data length or allow covert channels.

(iii) Criteria for evaluating block ciphers and modes of operation

Many criteria may be used for evaluating block ciphers in practice, including:

1. *estimated security level*. Confidence in the (historical) security of a cipher grows if it has been subjected to and withstood expert cryptanalysis over a substantial time period, e.g., several years or more; such ciphers are certainly considered more secure than those which have not. This may include the performance of selected cipher components relative to various design criteria which have been proposed or gained favor in recent years. The amount of ciphertext required to mount practical attacks often vastly exceeds a cipher's unicity distance (Definition 7.69), which provides a theoretical estimate of the amount of ciphertext required to recover the unique encryption key.
2. *key size*. The effective bitlength of the key, or more specifically, the entropy of the key space, defines an upper bound on the security of a cipher (by considering exhaustive search). Longer keys typically impose additional costs (e.g., generation, transmission, storage, difficulty to remember passwords).
3. *throughput*. Throughput is related to the complexity of the cryptographic mapping (see below), and the degree to which the mapping is tailored to a particular implementation medium or platform.
4. *block size*. Block size impacts both security (larger is desirable) and complexity (larger is more costly to implement). Block size may also affect performance, for example, if padding is required.
5. *complexity of cryptographic mapping*. Algorithmic complexity affects the implementation costs both in terms of development and fixed resources (hardware gate

count or software code/data size), as well as real-time performance for fixed resources (throughput). Some ciphers specifically favor hardware or software implementations.

6. *data expansion*. It is generally desirable, and often mandatory, that encryption does not increase the size of plaintext data. Homophonic substitution and randomized encryption techniques result in data expansion.
7. *error propagation*. Decryption of ciphertext containing bit errors may result in various effects on the recovered plaintext, including propagation of errors to subsequent plaintext blocks. Different error characteristics are acceptable in various applications. Block size (above) typically affects error propagation.

7.2.2 Modes of operation

A block cipher encrypts plaintext in fixed-size n-bit blocks (often $n = 64$). For messages exceeding n bits, the simplest approach is to partition the message into n-bit blocks and encrypt each separately. This electronic-codebook (ECB) mode has disadvantages in most applications, motivating other methods of employing block ciphers (*modes of operation*) on larger messages. The four most common modes are ECB, CBC, CFB, and OFB. These are summarized in Figure 7.1 and discussed below.

In what follows, E_K denotes the encryption function of the block cipher E parameterized by key K, while E_K^{-1} denotes decryption (cf. Definition 7.1). A plaintext message $x = x_1 \dots x_t$ is assumed to consist of n-bit blocks for ECB and CBC modes (see Algorithm 9.58 regarding padding), and r-bit blocks for CFB and OFB modes for appropriate fixed $r \leq n$.

(i) ECB mode

The *electronic codebook* (ECB) mode of operation is given in Algorithm 7.11 and illustrated in Figure 7.1(a).

7.11 Algorithm ECB mode of operation

INPUT: k-bit key K; n-bit plaintext blocks $x_1, \dots, x_t$.
SUMMARY: produce ciphertext blocks $c_1, \dots, c_t$; decrypt to recover plaintext.

1. Encryption: for $1 \leq j \leq t$, $c_j \leftarrow E_K(x_j)$.
2. Decryption: for $1 \leq j \leq t$, $x_j \leftarrow E_K^{-1}(c_j)$.

Properties of the ECB mode of operation:

1. Identical plaintext blocks (under the same key) result in identical ciphertext.
2. Chaining dependencies: blocks are enciphered independently of other blocks. Reordering ciphertext blocks results in correspondingly re-ordered plaintext blocks.
3. Error propagation: one or more bit errors in a single ciphertext block affect decipherment of that block only. For typical ciphers E, decryption of such a block is then random (with about 50% of the recovered plaintext bits in error). Regarding bits being deleted, see Remark 7.15.

7.12 Remark (*use of ECB mode*) Since ciphertext blocks are independent, malicious substitution of ECB blocks (e.g., insertion of a frequently occurring block) does not affect the decryption of adjacent blocks. Furthermore, block ciphers do not hide data patterns – identical ciphertext blocks imply identical plaintext blocks. For this reason, the ECB mode is not recommended for messages longer than one block, or if keys are reused for more than

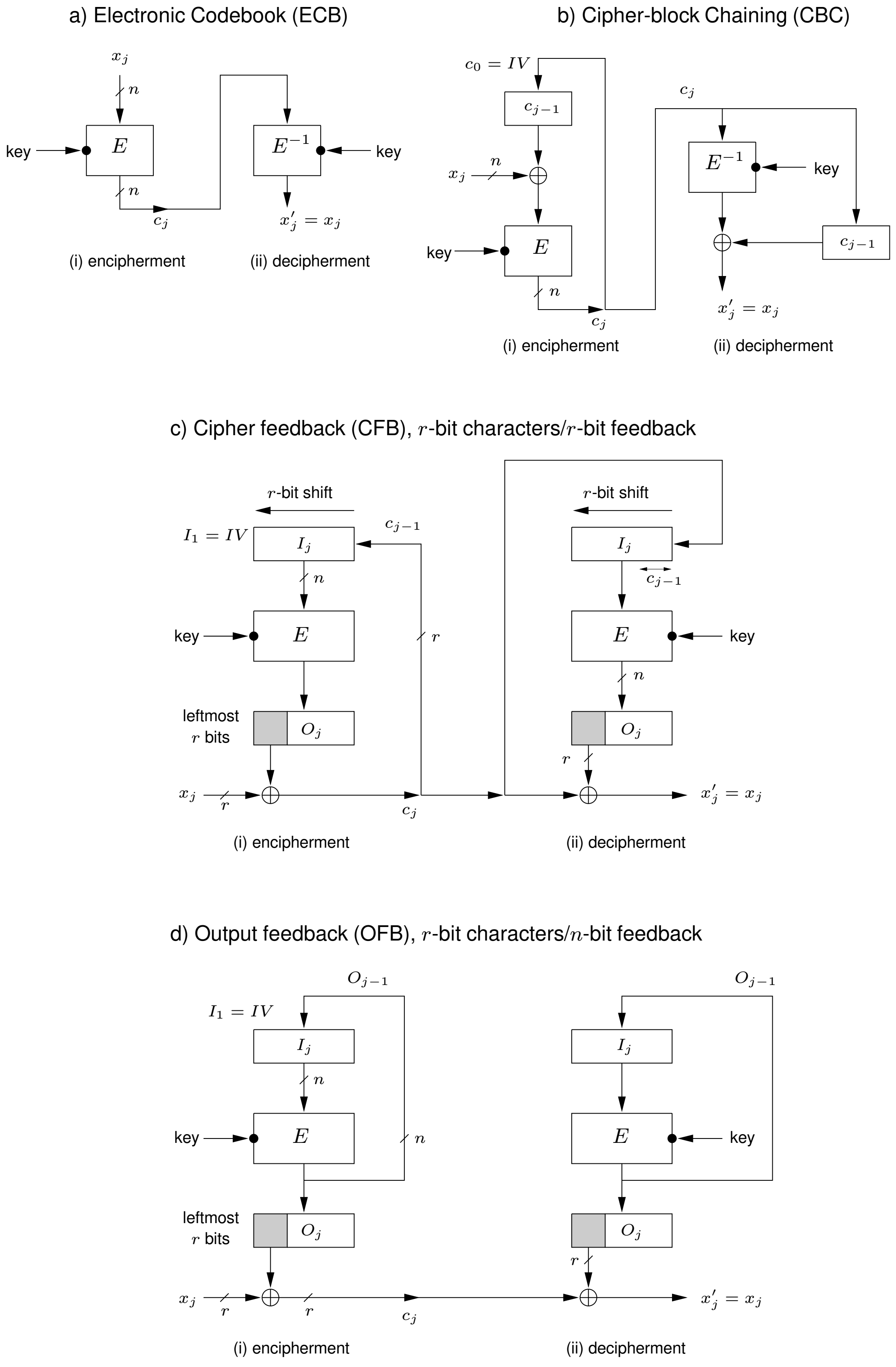

Figure 7.1: *Common modes of operation for an n-bit block cipher.*

a single one-block message. Security may be improved somewhat by inclusion of random padding bits in each block.

(ii) CBC mode

The *cipher-block chaining* (CBC) mode of operation, specified in Algorithm 7.13 and illustrated in Figure 7.1(b), involves use of an n-bit initialization vector, denoted IV.

7.13 Algorithm CBC mode of operation

INPUT: k-bit key K; n-bit IV; n-bit plaintext blocks $x_1, \ldots, x_t$.
SUMMARY: produce ciphertext blocks $c_1, \ldots, c_t$; decrypt to recover plaintext.

1. Encryption: $c_0 \leftarrow IV$. For $1 \leq j \leq t$, $c_j \leftarrow E_K(c_{j-1} \oplus x_j)$.
2. Decryption: $c_0 \leftarrow IV$. For $1 \leq j \leq t$, $x_j \leftarrow c_{j-1} \oplus E_K^{-1}(c_j)$.

Properties of the CBC mode of operation:

1. Identical plaintexts: identical ciphertext blocks result when the same plaintext is enciphered under the same key and IV. Changing the IV, key, or first plaintext block (e.g., using a counter or random field) results in different ciphertext.
2. Chaining dependencies: the chaining mechanism causes ciphertext c_j to depend on x_j and all preceding plaintext blocks (the entire dependency on preceding blocks is, however, contained in the value of the previous ciphertext block). Consequently, rearranging the order of ciphertext blocks affects decryption. Proper decryption of a correct ciphertext block requires a correct preceding ciphertext block.
3. Error propagation: a single bit error in ciphertext block c_j affects decipherment of blocks c_j and c_{j+1} (since x_j depends on c_j and c_{j-1}). Block x'_j recovered from c_j is typically totally random (50% in error), while the recovered plaintext x'_{j+1} has bit errors precisely where c_j did. Thus an adversary may cause predictable bit changes in x_{j+1} by altering corresponding bits of c_j. See also Remark 7.14.
4. Error recovery: the CBC mode is *self-synchronizing* or *ciphertext autokey* (see Remark 7.15) in the sense that if an error (including loss of one or more entire blocks) occurs in block c_j but not c_{j+1}, c_{j+2} is correctly decrypted to x_{j+2}.

7.14 Remark (*error propagation in encryption*) Although CBC mode decryption recovers from errors in ciphertext blocks, modifications to a plaintext block x_j during encryption alter all subsequent ciphertext blocks. This impacts the usability of chaining modes for applications requiring random read/write access to encrypted data. The ECB mode is an alternative (but see Remark 7.12).

7.15 Remark (*self-synchronizing vs. framing errors*) Although self-synchronizing in the sense of recovery from bit errors, recovery from "lost" bits causing errors in block boundaries (*framing integrity errors*) is not possible in the CBC or other modes.

7.16 Remark (*integrity of IV in CBC*) While the IV in the CBC mode need not be secret, its integrity should be protected, since malicious modification thereof allows an adversary to make predictable bit changes to the first plaintext block recovered. Using a secret IV is one method for preventing this. However, if message integrity is required, an appropriate mechanism should be used (see §9.6.5); encryption mechanisms typically guarantee confidentiality only.

(iii) CFB mode

While the CBC mode processes plaintext n bits at a time (using an n-bit block cipher), some applications require that r-bit plaintext units be encrypted and transmitted without delay, for some fixed $r < n$ (often $r = 1$ or $r = 8$). In this case, the *cipher feedback* (CFB) mode may be used, as specified in Algorithm 7.17 and illustrated in Figure 7.1(c).

7.17 Algorithm CFB mode of operation (CFB-r)

INPUT: k-bit key K; n-bit IV; r-bit plaintext blocks $x_1, \ldots, x_u$ ($1 \le r \le n$).
SUMMARY: produce r-bit ciphertext blocks $c_1, \ldots, c_u$; decrypt to recover plaintext.

1. Encryption: $I_1 \leftarrow IV$. (I_j is the input value in a shift register.) For $1 \le j \le u$:
   (a) $O_j \leftarrow E_K(I_j)$. (Compute the block cipher output.)
   (b) $t_j \leftarrow$ the r leftmost bits of O_j. (Assume the leftmost is identified as bit 1.)
   (c) $c_j \leftarrow x_j \oplus t_j$. (Transmit the r-bit ciphertext block c_j.)
   (d) $I_{j+1} \leftarrow 2^r \cdot I_j + c_j \bmod 2^n$. (Shift c_j into right end of shift register.)
2. Decryption: $I_1 \leftarrow IV$. For $1 \le j \le u$, upon receiving c_j:
   $x_j \leftarrow c_j \oplus t_j$, where t_j, O_j and I_j are computed as above.

Properties of the CFB mode of operation:

1. Identical plaintexts: as per CBC encryption, changing the IV results in the same plaintext input being enciphered to a different output. The IV need not be secret (although an unpredictable IV may be desired in some applications).
2. Chaining dependencies: similar to CBC encryption, the chaining mechanism causes ciphertext block c_j to depend on both x_j and preceding plaintext blocks; consequently, re-ordering ciphertext blocks affects decryption. Proper decryption of a correct ciphertext block requires the preceding $\lceil n/r \rceil$ ciphertext blocks to be correct (so that the shift register contains the proper value).
3. Error propagation: one or more bit errors in any single r-bit ciphertext block c_j affects the decipherment of that and the next $\lceil n/r \rceil$ ciphertext blocks (i.e., until n bits of ciphertext are processed, after which the error block c_j has shifted entirely out of the shift register). The recovered plaintext x'_j will differ from x_j precisely in the bit positions c_j was in error; the other incorrectly recovered plaintext blocks will typically be random vectors, i.e., have 50% of bits in error. Thus an adversary may cause predictable bit changes in x_j by altering corresponding bits of c_j.
4. Error recovery: the CFB mode is self-synchronizing similar to CBC, but requires $\lceil n/r \rceil$ ciphertext blocks to recover.
5. Throughput: for $r < n$, throughput is decreased by a factor of n/r (vs. CBC) in that each execution of E yields only r bits of ciphertext output.

7.18 Remark (*CFB use of encryption only*) Since the encryption function E is used for both CFB encryption and decryption, the CFB mode must not be used if the block cipher E is a public-key algorithm; instead, the CBC mode should be used.

7.19 Example (*ISO variant of CFB*) The CFB mode of Algorithm 7.17 may be modified as follows, to allow processing of plaintext blocks (characters) whose bitsize s is less than the bitsize r of the feedback variable (e.g., 7-bit characters using 8-bit feedback; $s < r$). The leftmost s (rather than r) bits of O_j are assigned to t_j; the s-bit ciphertext character c_j is computed; the feedback variable is computed from c_j by pre-prepending (on the left) $r - s$ 1-bits; the resulting r-bit feedback variable is shifted into the least significant (LS) end of the shift register as before. □

(iv) OFB mode

The *output feedback* (OFB) mode of operation may be used for applications in which all error propagation must be avoided. It is similar to CFB, and allows encryption of various block sizes (characters), but differs in that the output of the encryption block function E (rather than the ciphertext) serves as the feedback.

Two versions of OFB using an n-bit block cipher are common. The ISO version (Figure 7.1(d) and Algorithm 7.20) requires an n-bit feedback, and is more secure (Note 7.24). The earlier FIPS version (Algorithm 7.21) allows $r < n$ bits of feedback.

7.20 Algorithm OFB mode with full feedback (per ISO 10116)

INPUT: k-bit key K; n-bit IV; r-bit plaintext blocks $x_1, \dots, x_u$ ($1 \le r \le n$).
SUMMARY: produce r-bit ciphertext blocks $c_1, \dots, c_u$; decrypt to recover plaintext.

1. Encryption: $I_1 \leftarrow IV$. For $1 \le j \le u$, given plaintext block x_j:
   (a) $O_j \leftarrow E_K(I_j)$. (Compute the block cipher output.)
   (b) $t_j \leftarrow$ the r leftmost bits of O_j. (Assume the leftmost is identified as bit 1.)
   (c) $c_j \leftarrow x_j \oplus t_j$. (Transmit the r-bit ciphertext block c_j.)
   (d) $I_{j+1} \leftarrow O_j$. (Update the block cipher input for the next block.)
2. Decryption: $I_1 \leftarrow IV$. For $1 \le j \le u$, upon receiving c_j:
   $x_j \leftarrow c_j \oplus t_j$, where t_j, O_j, and I_j are computed as above.

7.21 Algorithm OFB mode with r-bit feedback (per FIPS 81)

INPUT: k-bit key K; n-bit IV; r-bit plaintext blocks $x_1, \dots, x_u$ ($1 \le r \le n$).
SUMMARY: produce r-bit ciphertext blocks $c_1, \dots, c_u$; decrypt to recover plaintext.
As per Algorithm 7.20, but with "$I_{j+1} \leftarrow O_j$" replaced by:
$I_{j+1} \leftarrow 2^r \cdot I_j + t_j \bmod 2^n$. (Shift output t_j into right end of shift register.)

Properties of the OFB mode of operation:

1. Identical plaintexts: as per CBC and CFB modes, changing the IV results in the same plaintext being enciphered to a different output.
2. Chaining dependencies: the keystream is plaintext-independent (see Remark 7.22).
3. Error propagation: one or more bit errors in any ciphertext character c_j affects the decipherment of only that character, in the precise bit position(s) c_j is in error, causing the corresponding recovered plaintext bit(s) to be complemented.
4. Error recovery: the OFB mode recovers from ciphertext bit errors, but cannot self-synchronize after loss of ciphertext bits, which destroys alignment of the decrypting keystream (in which case explicit re-synchronization is required).
5. Throughput: for $r < n$, throughput is decreased as per the CFB mode. However, in all cases, since the keystream is independent of plaintext or ciphertext, it may be pre-computed (given the key and IV).

7.22 Remark (*changing IV in OFB*) The IV, which need not be secret, must be changed if an OFB key K is re-used. Otherwise an identical keystream results, and by XORing corresponding ciphertexts an adversary may reduce cryptanalysis to that of a running-key cipher with one plaintext as the running key (cf. Example 7.58 ff.).

Remark 7.18 on public-key block ciphers applies to the OFB mode as well as CFB.

7.23 Example (*counter mode*) A simplification of OFB involves updating the input block as a counter, $I_{j+1} = I_j + 1$, rather than using feedback. This both avoids the short-cycle problem of Note 7.24, and allows recovery from errors in computing E. Moreover, it provides a random-access property: ciphertext block i need not be decrypted in order to decrypt block $i+1$. □

7.24 Note (*OFB feedback size*) In OFB with full n-bit feedback (Algorithm 7.20), the keystream is generated by the iterated function $O_j = E_K(O_{j-1})$. Since E_K is a permutation, and under the assumption that for random K, E_K is effectively a random choice among all $(2^n)!$ permutations on n elements, it can be shown that for a fixed (random) key and starting value, the expected cycle length before repeating any value O_j is about 2^{n-1}. On the other hand, if the number of feedback bits is $r < n$ as allowed in Algorithm 7.21, the keystream is generated by the iteration $O_j = f(O_{j-1})$ for some non-permutation f which, assuming it behaves as a random function, has an expected cycle length of about $2^{n/2}$. Consequently, it is strongly recommended to use the OFB mode with full n-bit feedback.

7.25 Remark (*modes as stream ciphers*) It is clear that both the OFB mode with full feedback (Algorithm 7.20) and the counter mode (Example 7.23) employ a block cipher as a keystream generator for a stream cipher. Similarly the CFB mode encrypts a character stream using the block cipher as a (plaintext-dependent) keystream generator. The CBC mode may also be considered a stream cipher with n-bit blocks playing the role of very large characters. Thus modes of operation allow one to define stream ciphers from block ciphers.

7.2.3 Exhaustive key search and multiple encryption

A fixed-size key defines an upper bound on the security of a block cipher, due to exhaustive key search (Fact 7.26). While this requires either known-plaintext or plaintext containing redundancy, it has widespread applicability since cipher operations (including decryption) are generally designed to be computationally efficient.

A design technique which complicates exhaustive key search is to make the task of changing cipher keys computationally expensive, while allowing encryption with a fixed key to remain relatively efficient. Examples of ciphers with this property include the block cipher Khufu and the stream cipher SEAL.

7.26 Fact (*exhaustive key search*) For an n-bit block cipher with k-bit key, given a small number (e.g., $\lceil (k+4)/n \rceil$) of plaintext-ciphertext pairs encrypted under key K, K can be recovered by exhaustive key search in an expected time on the order of 2^{k-1} operations.

Justification: Progress through the entire key space, decrypting a fixed ciphertext C with each trial key, and discarding those keys which do not yield the known plaintext P. The target key is among the undiscarded keys. The number of false alarms expected (non-target keys which map C to P) depends on the relative size of k and n, and follows from unicity distance arguments; additional (P', C') pairs suffice to discard false alarms. One expects to find the correct key after searching half the key space.

7.27 Example (*exhaustive DES key search*) For DES, $k = 56$, $n = 64$, and the expected requirement by Fact 7.26 is 2^{55} decryptions and a single plaintext-ciphertext pair. □

If the underlying plaintext is known to contain redundancy as in Example 7.28, then ciphertext-only exhaustive key search is possible with a relatively small number of ciphertexts.

7.28 Example (*ciphertext-only DES key search*) Suppose DES is used to encrypt 64-bit blocks of 8 ASCII characters each, with one bit per character serving as an even parity bit. Trial decryption with an incorrect key K yields all 8 parity bits correct with probability 2^{-8}, and correct parity for t different blocks (each encrypted by K) with probability 2^{-8t}. If this is used as a filter over all 2^{56} keys, the expected number of unfiltered incorrect keys is $2^{56}/2^{8t}$. For most practical purposes, $t = 10$ suffices. □

(i) Cascades of ciphers and multiple encryption

If a block cipher is susceptible to exhaustive key search (due to inadequate keylength), encipherment of the same message block more than once may increase security. Various such techniques for multiple encryption of n-bit messages are considered here. Once defined, they may be extended to messages exceeding one block by using standard modes of operation (§7.2.2), with E denoting multiple rather than single encryption.

7.29 Definition A *cascade cipher* is the concatenation of $L \geq 2$ block ciphers (called *stages*), each with independent keys. Plaintext is input to first stage; the output of stage i is input to stage $i + 1$; and the output of stage L is the cascade's ciphertext output.

In the simplest case, all stages in a cascade cipher have k-bit keys, and the stage inputs and outputs are all n-bit quantities. The stage ciphers may differ (*general cascade of ciphers*), or all be identical (*cascade of identical ciphers*).

7.30 Definition *Multiple encryption* is similar to a cascade of L identical ciphers, but the stage keys need not be independent, and the stage ciphers may be either a block cipher E or its corresponding decryption function $D = E^{-1}$.

Two important cases of multiple encryption are double and triple encryption, as illustrated in Figure 7.2 and defined below.

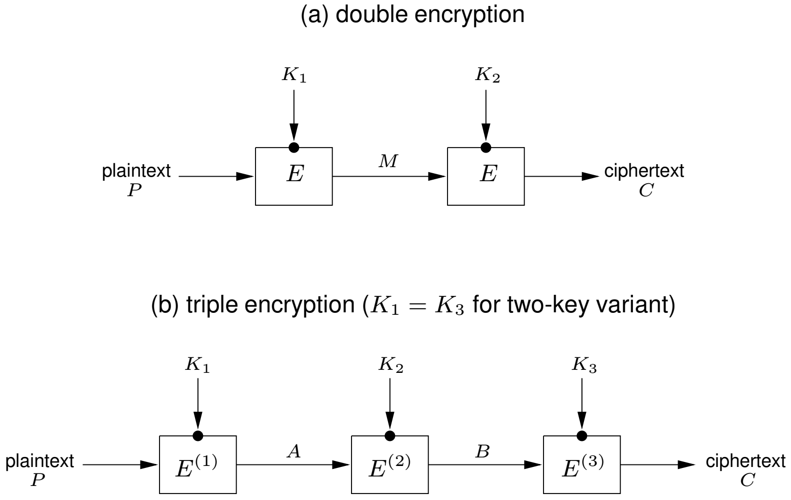

Figure 7.2: *Multiple encryption.*

7.31 Definition *Double encryption* is defined as $E(x) = E_{K_2}(E_{K_1}(x))$, where E_K denotes a block cipher E with key K.

7.32 Definition *Triple encryption* is defined as $E(x) = E^{(3)}_{K_3}(E^{(2)}_{K_2}(E^{(1)}_{K_1}(x)))$, where $E^{(j)}_K$ denotes either E_K or $D_K = E^{-1}_K$. The case $E(x) = E_{K_3}(D_{K_2}(E_{K_1}(x)))$ is called *E-D-E triple-encryption*; the subcase $K_1 = K_3$ is often called *two-key triple-encryption*.

Independent stage keys K_1 and K_2 are typically used in double encryption. In triple encryption (Definition 7.32), to save on key management and storage costs, dependent stage keys are often used. E-D-E triple-encryption with $K_1 = K_2 = K_3$ is backwards compatible with (i.e., equivalent to) single encryption.

(ii) Meet-in-the-middle attacks on multiple encryption

A naive exhaustive key search attack on double encryption tries all 2^{2k} key pairs. The attack of Fact 7.33 reduces time from 2^{2k}, at the cost of substantial space.

7.33 Fact For a block cipher with a k-bit key, a known-plaintext *meet-in-the-middle* attack defeats double encryption using on the order of 2^k operations and 2^k storage.

Justification (basic meet-in-the-middle): Noting Figure 7.2(a), given a (P, C) pair, compute $M_i = E_i(P)$ under all 2^k possible key values $K_1 = i$; store all pairs (M_i, i), sorted or indexed on M_i (e.g., using conventional hashing). Decipher C under all 2^k possible values $K_2 = j$, and for each pair (M_j, j) where $M_j = D_j(C)$, check for *hits* $M_j = M_i$ against entries M_i in the first table. (This can be done creating a second sorted table, or simply checking each M_j entry as generated.) Each hit identifies a candidate solution key pair (i, j), since $E_i(P) = M = D_j(C)$. Using a second known-plaintext pair (P', C') (cf. Fact 7.35), discard candidate key pairs which do not map P' to C'.

A concept analogous to unicity distance for ciphertext-only attack (Definition 7.69) can be defined for known-plaintext key search, based on the following strategy. Select a key; check if it is consistent with a given set (history) of plaintext-ciphertext pairs; if so, label the key a *hit*. A hit that is not the target key is a *false key hit*.

7.34 Definition The number of plaintext-ciphertext pairs required to uniquely determine a key under a known-plaintext key search is the *known-plaintext unicity distance*. This is the smallest integer t such that a history of length t makes false key hits improbable.

Using Fact 7.35, the (known-plaintext) unicity distance of a cascade of L random ciphers can be estimated. Less than one false hit is expected when $t > Lk/n$.

7.35 Fact For an L-stage cascade of random block ciphers with n-bit blocks and k-bit keys, the expected number of false key hits for a history of length t is about 2^{Lk-tn}.

Fact 7.35 holds with respect to random block ciphers defined as follows (cf. Definitions 7.2 and 7.70): given n and k, of the possible $(2^n)!$ permutations on 2^n elements, choose 2^k randomly and with equal probabilities, and associate these with the 2^k keys.

7.36 Example (*meet-in-the-middle – double-DES*) Applying Fact 7.33 to DES ($n = 64$, $k = 56$), the number of candidate key pairs expected for one (P, C) pair is $2^{48} = 2^k \cdot 2^k/2^n$, and the likelihood of a false key pair satisfying a second (P', C') sample is $2^{-16} = 2^{48}/2^n$. Thus with high probability, two (P, C) pairs suffice for key determination. This agrees with the unicity distance estimate of Fact 7.35: for $L = 2$, a history of length $t = 2$ yields 2^{-16} expected false key hits. □

A naive exhaustive attack on all key pairs in double-DES uses 2^{112} time and negligible space, while the meet-in-the-middle attack (Fact 7.33) requires 2^{56} time and 2^{56} space. Note 7.37 illustrates that the latter can be modified to yield a time-memory trade-off at any point between these two extremes, with the time-memory product essentially constant at 2^{112} (e.g., 2^{72} time, 2^{40} space).

7.37 Note (*time-memory tradeoff – double-encryption*) In the attack of Example 7.36, memory may be reduced (from tables of 2^{56} entries) by independently guessing s bits of each of K_1, K_2 (for any fixed s, $0 \le s \le k$). The tables then each have 2^{k-s} entries (fixing s key bits eliminates 2^s entries), but the attack must be run over $2^s \cdot 2^s$ pairs of such tables to allow all possible key pairs. The memory requirement is $2 \cdot 2^{k-s}$ entries (each $n+k-s$ bits, omitting s fixed key bits), while time is on the order of $2^{2s} \cdot 2^{k-s} = 2^{k+s}$. The time-memory product is 2^{2k+1}.

7.38 Note (*generalized meet-in-the-middle trade-off*) Variations of Note 7.37 allow time-space tradeoffs for meet-in-the-middle key search on any concatenation of $L \ge 2$ ciphers. For L even, meeting between the first and last $L/2$ stages results in requirements on the order of $2 \cdot 2^{(kL/2)-s}$ space and $2^{(kL/2)+s}$ time, $0 \le s \le kL/2$. For L odd, meeting after the first $(L-1)/2$ and before the last $(L+1)/2$ stages results in requirements on the order of $2 \cdot 2^{k(L-1)/2\,-\,s}$ space and $2^{k(L+1)/2\,+\,s}$ time, $1 \le s \le k(L-1)/2$.

For a block cipher with k-bit key, a naive attack on two-key triple encryption (Definition 7.32) involves trying all 2^{2k} key pairs. Fact 7.39 notes a chosen-plaintext alternative.

7.39 Fact For an n-bit block cipher with k-bit key, two-key triple encryption may be defeated by a chosen-plaintext attack requiring on the order of 2^k of each of the following: cipher operations, words of $(n+k)$-bit storage, and plaintext-ciphertext pairs with plaintexts chosen.

Justification (chosen-plaintext attack on two-key triple-encryption): Using 2^k chosen plaintexts, two-key triple encryption may be reduced to double-encryption as follows. Noting Figure 7.2(b), focus on the case where the result after the first encryption stage is the all-zero vector $A = 0$. For all 2^k values $K_1 = i$, compute $P_i = E_i^{-1}(A)$. Submit each resulting P_i as a chosen plaintext, obtaining the corresponding ciphertext C_i. For each, compute $B_i = E_i^{-1}(C_i)$, representing an intermediate result B after the second of three encryption stages. Note that the values P_i also represent candidate values B. Sort the values P_j and B_j in a table (using standard hashing for efficiency). Identify the keys corresponding to pairs $P_j = B_i$ as candidate solution key pairs $K_1 = i$, $K_2 = j$ to the given problem. Confirm these by testing each key pair on a small number of additional known plaintext-ciphertext pairs as required.

While generally impractical due to the storage requirement, the attack of Fact 7.39 is referred to as a *certificational attack* on two-key triple encryption, demonstrating it to be weaker than triple encryption. This motivates consideration of triple-encryption with three independent keys, although a penalty is a third key to manage.

Fact 7.40, stated specifically for DES ($n = 64$, $k = 56$), indicates that for the price of additional computation, the memory requirement in Fact 7.39 may be reduced and the chosen-plaintext condition relaxed to known-plaintext. The attack, however, appears impractical even with extreme parallelization; for example, for $\lg t = 40$, the number of operations is still 2^{80}.

7.40 Fact If t known plaintext-ciphertext pairs are available, an attack on two-key triple-DES requires $O(t)$ space and $2^{120-\lg t}$ operations.

(iii) Multiple-encryption modes of operation

In contrast to the *single modes* of operation in Figure 7.1, *multiple modes* are variants of multiple encryption constructed by concatenating selected single modes. For example, the combination of three single-mode CBC operations provides *triple-inner-CBC*; an alternative is *triple-outer-CBC*, the composite operation of triple encryption (per Definition 7.32) with one outer ciphertext feedback after the sequential application of three single-ECB operations. With replicated hardware, multiple modes such as triple-inner-CBC may be pipelined allowing performance comparable to single encryption, offering an advantage over triple-outer-CBC. Unfortunately (Note 7.41), they are often less secure.

7.41 Note (*security of triple-inner-CBC*) Many multiple modes of operation are weaker than the corresponding multiple-ECB mode (i.e., multiple encryption operating as a black box with only outer feedbacks), and in some cases multiple modes (e.g., ECB-CBC-CBC) are not significantly stronger than single encryption. In particular, under some attacks triple-inner-CBC is significantly weaker than triple-outer-CBC; against other attacks based on the block size (e.g., Note 7.8), it appears stronger.

(iv) Cascade ciphers

Counter-intuitively, it is possible to devise examples whereby cascading of ciphers (Definition 7.29) actually reduces security. However, Fact 7.42 holds under a wide variety of attack models and meaningful definitions of "breaking".

7.42 Fact A cascade of n (independently keyed) ciphers is at least as difficult to break as the first component cipher. Corollary: for stage ciphers which commute (e.g., additive stream ciphers), a cascade is at least as strong as the strongest component cipher.

Fact 7.42 does not apply to product ciphers consisting of component ciphers which may have dependent keys (e.g., two-key triple-encryption); indeed, keying dependencies across stages may compromise security entirely, as illustrated by a two-stage cascade wherein the components are two binary additive stream ciphers using an identical keystream – in this case, the cascade output is the original plaintext.

Fact 7.42 may suggest the following practical design strategy: cascade a set of keystream generators each of which relies on one or more different design principles. It is not clear, however, if this is preferable to one large keystream generator which relies on a single principle. The cascade may turn out to be less secure for a fixed set of parameters (number of key bits, block size), since ciphers built piecewise may often be attacked piecewise.

7.3 Classical ciphers and historical development

The term *classical ciphers* refers to encryption techniques which have become well-known over time, and generally created prior to the second half of the twentieth century (in some cases, many hundreds of years earlier). Many classical techniques are variations of simple substitution and simple transposition. Some techniques that are not technically block ciphers are also included here for convenience and context.

Classical ciphers and techniques are presented under §7.3 for historical and pedagogical reasons only. They illustrate important basic principles and common pitfalls. However, since these techniques are neither sophisticated nor secure against current cryptanalytic capabilities, *they are not generally suitable for practical use*.

7.3.1 Transposition ciphers (background)

For a *simple transposition* cipher with fixed period t, encryption involves grouping the plaintext into blocks of t characters, and applying to each block a single permutation e on the numbers 1 through t. More precisely, the ciphertext corresponding to plaintext block $m = m_1 \dots m_t$ is $c = E_e(m) = m_{e(1)} \dots m_{e(t)}$. The encryption key is e, which implicitly defines t; the key space $\mathcal{K}$ has cardinality $t!$ for a given value t. Decryption involves use of the permutation d which inverts e. The above corresponds to Definition 1.32.

The mathematical notation obscures the simplicity of the encryption procedure, as is evident from Example 7.43.

7.43 Example (*simple transposition*) Consider a simple transposition cipher with $t = 6$ and $e = (6\ 4\ 1\ 3\ 5\ 2)$. The message $m =$ CAESAR is encrypted to $c =$ RSCEAA. Decryption uses the inverse permutation $d = (3\ 6\ 4\ 2\ 5\ 1)$. The transposition may be represented by a two-row matrix with the second indicating the position to which the element indexed by the corresponding number of the first is mapped to: $\binom{1\,2\,3\,4\,5\,6}{3\,6\,4\,2\,5\,1}$. Encryption may be done by writing a block of plaintext under headings "3 6 4 2 5 1", and then reading off the characters under the headings in numerical order. □

7.44 Note (*terminology: transposition vs. permutation*) While the term "transposition" is traditionally used to describe a transposition cipher, the mapping of Example 7.43 may alternately be called a *permutation* on the set $\{1, 2, \dots, 6\}$. The latter terminology is used, for example, in substitution-permutation networks, and in DES (§7.4).

A mnemonic keyword may be used in place of a key, although this may seriously decrease the key space entropy. For example, for $n = 6$, the keyword "CIPHER" could be used to specify the column ordering 1, 5, 4, 2, 3, 6 (by alphabetic priority).

7.45 Definition Sequential composition of two or more simple transpositions with respective periods $t_1, t_2, \dots, t_i$ is called a *compound transposition*.

7.46 Fact The compound transposition of Definition 7.45 is equivalent to a simple transposition of period $t = \mathrm{lcm}(t_1, \dots, t_i)$.

7.47 Note (*recognizing simple transposition*) Although simple transposition ciphers alter dependencies between consecutive characters, they are easily recognized because they preserve the frequency distribution of each character.

7.3.2 Substitution ciphers (background)

This section considers the following types of classical ciphers: simple (or mono-alphabetic) substitution, polygram substitution, and homophonic substitution. The difference between codes and ciphers is also noted. Polyalphabetic substitution ciphers are considered in §7.3.3.

(i) Mono-alphabetic substitution

Suppose the ciphertext and plaintext character sets are the same. Let $m = m_1m_2m_3\ldots$ be a plaintext message consisting of juxtaposed characters $m_i \in \mathcal{A}$, where $\mathcal{A}$ is some fixed character alphabet such as $\mathcal{A} = \{A, B, \ldots, Z\}$. A *simple substitution cipher* or *mono-alphabetic substitution cipher* employs a permutation e over $\mathcal{A}$, with encryption mapping $E_e(m) = e(m_1)e(m_2)e(m_3)\ldots$. Here juxtaposition indicates concatenation (rather than multiplication), and $e(m_i)$ is the character to which m_i is mapped by e. This corresponds to Definition 1.27.

7.48 Example (*trivial shift cipher/Caesar cipher*) A *shift cipher* is a simple substitution cipher with the permutation e constrained to an alphabetic shift through k characters for some fixed k. More precisely, if $|\mathcal{A}| = s$, and m_i is associated with the integer value i, $0 \le i \le s-1$, then $c_i = e(m_i) = m_i + k \bmod s$. The decryption mapping is defined by $d(c_i) = c_i - k \bmod s$. For English text, $s = 26$, and characters A through Z are associated with integers 0 through 25. For $k = 1$, the message $m =$ HAL is encrypted to $c =$ IBM. According to folklore, Julius Caesar used the key $k = 3$. □

The shift cipher can be trivially broken because there are only $s = |\mathcal{A}|$ keys (e.g., $s = 26$) to exhaustively search. A similar comment holds for affine ciphers (Example 7.49). More generally, see Fact 7.68.

7.49 Example (*affine cipher – historical*) The affine cipher on a 26-letter alphabet is defined by $e_K(x) = ax + b \bmod 26$, where $0 \le a, b \le 25$. The key is (a, b). Ciphertext $c = e_K(x)$ is decrypted using $d_K(c) = (c - b)a^{-1} \bmod 26$, with the necessary and sufficient condition for invertibility that $\gcd(a, 26) = 1$. Shift ciphers are a subclass defined by $a = 1$. □

7.50 Note (*recognizing simple substitution*) Mono-alphabetic substitution alters the frequency of individual plaintext characters, but does not alter the frequency distribution of the overall character set. Thus, comparing ciphertext character frequencies to a table of expected letter frequencies (unigram statistics) in the plaintext language allows associations between ciphertext and plaintext characters. (E.g., if the most frequent plaintext character X occurred twelve times, then the ciphertext character that X maps to will occur twelve times).

(ii) Polygram substitution

A simple substitution cipher substitutes for single plaintext letters. In contrast, *polygram substitution ciphers* involve groups of characters being substituted by other groups of characters. For example, sequences of two plaintext characters (*digrams*) may be replaced by other digrams. The same may be done with sequences of three plaintext characters (*trigrams*), or more generally using n-grams.

In full digram substitution over an alphabet of 26 characters, the key may be any of the 26^2 digrams, arranged in a table with row and column indices corresponding to the first and second characters in the digram, and the table entries being the ciphertext digrams substituted for the plaintext pairs. There are then $(26^2)!$ keys.

7.51 Example (*Playfair cipher – historical*) A digram substitution may be defined by arranging the characters of a 25-letter alphabet (I and J are equated) in a 5×5 matrix M. Adjacent plaintext characters are paired. The pair (p_1, p_2) is replaced by the digram (c_3, c_4) as follows. If p_1 and p_2 are in distinct rows and columns, they define the corners of a submatrix (possibly M itself), with the remaining corners c_3 and c_4; c_3 is defined as the character in the same column as p_1. If p_1 and p_2 are in a common row, c_3 is defined as the character immediately to the right of p_1 and c_4 that immediately right of p_2 (the first column is viewed as being to the right of the last). If p_1 and p_2 are in the same column, the characters immediately (circularly) below them are c_3 and c_4. If $p_1 = p_2$, an infrequent plaintext character (e.g., X) is inserted between them and the plaintext is re-grouped. While cryptanalysis based on single character frequencies fails for the Playfair cipher (each letter may be replaced by any other), cryptanalysis employing digram frequencies succeeds. □

The key for a Playfair cipher is the 5×5 square. A mnemonic aid may be used to more easily remember the square. An example is the use of a meaningful keyphrase, with repeated letters deleted and the remaining alphabet characters included alphabetically at the end. The keyphrase "PLAYFAIR IS A DIGRAM CIPHER" would define a square with rows PLAYF, IRSDG, MCHEB, KNOQT, VWXYZ. To avoid the trailing characters always being from the end of the alphabet, a further shift cipher (Example 7.48) could be applied to the resulting 25-character string.

Use of keyphrases may seriously reduce the key space entropy. This effect is reduced if the keyphrase is not directly written into the square. For example, the non-repeated keyphrase characters might be written into an 8-column rectangle (followed by the remaining alphabet letters), the trailing columns being incomplete. The 25-character string obtained by reading the columns vertically is then used to fill the 5×5 square row by row.

7.52 Example (*Hill cipher – historical*) An n-gram substitution may be defined using an invertible $n \times n$ matrix $A = a_{ij}$ as the key to map an n-character plaintext $m_1 \dots m_n$ to a ciphertext n-gram $c_i = \sum_{j=1}^{n} a_{ij} m_j$, $i = 1, \dots, n$. Decryption involves using A^{-1}. Here characters A–Z, for example, are associated with integers 0–25. This polygram substitution cipher is a linear transformation, and falls under known-plaintext attack. □

(iii) Homophonic substitution

The idea of homophonic substitution, introduced in §1.5, is for each fixed key k to associate with each plaintext unit (e.g., character) m a set $S(k, m)$ of potential corresponding ciphertext units (generally all of common size). To encrypt m under k, randomly choose one element from this set as the ciphertext. To allow decryption, for each fixed key this one-to-many encryption function must be injective on ciphertext space. Homophonic substitution results in ciphertext data expansion.

In homophonic substitution, $|S(k, m)|$ should be proportional to the frequency of m in the message space. The motivation is to smooth out obvious irregularities in the frequency distribution of ciphertext characters, which result from irregularities in the plaintext frequency distribution when simple substitution is used.

While homophonic substitution complicates cryptanalysis based on simple frequency distribution statistics, sufficient ciphertext may nonetheless allow frequency analysis, in conjunction with additional statistical properties of plaintext manifested in the ciphertext. For example, in long ciphertexts each element of $S(k, m)$ will occur roughly the same number of times. Digram distributions may also provide information.

(iv) Codes vs. ciphers

A technical distinction is made between *ciphers* and *codes*. Ciphers are encryption techniques which are applied to plaintext units (bits, characters, or blocks) independent of their semantic or linguistic meaning; the result is called ciphertext. In contrast, cryptographic codes operate on linguistic units such as words, groups of words, or phrases, and substitute (replace) these by designated words, letter groups, or number groups called *codegroups*. The key is a dictionary-like *codebook* listing plaintext units and their corresponding codegroups, indexed by the former; a corresponding codebook for decoding is reverse-indexed.

When there is potential ambiguity, codes in this context (vs. ciphers) may be qualified as *cryptographic codebooks*, to avoid confusion with error-correcting codes (EC-codes) used to detect and/or correct non-malicious errors and authentication codes (A-codes, or MACs as per Definition 9.7) which provide data origin authentication.

Several factors suggest that codes may be more difficult to break than ciphers: the key (codebook) is vastly larger than typical cipher keys; codes may result in data compression (cf. Fact 7.71); and statistical analysis is complicated by the large plaintext unit block size (cf. Note 7.74). Opposing this are several major disadvantages: the coding operation not being easily automated (relative to an algorithmic mapping); and identical encryption of repeated occurrences of plaintext units implies susceptibility to known-plaintext attacks, and allows frequency analysis based on observed traffic. This implies a need for frequent rekeying (changing the codebook), which is both more costly and inconvenient. Consequently, codes are not commonly used to secure modern telecommunications.

7.3.3 Polyalphabetic substitutions and Vigenère ciphers (historical)

A simple substitution cipher involves a single mapping of the plaintext alphabet onto ciphertext characters. A more complex alternative is to use different substitution mappings (called *multiple alphabets*) on various portions of the plaintext. This results in so-called *polyalphabetic substitution* (also introduced in Definition 1.30). In the simplest case, the different alphabets are used sequentially and then repeated, so the position of each plaintext character in the source string determines which mapping is applied to it. Under different alphabets, the same plaintext character is thus encrypted to different ciphertext characters, precluding simple frequency analysis as per mono-alphabetic substitution (§7.3.5).

The simple Vigenère cipher is a polyalphabetic substitution cipher, introduced in Example 1.31. The definition is repeated here for convenience.

7.53 Definition A *simple Vigenère* cipher of period t, over an s-character alphabet, involves a t-character key $k_1k_2\dots k_t$. The mapping of plaintext $m = m_1m_2m_3\dots$ to ciphertext $c = c_1c_2c_3\dots$ is defined on individual characters by $c_i = m_i + k_i \bmod s$, where subscript i in k_i is taken modulo t (the key is re-used).

The simple Vigenère uses t shift ciphers (see Example 7.48), defined by t shift values k_i, each specifying one of s (mono-alphabetic) substitutions; k_i is used on the characters in position i, $i + s$, $i + 2s$, ... In general, each of the t substitutions is different; this is referred to as using t alphabets rather than a single substitution mapping. The shift cipher (Example 7.48) is a simple Vigenère with period $t = 1$.

7.54 Example (*Beaufort variants of Vigenère*) Compared to the simple Vigenère mapping $c_i = m_i + k_i \bmod s$, the *Beaufort cipher* has $c_i = k_i - m_i \bmod s$, and is its own inverse. The *variant Beaufort* has encryption mapping $c_i = m_i - k_i \bmod s$. □

7.55 Example (*compound Vigenère*) The compound Vigenère has encryption mapping $c_i = m_i + (k_i^1 + k_i^2 + \cdots + k_i^r) \bmod s$, where in general the keys k^j, $1 \le j \le r$, have distinct periods t_j, and the subscript i in k_i^j, indicating the ith character of k^j, is taken modulo t_j. This corresponds to the sequential application of r simple Vigenères, and is equivalent to a simple Vigenère of period $\mathrm{lcm}(t_1, \dots, t_r)$. □

7.56 Example (*single mixed alphabet Vigenère*) A simple substitution mapping defined by a general permutation e (not restricted to an alphabetic shift), followed by a simple Vigenère, is defined by the mapping $c_i = e(m_i) + k_i \bmod s$, with inverse $m_i = e^{-1}(c_i - k_i) \bmod s$. An alternative is a simple Vigenère followed by a simple substitution: $c_i = e(m_i + k_i \bmod s)$, with inverse $m_i = e^{-1}(c_i) - k_i \bmod s$. □

7.57 Example (*full Vigenère*) In a simple Vigenère of period t, replace the mapping defined by the shift value k_i (for shifting character m_i) by a general permutation e_i of the alphabet. The result is the substitution mapping $c_i = e_i(m_i)$, where the subscript i in e_i is taken modulo t. The key consists of t permutations $e_1, \ldots, e_t$. □

7.58 Example (*running-key Vigenère*) If the keystream k_i of a simple Vigenère is as long as the plaintext, the cipher is called a *running-key cipher*. For example, the key may be meaningful text from a book. □

While running-key ciphers prevent cryptanalysis by the Kasiski method (§7.3.5), if the key has redundancy, cryptanalysis exploiting statistical imbalances may nonetheless succeed. For example, when encrypting plaintext English characters using a meaningful text as a running key, cryptanalysis is possible based on the observation that a significant proportion of ciphertext characters results from the encryption of high-frequency running text characters with high-frequency plaintext characters.

7.59 Fact A running-key cipher can be strengthened by successively enciphering plaintext under two or more distinct running keys. For typical English plaintext and running keys, it can be shown that iterating four such encipherments appears unbreakable.

7.60 Definition An *auto-key cipher* is a cipher wherein the plaintext itself serves as the key (typically subsequent to the use of an initial priming key).

7.61 Example (*auto-key Vigenère*) In a running-key Vigenère (Example 7.58) with an s-character alphabet, define a *priming key* $k = k_1k_2 \ldots k_t$. Plaintext characters m_i are encrypted as $c_i = m_i + k_i \bmod s$ for $1 \le i \le t$ (simplest case: $t = 1$). For $i > t$, $c_i = (m_i + m_{i-t}) \bmod s$. An alternative involving more keying material is to replace the simple shift by a full Vigenère with permutations e_i, $1 \le i \le s$, defined by the key k_i or character m_i: for $1 \le i \le t$, $c_i = e_{k_i}(m_i)$, and for $i > t$, $c_i = e_{m_{i-t}}(m_i)$. □

An alternative to Example 7.61 is to auto-key a cipher using the resulting ciphertext as the key: for example, for $i > t$, $c_i = (m_i + c_{i-t}) \bmod s$. This, however, is far less desirable, as it provides an eavesdropping cryptanalyst the key itself.

7.62 Example (*Vernam viewed as a Vigenère*) Consider a simple Vigenère defined by $c_i = m_i + k_i \bmod s$. If the keystream is truly random and independent – as long as the plaintext and never repeated (cf. Example 7.58) – this yields the unconditionally secure Vernam cipher (Definition 1.39; §6.1.1), generalized from a binary to an arbitrary alphabet. □

7.3.4 Polyalphabetic cipher machines and rotors (historical)

The *Jefferson cylinder* is a deceptively simple device which implements a polyalphabetic substitution cipher; conceived in the late 18th century, it had remarkable cryptographic strength for its time. Polyalphabetic substitution ciphers implemented by a class of rotor-based machines were the dominant cryptographic tool in World War II. Such machines, including the Enigma machine and those of Hagelin, have an alphabet which changes continuously for a very long period before repeating; this provides protection against Kasiski analysis and methods based on the index of coincidence (§7.3.5).

(i) Jefferson cylinder

The *Jefferson cylinder* (Figure 7.3) implements a polyalphabetic substitution cipher while avoiding complex machinery, extensive user computations, and Vigenère tableaus. A solid cylinder 6 inches long is sliced into 36 disks. A rod inserted through the cylinder axis allows the disks to rotate. The periphery of each disk is divided into 26 parts. On each disk, the letters A–Z are inscribed in a (different) random ordering. Plaintext messages are encrypted in 36-character blocks. A reference bar is placed along the cylinder's length. Each of the 36 wheels is individually rotated to bring the appropriate character (matching the plaintext block) into position along the reference line. The 25 other parallel reference positions then each define a ciphertext, from which (in an early instance of randomized encryption) one is selected as the ciphertext to transmit.

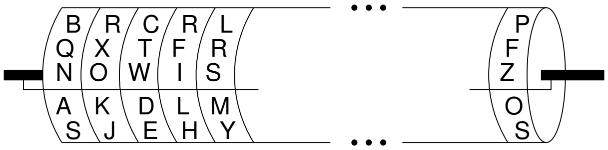

Figure 7.3: *The Jefferson cylinder.*

The second party possesses a cylinder with identically marked and ordered disks (1–36). The ciphertext is decrypted by rotating each of the 36 disks to obtain characters along a fixed reference line matching the ciphertext. The other 25 reference positions are examined for a recognizable plaintext. If the original message is not recognizable (e.g., random data), both parties agree beforehand on an index 1 through 25 specifying the offset between plaintext and ciphertext lines.

To accommodate plaintext digits 0–9 without extra disk sections, each digit is permanently assigned to one of 10 letters (a,e,i,o,u,y and f,l,r,s) which is encrypted as above but annotated with an overhead dot, identifying that the procedure must be reversed. Reordering disks (1 through 36) alters the polyalphabetic substitution key. The number of possible orderings is $36! \approx 3.72 \times 10^{41}$. Changing the ordering of letters on each disk affords 25! further mappings (per disk), but is more difficult in practice.

(ii) Rotor-based machines – technical overview

A simplified generic rotor machine (Figure 7.4) consists of a number of *rotors* (*wired codewheels*) each implementing a different fixed mono-alphabetic substitution, mapping a character at its input face to one on its output face. A plaintext character input to the first rotor generates an output which is input to the second rotor, and so on, until the final ciphertext character emerges from the last. For fixed rotor positions, the bank of rotors collectively implements a mono-alphabetic substitution which is the composition of the substitutions defined by the individual rotors.

To provide polyalphabetic substitution, the encipherment of each plaintext character causes various rotors to move. The simplest case is an odometer-like movement, with a single rotor stepped until it completes a full revolution, at which time it steps the adjacent

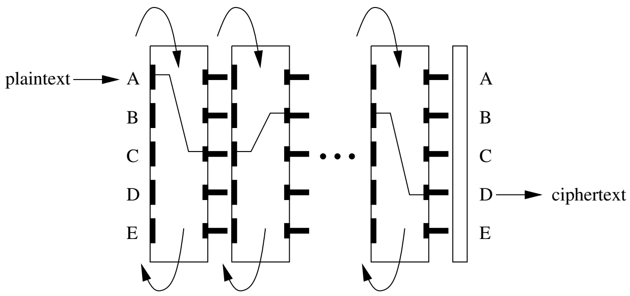

Figure 7.4: *A rotor-based machine.*

rotor one position, and so on. Stepping a rotor changes the mono-alphabetic substitution it defines (the *active* mapping). More precisely, each rotor R_i effects a mono-alphabetic substitution f_i. R_i can rotate into t_i positions (e.g., $t_i = 26$). When offset j places from a reference setting, R_i maps input a to $f_i(a - j) + j$, where both the input to f_i and the final output are reduced mod 26.

The cipher key is defined by the mono-alphabetic substitutions determined by the fixed wheel wirings and initial rotor positions. Re-arranging the order of rotors provides additional variability. Providing a machine with more rotors than necessary for operation at any one time allows further keying variation (by changing the active rotors).

7.63 Fact Two properties of rotor machines desirable for security-related reasons are: (1) long periods; and (2) state changes which are almost all "large".

The second property concerns the motion of rotors relative to each other, so that the sub-mappings between rotor faces change when the state changes. Rotor machines with odometer-like state changes fail to achieve this second property.

7.64 Note (*rotor machine output methods*) Rotor machines were categorized by their method of providing ciphertext output. In *indicating machines*, ciphertext output characters are indicated by means such as lighted lamps or displayed characters in output apertures. In *printing machines*, ciphertext is printed or typewritten onto an output medium such as paper. With *on-line machines*, output characters are produced in electronic form suitable for direct transmission over telecommunications media.

(iii) Rotor-based machines – historical notes

A number of individuals are responsible for the development of early machines based on rotor principles. In 1918, the American E.H. Hebern built the first rotor apparatus, based on an earlier typewriting machine modified with wired connections to generate a mono-alphabetic substitution. The output was originally by lighted indicators. The first rotor patent was filed in 1921, the year *Hebern Electric Code, Inc.* became the first U.S. cipher machine company (and first to bankrupt in 1926). The U.S. Navy (circa 1929-1930 and some years thereafter) used a number of Hebern's five-rotor machines.

In October 1919, H.A. Koch filed Netherlands patent no.10,700 ("Geheimschrijfmachine" – secret writing machine), demonstrating a deep understanding of rotor principles; no machine was built. In 1927, the patent rights were assigned to A. Scherbius.

The German inventor Scherbius built a rotor machine called the *Enigma*. Model A was replaced by Model B with typewriter output, and a portable Model C with indicator lamps.

The company set up in 1923 dissolved in 1934, but thereafter the Germans used the portable battery-powered Enigma, including for critical World War II operations.

In October 1919, three days after Koch, A.G. Damm filed Swedish patent no.52,279 describing a double-rotor device. His firm was joined by the Swede, B. Hagelin, whose 1925 modification yielded the B-21 rotor machine (with indicating lamps) used by the Swedish army. The B-21 had *keywheels* with varying number of teeth or gears, each of which was associated with a settable two-state pin. The period of the resulting polyalphabetic substitution was the product of the numbers of keywheel pins; the key was defined by the state of each pin and the initial keywheel positions. Hagelin later produced other models: B-211 (a printing machine); a more compact (phone-sized) model C-36 for the French in 1934; and based on alterations suggested by Friedman and others, model C-48 (of which over 140 000 were produced) which was called M-209 when used by the U.S. Army as a World War II field cipher. His 1948 Swiss factory later produced: model C-52, a strengthened version of M-209 (C-48) with period exceeding 2.75×10^9 (with keywheels of 47, 43, 41, 37, 31, 29 pins); CD-55, a pocket-size version of the C-52; and T-55, an on-line version of the same, modifiable to use a one-time tape. A further model was CD-57.

7.65 Note (*Enigma details*) The Enigma initially had three rotors R_i, each with 26 positions. R_1 stepped R_2 which stepped R_3 odometer-like, with R_2 also stepping itself; the period was $26 \cdot 25 \cdot 26 \approx 17\,000$. The key consisted of the initial positions of these rotors ($\approx 17\,000$ choices), their order ($3! = 6$ choices), and the state of a *plugboard*, which implemented a fixed but easily changed (e.g., manually, every hour) mono-alphabetic substitution (26! choices), in addition to that carried out by rotor combinations.

7.66 Note (*Hagelin M-209 details*) The Hagelin M-209 rotor machine implements a polyalphabetic substitution using 6 keywheels – more specifically, a self-decrypting Beaufort cipher (Example 7.54), $E_{k_i}(m_i) = k_i - m_i \bmod 26$, of period $101\,405\,850 = 26\cdot25\cdot23\cdot21\cdot19\cdot17$ letters. Thus for a fixed ordered set of 6 keywheels, the cipher period exceeds 10^8. k_i may be viewed as the ith character in the key stream, as determined by a particular ordering of keywheels, their pin settings, and starting positions. All keywheels rotate one position forward after each character is enciphered. The wheels simultaneously return to their initial position only after a period equal to the least-common-multiple of their gear-counts, which (since these are co-prime) is their product. A ciphertext-only attack is possible with 1000-2000 characters, using knowledge of the machine's internal mechanical details, and assuming natural language redundancy in the plaintext; a known-plaintext attack is possible with 50-100 characters.

7.3.5 Cryptanalysis of classical ciphers (historical)

This section presents background material on redundancy and unicity distance, and techniques for cryptanalysis of classical ciphers,

(i) Redundancy

All natural languages are redundant. This redundancy results from linguistic structure. For example, in English the letter "E" appears far more frequently than "Z", "Q" is almost always followed by "U", and "TH" is a common digram.

An alphabet with 26 characters (e.g., Roman alphabet) can theoretically carry up to $\lg 26 = 4.7$ bits of information per character. Fact 7.67 indicates that, on average, far less information is actually conveyed by a natural language.

7.67 Fact The estimated average amount of information carried per character (per-character entropy) in meaningful English alphabetic text is 1.5 bits.

The per-character redundancy of English is thus about $4.7 - 1.5 = 3.2$ bits.

7.68 Fact Empirical evidence suggests that, for essentially any simple substitution cipher on a meaningful message (e.g., with redundancy comparable to English), as few as 25 ciphertext characters suffices to allow a skilled cryptanalyst to recover the plaintext.

(ii) Unicity distance and random cipher model

7.69 Definition The *unicity distance* of a cipher is the minimum amount of ciphertext (number of characters) required to allow a computationally unlimited adversary to recover the unique encryption key.

The unicity distance is primarily a theoretical measure, useful in relation to unconditional security. A small unicity distance does not necessarily imply that a block cipher is insecure in practice. For example, consider a 64-bit block cipher with a unicity distance of two ciphertext blocks. It may still be computationally infeasible for a cryptanalyst (of reasonable but bounded computing power) to recover the key, although theoretically there is sufficient information to allow this.

The random cipher model (Definition 7.70) is a simplified model of a block cipher providing a reasonable approximation for many purposes, facilitating results on block cipher properties not otherwise easily established (e.g., Fact 7.71).

7.70 Definition Let C and K be random variables, respectively, denoting the ciphertext block and the key, and let D denote the decryption function. Under the *random cipher model*, $D_K(C)$ is a random variable uniformly distributed over all possible pre-images of C (meaningful messages and otherwise, with and without redundancy).

In an intuitive sense, a random cipher as per the model of Definition 7.70 is a random mapping. (A more precise approximation would be as a random permutation.)

7.71 Fact Under the random cipher model, the expected unicity distance N_0 of a cipher is $N_0 = H(\mathcal{K})/D$, where $H(\mathcal{K})$ is the entropy of the key space (e.g., 64 bits for 2^{64} equiprobable keys), and D is the plaintext redundancy (in bits/character).

For a one-time pad, the unbounded entropy of the key space implies, by Fact 7.71, that the unicity distance is likewise unbounded. This is consistent with the one-time pad being theoretically unbreakable.

Data compression reduces redundancy. Fact 7.71 implies that data compression prior to encryption increases the unicity distance, thus increasing security. If the plaintext contains no redundancy whatsoever, then the unicity distance is infinite; that is, the system is theoretically unbreakable under a ciphertext-only attack.

7.72 Example (*unicity distance – transposition cipher*) The unicity distance of a simple transposition cipher of period t can be estimated under the random cipher model using Fact 7.71, and the assumption of plaintext redundancy of $D = 3.2$ bits/character. In this case, $H(\mathcal{K})/D = \lg(t!)/3.2$ and for $t = 12$ the estimated unicity distance is 9 characters, which is very crude, this being less than one 12-character block. For $t = 27$, the estimated unicity distance is a more plausible 29 characters; this can be computed using Stirling's approximation of Fact 2.57(iii) ($t! \approx \sqrt{2\pi t}(t/e)^t$, for large t and $e = 2.718$) as $H(\mathcal{K})/D = \lg(t!)/3.2 \approx (0.3t) \cdot \lg(t/e)$. □

7.73 Example (*unicity distance – simple substitution*) The number of keys for a mono-alphabetic substitution cipher over alphabet $\mathcal{A}$ is $|\mathcal{K}| = s!$, where $s = |\mathcal{A}|$. For example, $s = 26$ (Roman alphabet) yields $26! \approx 4 \times 10^{26}$ keys. Assuming equiprobable keys, an estimate of the entropy of the key space is then (cf. Example 7.72) $H(\mathcal{K}) = \lg(26!) \approx 88.4$ bits. Assuming English text with $D = 3.2$ bits of redundancy per character (Fact 7.67), a theoretical estimate of the unicity distance of a simple substitution cipher is $H(\mathcal{K})/D = 88.4/3.2 \approx 28$ characters. This agrees closely with empirical evidence (Fact 7.68). □

(iii) Language statistics

Cryptanalysis of classical ciphers typically relies on redundancy in the source language (plaintext). In many cases a divide-and-conquer approach is possible, whereby the plaintext or key is recovered piece by piece, each facilitating further recovery.

Mono-alphabetic substitution on short plaintext blocks (e.g., Roman alphabet characters) is easily defeated by associating ciphertext characters with plaintext characters (Note 7.50). The frequency distribution of individual ciphertext characters can be compared to that of single characters in the source language, as given by Figure 7.5 (estimated from 1964 English text). This is facilitated by grouping plaintext letters by frequency into high, medium, low, and rare classes; focussing on the high-frequency class, evidence supporting trial letter assignments can be obtained by examining how closely hypothesized assignments match those of the plaintext language. Further evidence is available by examination of digram and trigram frequencies. Figure 7.6 gives the most common English digrams as a percentage of all digrams; note that of $26^2 = 676$ possible digrams, the top 15 account for 27% of all occurrences. Other examples of plaintext redundancy appearing in the ciphertext include associations of vowels with consonants, and repeated letters in *pattern words* (e.g., “that”, “soon”, “three”).

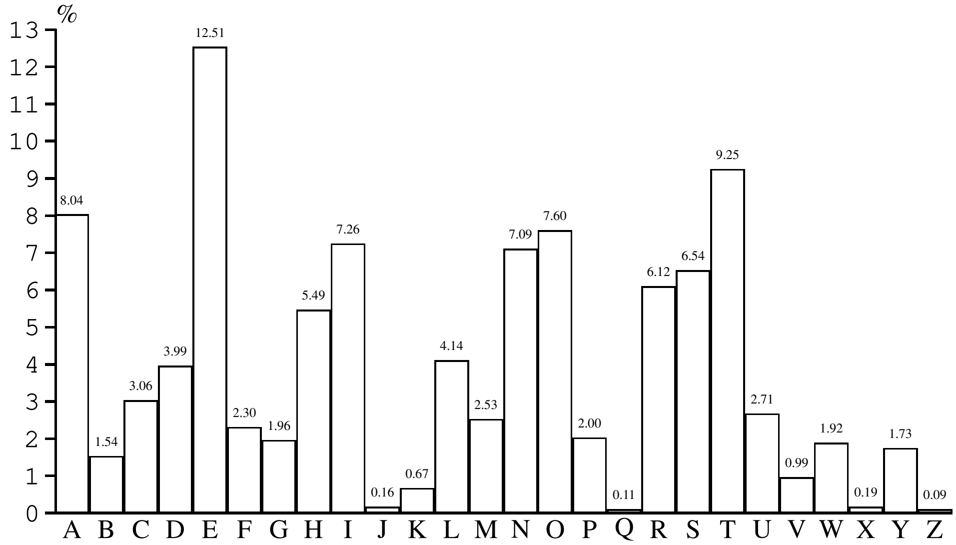

Figure 7.5: *Frequency of single characters in English text.*

7.74 Note (*large blocks preclude statistical analysis*) An n-bit block size implies 2^n plaintext units (“characters”). Compilation of frequency statistics on plaintext units thus becomes infeasible as the block size of the simple substitution increases; for example, this is clearly infeasible for DES (§7.4), where $n = 64$.

Cryptanalysis of simple transposition ciphers is similarly facilitated by source language statistics (see Note 7.47). Cryptanalyzing transposed blocks resembles solving an anagram. Attempts to reconstruct common digrams and trigrams are facilitated by frequency statistics. Solutions may be constructed piecewise, with the appearance of digrams and trigrams in trial decryptions confirming (partial) success.

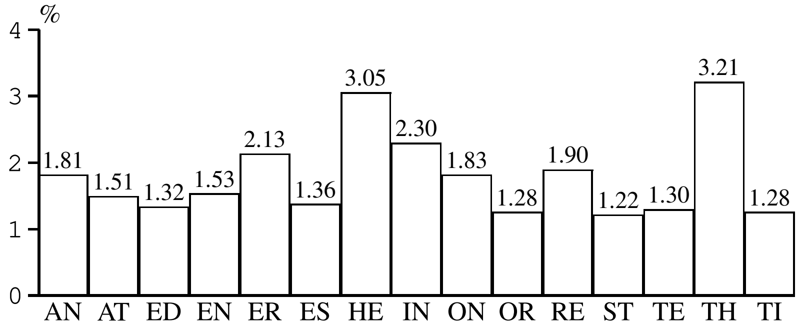

Figure 7.6: *Frequency of 15 common digrams in English text.*

Cryptanalysis of polyalphabetic ciphers is possible by various methods, including Kasiski's method and methods based on the index of coincidence, as discussed below.

(iv) Method of Kasiski (vs. polyalphabetic substitution)

Kasiski's method provides a general technique for cryptanalyzing polyalphabetic ciphers with repeated keywords, such as the simple Vigenère cipher (Definition 7.53), based on the following observation: repeated portions of plaintext encrypted with the same portion of the keyword result in identical ciphertext segments. Consequently one expects the number of characters between the beginning of repeated ciphertext segments to be a multiple of the keyword length. Ideally, it suffices to compute the greatest common divisor of the various distances between such repeated segments, but coincidental repeated ciphertext segments may also occur. Nonetheless, an analysis (*Kasiski examination*) of the common factors among all such distances is possible; the largest factor which occurs most commonly is the most likely keyword length. Repeated ciphertext segments of length 4 or longer are most useful, as coincidental repetitions are then less probable.

The number of letters in the keyword indicates the number of alphabets t in the polyalphabetic substitution. Ciphertext characters can then be partitioned into t sets, each of which is then the result of a mono-alphabetic substitution. Trial values for t are confirmed if the frequency distribution of the (candidate) mono-alphabetic groups matches the frequency distribution of the plaintext language. For example, the profile for plaintext English (Figure 7.5) exhibits a long trough characterizing $uvwxyz$, followed by a spike at a, and preceded by the triple-peak of rst. The resulting mono-alphabetic portions can be solved individually, with additional information available by combining their solution (based on digrams, probable words, etc.). If the source language is unknown, comparing the frequency distribution of ciphertext characters to that of candidate languages may allow determination of the source language itself.

(v) Index of coincidence (vs. polyalphabetic substitution)

The *index of coincidence* (*IC*) is a measure of the relative frequency of letters in a ciphertext sample, which facilitates cryptanalysis of polyalphabetic ciphers by allowing determination of the period t (as an alternative to Kasiski's method). For concreteness, consider a Vigènere cipher and assume natural language English plaintext.

Let the ciphertext alphabet be $\{a_0, a_1, \ldots, a_{n-1}\}$, and let p_i be the unknown probability that an arbitrarily chosen character in a random ciphertext is a_i. The *measure of roughness* measures the deviation of ciphertext characters from a flat frequency distribution as follows:

$$\text{MR} = \sum_{i=0}^{n-1} \left(p_i - \frac{1}{n}\right)^2 = \sum_{i=0}^{n-1} {p_i}^2 - \frac{1}{n} \tag{7.1}$$

The minimum value is $\text{MR}_{\min} = 0$, corresponding to a flat distribution (for equiprobable a_i, $p_i = 1/n$). The maximum value occurs when the frequency distribution of p_i has greatest variability, corresponding to a mono-alphabetic substitution (the plaintext frequency distribution is then manifested). Define this maximum value $\text{MR}_{\max} = \kappa_p - 1/n$, where κ_p corresponds to $\sum {p_i}^2$ when p_i are plaintext frequencies. For English as per Figure 7.5, the maximum value is $\text{MR} = \kappa_p - 1/n \approx 0.0658 - 0.0385 = 0.0273$. (This varies with letter frequency estimates; $\kappa_p = 0.0667$, yielding $\kappa_p - 1/n = 0.0282$ is commonly cited, and is used in Table 7.1.) While MR cannot be computed directly from a ciphertext sample (since the period t is unknown, the mono-alphabetic substitutions cannot be separated), it may be estimated from the frequency distribution of ciphertext characters as follows.

Let f_i denote the number of appearances of a_i in an L-character ciphertext sample (thus $\sum f_i = L$). The number of pairs of letters among these L is $L(L-1)/2$, of which $f_i(f_i - 1)/2$ are the pair (a_i, a_i) for any fixed character a_i. Define IC as the probability that two characters arbitrarily chosen from the *given* ciphertext sample are equal:

$$\text{IC} = \frac{\sum_{i=0}^{n-1} \binom{f_i}{2}}{\binom{L}{2}} = \frac{\sum_{i=0}^{n-1} f_i(f_i - 1)}{L(L-1)} \tag{7.2}$$

Independent of this given ciphertext sample, the probability that two randomly chosen ciphertext characters are equal is $\sum_{i=0}^{n-1} {p_i}^2$. Thus (comparing word definitions) IC is an estimate of $\sum {p_i}^2$, and by equation (7.1), thereby an estimate of MR + $1/n$. Moreover, IC can be directly computed from a ciphertext sample, allowing estimation of MR itself. Since MR varies from 0 to $\kappa_p - 1/n$, one expects IC to range from $1/n$ (for polyalphabetic substitution with infinite period) to κ_p (for mono-alphabetic substitution). More precisely, the following result may be established.

7.75 Fact For a polyalphabetic cipher of period t, $E(\text{IC})$ as given below is the expected value of the index of coincidence for a ciphertext string of length L, where n is the number of alphabet characters, $\kappa_r = 1/n$, and κ_p is given in Table 7.1:

$$E(\text{IC}) = \frac{1}{t} \cdot \frac{L-t}{L-1} \cdot \kappa_p + \frac{t-1}{t} \cdot \frac{L}{L-1} \cdot \kappa_r \tag{7.3}$$

(p in κ_p is intended to denote a plaintext frequency distribution, while the r in κ_r denotes a distribution for random characters.) For Roman-alphabet languages, $n = 26$ implies $\kappa_r = 0.03846$; for the Russian Cyrillic alphabet, $n = 30$.

7.76 Example (*estimating polyalphabetic period using IC*) Tabulating the expected values for IC for periods $t = 1, 2, \ldots$ using Equation (7.3) (which is essentially independent of L for large L and small t), and comparing this to that obtained from a particular ciphertext using Equation (7.2) allows a crude estimate of the period t of the cipher, e.g., whether it is mono-alphabetic or polyalphabetic with small period. Candidate values t in the range thus determined may be tested for correctness by partitioning ciphertext characters into groups of letters separated by t ciphertext positions, and in one or more such groups, comparing the character frequency distribution to that of plaintext. □

Language	κ_p
French	0.0778
Spanish	0.0775
German	0.0762
Italian	0.0738
English	0.0667
Russian	0.0529

Table 7.1: *Estimated roughness constant κ_p for various languages (see Fact 7.75).*

A polyalphabetic period t may be determined either by Example 7.76 or the alternative of Example 7.77, based on the same underlying ideas. Once t is determined, the situation is as per after successful completion of the Kasiski method.

7.77 Example (*determining period by ciphertext auto-correlation*) Given a sample of polyalphabetic ciphertext, the unknown period t may be determined by examining the number of coincidences when the ciphertext is auto-correlated. More specifically, given a ciphertext sample $c_1c_2\ldots c_L$, starting with $t = 1$, count the total number of occurrences $c_i = c_{i+t}$ for $1 \le i \le L - t$. Repeat for $t = 2, 3, \ldots$ and tabulate the counts (or plot a bar graph). The actual period t^* is revealed as follows: for values t that are a multiple of t^*, the counts will be noticeably higher (easily recognized as spikes on the bar graph). In fact, for L appropriately large, one expects approximately $L \cdot \kappa_p$ coincidences in this case, and significantly fewer in other cases. □

In the auto-correlation method of coincidences of Example 7.77, the spikes on the bar graph reveal the period, independent of the source language. Once the period is determined, ciphertext characters from like alphabets can be grouped, and the profile of single-character letter frequencies among these, which differs for each language, may be used to determine the plaintext language.

7.4 DES

The Data Encryption Standard (DES) is the most well-known symmetric-key block cipher. Recognized world-wide, it set a precedent in the mid 1970s as the first commercial-grade modern algorithm with openly and fully specified implementation details. It is defined by the American standard FIPS 46–2.

7.4.1 Product ciphers and Feistel ciphers

The design of DES is related to two general concepts: product ciphers and Feistel ciphers. Each involves iterating a common sequence or round of operations.

The basic idea of a product cipher (see §1.5.3) is to build a complex encryption function by composing several simple operations which offer complementary, but individually insufficient, protection (note cascade ciphers per Definition 7.29 use independent keys). Basic operations include transpositions, translations (e.g., XOR) and linear transformations, arithmetic operations, modular multiplication, and simple substitutions.

7.78 Definition A *product cipher* combines two or more transformations in a manner intending that the resulting cipher is more secure than the individual components.

7.79 Definition A *substitution-permutation* (SP) *network* is a product cipher composed of a number of stages each involving substitutions and permutations (Figure 7.7).

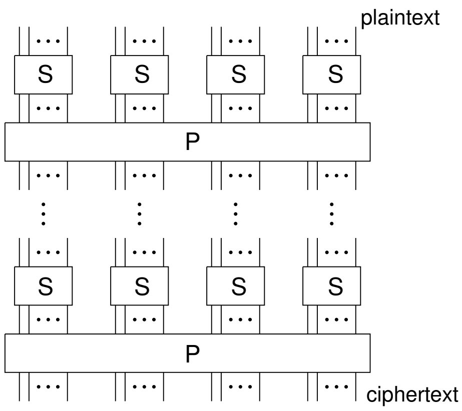

Figure 7.7: *Substitution-permutation (SP) network.*

Many SP networks are iterated ciphers as per Definition 7.80.

7.80 Definition An *iterated block cipher* is a block cipher involving the sequential repetition of an internal function called a *round function*. Parameters include the number of rounds r, the block bitsize n, and the bitsize k of the input key K from which r *subkeys* K_i (round keys) are derived. For invertibility (allowing unique decryption), for each value K_i the round function is a bijection on the round input.

7.81 Definition A *Feistel cipher* is an iterated cipher mapping a $2t$-bit plaintext (L_0, R_0), for t-bit blocks L_0 and R_0, to a ciphertext (R_r, L_r), through an r-round process where $r \geq 1$. For $1 \leq i \leq r$, round i maps $(L_{i-1}, R_{i-1}) \xrightarrow{K_i} (L_i, R_i)$ as follows: $L_i = R_{i-1}$, $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$, where each *subkey* K_i is derived from the cipher key K.

Typically in a Feistel cipher, $r \geq 3$ and often is even. The Feistel structure specifically orders the ciphertext output as (R_r, L_r) rather than (L_r, R_r); the blocks are exchanged from their usual order after the last round. Decryption is thereby achieved using the same r-round process but with subkeys used in reverse order, K_r through K_1; for example, the last round is undone by simply repeating it (see Note 7.84). The f function of the Feistel cipher may be a product cipher, though f itself need not be invertible to allow inversion of the Feistel cipher.

Figure 7.9(b) illustrates that successive rounds of a Feistel cipher operate on alternating halves of the ciphertext, while the other remains constant. Note the round function of Definition 7.81 may also be re-written to eliminate L_i: $R_i = R_{i-2} \oplus f(R_{i-1}, K_i)$. In this case, the final ciphertext output is (R_r, R_{r-1}), with input labeled (R_{-1}, R_0).

7.4.2 DES algorithm

DES is a Feistel cipher which processes plaintext blocks of $n = 64$ bits, producing 64-bit ciphertext blocks (Figure 7.8). The effective size of the secret key K is $k = 56$ bits; more precisely, the input key K is specified as a 64-bit key, 8 bits of which (bits $8, 16, \ldots, 64$) may be used as parity bits. The 2^{56} keys implement (at most) 2^{56} of the $2^{64}!$ possible bijections on 64-bit blocks. A widely held belief is that the parity bits were introduced to reduce the effective key size from 64 to 56 bits, to intentionally reduce the cost of exhaustive key search by a factor of 256.

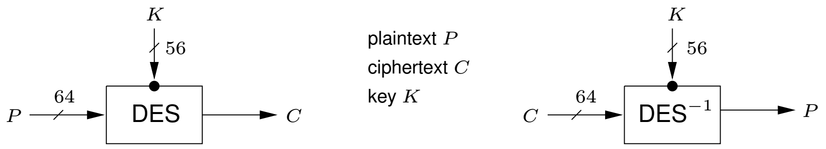

Figure 7.8: *DES input-output.*

Full details of DES are given in Algorithm 7.82 and Figures 7.9 and 7.10. An overview follows. Encryption proceeds in 16 stages or *rounds*. From the input key K, sixteen 48-bit subkeys K_i are generated, one for each round. Within each round, 8 fixed, carefully selected 6-to-4 bit substitution mappings (*S-boxes*) S_i, collectively denoted S, are used. The 64-bit plaintext is divided into 32-bit halves L_0 and R_0. Each round is functionally equivalent, taking 32-bit inputs L_{i-1} and R_{i-1} from the previous round and producing 32-bit outputs L_i and R_i for $1 \le i \le 16$, as follows:

$$L_i = R_{i-1}; \tag{7.4}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i), \text{ where } f(R_{i-1}, K_i) = P(S(E(R_{i-1}) \oplus K_i)) \tag{7.5}$$

Here E is a fixed expansion permutation mapping R_{i-1} from 32 to 48 bits (all bits are used once; some are used twice). P is another fixed permutation on 32 bits. An initial bit permutation (IP) precedes the first round; following the last round, the left and right halves are exchanged and, finally, the resulting string is bit-permuted by the inverse of IP. Decryption involves the same key and algorithm, but with subkeys applied to the internal rounds in the reverse order (Note 7.84).

A simplified view is that the right half of each round (after expanding the 32-bit input to 8 characters of 6 bits each) carries out a key-dependent substitution on each of 8 characters, then uses a fixed bit transposition to redistribute the bits of the resulting characters to produce 32 output bits.

Algorithm 7.83 specifies how to compute the DES round keys K_i, each of which contains 48 bits of K. These operations make use of tables PC1 and PC2 of Table 7.4, which are called *permuted choice 1* and *permuted choice 2*. To begin, 8 bits ($k_8, k_{16}, \ldots, k_{64}$) of K are discarded (by PC1). The remaining 56 bits are permuted and assigned to two 28-bit variables C and D; and then for 16 iterations, both C and D are rotated either 1 or 2 bits, and 48 bits (K_i) are selected from the concatenated result.

7.82 Algorithm Data Encryption Standard (DES)

INPUT: plaintext $m_1 \dots m_{64}$; 64-bit key $K = k_1 \dots k_{64}$ (includes 8 parity bits).
OUTPUT: 64-bit ciphertext block $C = c_1 \dots c_{64}$. (For decryption, see Note 7.84.)

1. (key schedule) Compute sixteen 48-bit round keys K_i from K using Algorithm 7.83.
2. $(L_0, R_0) \leftarrow \text{IP}(m_1 m_2 \dots m_{64})$. (Use IP from Table 7.2 to permute bits; split the result into left and right 32-bit halves $L_0 = m_{58} m_{50} \dots m_8$, $R_0 = m_{57} m_{49} \dots m_7$.)
3. (16 rounds) for i from 1 to 16, compute L_i and R_i using Equations (7.4) and (7.5) above, computing $f(R_{i-1},\ K_i) = P(S(E(R_{i-1}) \oplus K_i))$ as follows:
   (a) Expand $R_{i-1} = r_1 r_2 \dots r_{32}$ from 32 to 48 bits using E per Table 7.3: $T \leftarrow E(R_{i-1})$. (Thus $T = r_{32} r_1 r_2 \dots r_{32} r_1$.)
   (b) $T' \leftarrow T \oplus K_i$. Represent T' as eight 6-bit character strings: $(B_1, \dots, B_8) = T'$.
   (c) $T'' \leftarrow (S_1(B_1), S_2(B_2), \dots S_8(B_8))$. (Here $S_i(B_i)$ maps $B_i = b_1 b_2 \dots b_6$ to the 4-bit entry in row r and column c of S_i in Table 7.8, page 260 where $r = 2 \cdot b_1 + b_6$, and $b_2 b_3 b_4 b_5$ is the radix-2 representation of $0 \le c \le 15$. Thus $S_1(011011)$ yields $r = 1$, $c = 13$, and output 5, i.e., binary 0101.)
   (d) $T''' \leftarrow P(T'')$. (Use P per Table 7.3 to permute the 32 bits of $T'' = t_1 t_2 \dots t_{32}$, yielding $t_{16} t_7 \dots t_{25}$.)
4. $b_1 b_2 \dots b_{64} \leftarrow (R_{16}, L_{16})$. (Exchange final blocks L_{16}, R_{16}.)
5. $C \leftarrow \text{IP}^{-1}(b_1 b_2 \dots b_{64})$. (Transpose using IP^{-1} from Table 7.2; $C = b_{40} b_8 \dots b_{25}$.)

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Table 7.2: *DES initial permutation and inverse (IP and IP^{-1}).*

E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Table 7.3: *DES per-round functions: expansion E and permutation P.*

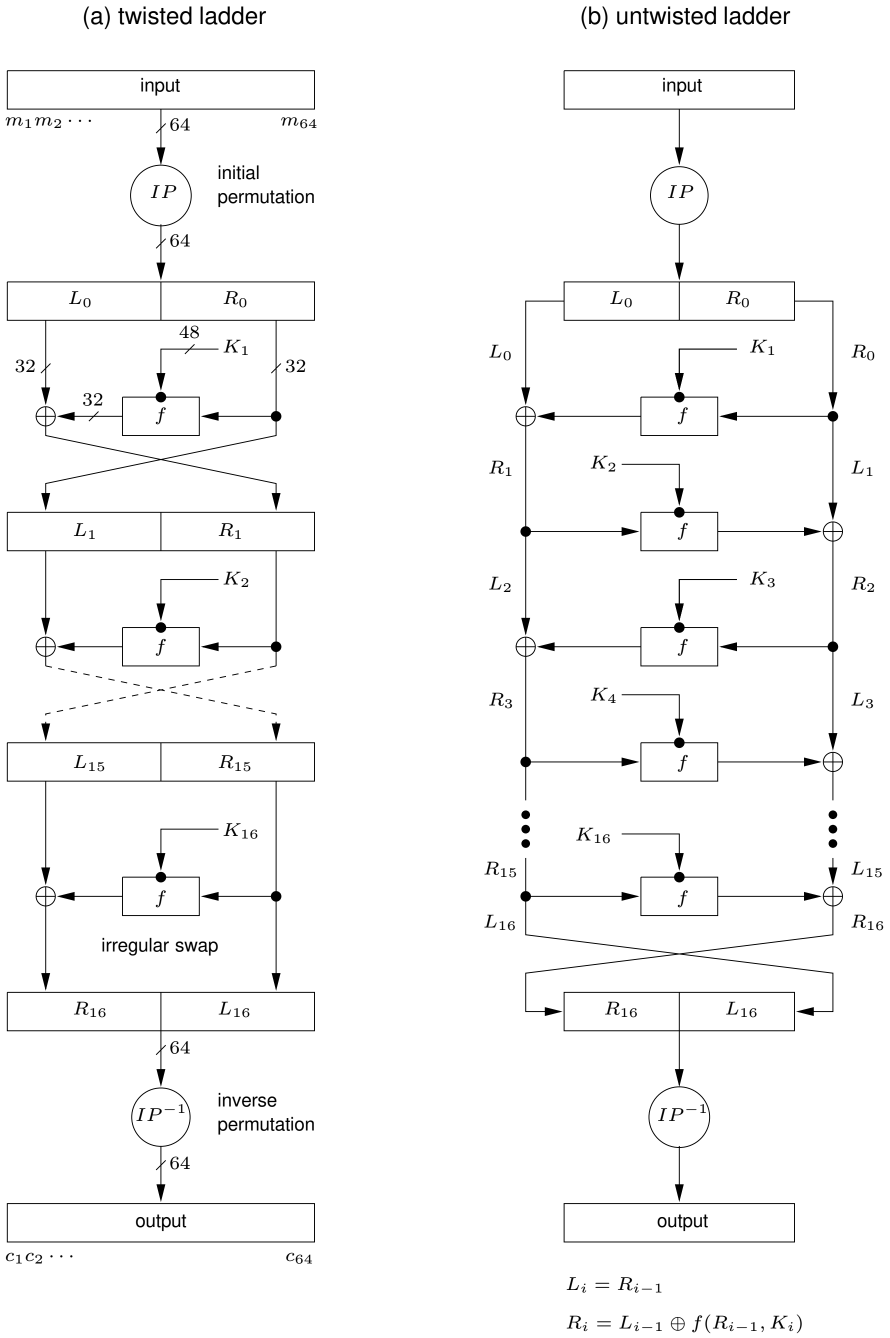

Figure 7.9: *DES computation path.*

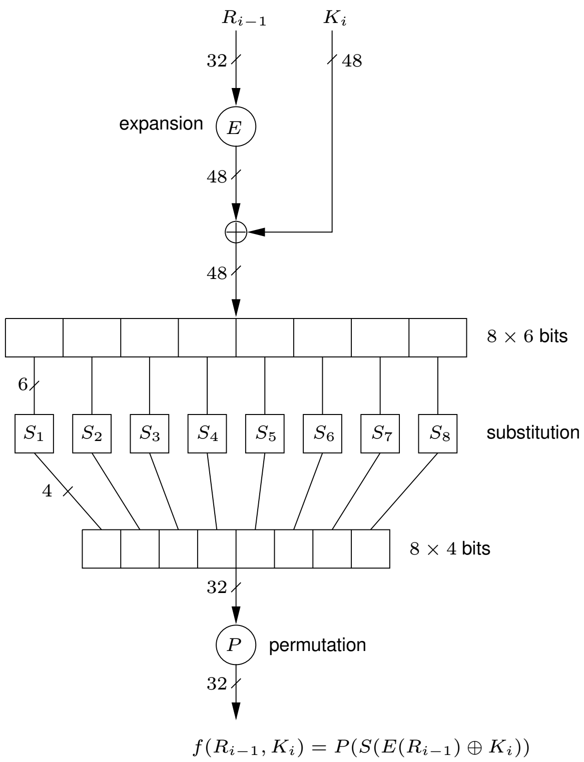

Figure 7.10: *DES inner function f.*

7.83 Algorithm DES key schedule

INPUT: 64-bit key $K = k_1 \dots k_{64}$ (including 8 odd-parity bits).
OUTPUT: sixteen 48-bit keys K_i, $1 \le i \le 16$.

1. Define v_i, $1 \le i \le 16$ as follows: $v_i = 1$ for $i \in \{1, 2, 9, 16\}$; $v_i = 2$ otherwise. (These are left-shift values for 28-bit circular rotations below.)
2. $T \leftarrow$ PC1(K); represent T as 28-bit halves (C_0, D_0). (Use PC1 in Table 7.4 to select bits from K: $C_0 = k_{57}k_{49} \dots k_{36}$, $D_0 = k_{63}k_{55} \dots k_4$.)
3. For i from 1 to 16, compute K_i as follows: $C_i \leftarrow (C_{i-1} \hookleftarrow v_i)$, $D_i \leftarrow (D_{i-1} \hookleftarrow v_i)$, $K_i \leftarrow$ PC2(C_i, D_i). (Use PC2 in Table 7.4 to select 48 bits from the concatenation $b_1b_2 \dots b_{56}$ of C_i and D_i: $K_i = b_{14}b_{17} \dots b_{32}$. '$\hookleftarrow$' denotes left circular shift.)

If decryption is designed as a simple variation of the encryption function, savings result in hardware or software code size. DES achieves this as outlined in Note 7.84.

7.84 Note (*DES decryption*) DES decryption consists of the encryption algorithm with the same key but reversed key schedule, using in order $K_{16}, K_{15}, \dots, K_1$ (see Note 7.85). This works as follows (refer to Figure 7.9). The effect of IP^{-1} is cancelled by IP in decryption, leaving (R_{16}, L_{16}); consider applying round 1 to this input. The operation on the left half yields, rather than $L_0 \oplus f(R_0, K_1)$, now $R_{16} \oplus f(L_{16}, K_{16})$ which, since $L_{16} = R_{15}$ and $R_{16} = L_{15} \oplus f(R_{15}, K_{16})$, is equal to $L_{15} \oplus f(R_{15}, K_{16}) \oplus f(R_{15}, K_{16}) = L_{15}$. Thus round 1 decryption yields (R_{15}, L_{15}), i.e., inverting round 16. Note that the cancellation

PC1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
above for C_i; below for D_i						
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

PC2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Table 7.4: *DES key schedule bit selections (PC1 and PC2).*

of each round is independent of the definition of f and the specific value of K_i; the swapping of halves combined with the XOR process is inverted by the second application. The remaining 15 rounds are likewise cancelled one by one in reverse order of application, due to the reversed key schedule.

7.85 Note (*DES decryption key schedule*) Subkeys $K_1, \dots, K_{16}$ may be generated by Algorithm 7.83 and used in reverse order, or generated in reverse order directly as follows. Note that after K_{16} is generated, the original values of the 28-bit registers C and D are restored (each has rotated 28 bits). Consequently, and due to the choice of shift-values, modifying Algorithm 7.83 as follows generates subkeys in order $K_{16}, \dots, K_1$: replace the left-shifts by right-shift rotates; change the shift value v_1 to 0.

7.86 Example (*DES test vectors*) The plaintext "Now is the time for all ", represented as a string of 8-bit hex characters (7-bit ASCII characters plus leading 0-bit), and encrypted using the DES key specified by the hex string K = `0123456789ABCDEF` results in the following plaintext/ciphertext:
P = `4E6F772069732074 68652074696D6520 666F7220616C6C20`
C = `3FA40E8A984D4815 6A271787AB8883F9 893D51EC4B563B53`. □

7.4.3 DES properties and strength

There are many desirable characteristics for block ciphers. These include: each bit of the ciphertext should depend on all bits of the key and all bits of the plaintext; there should be no statistical relationship evident between plaintext and ciphertext; altering any single plaintext or key bit should alter each ciphertext bit with probability $\frac{1}{2}$; and altering a ciphertext bit should result in an unpredictable change to the recovered plaintext block. Empirically, DES satisfies these basic objectives. Some known properties and anomalies of DES are given below.

(i) Complementation property

7.87 Fact Let E denote DES, and $\overline{x}$ the bitwise complement of x. Then $y = E_K(x)$ implies $\overline{y} = E_{\overline{K}}(\overline{x})$. That is, bitwise complementing both the key K and the plaintext x results in complemented DES ciphertext.

Justification: Compare the first round output (see Figure 7.10) to (L_0, R_0) for the uncomplemented case. The combined effect of the plaintext and key being complemented results in the inputs to the XOR preceding the S-boxes (the expanded R_{i-1} and subkey K_i) both being complemented; this double complementation cancels out in the XOR operation, resulting in S-box inputs, and thus an overall result $f(R_0, K_1)$, as before. This quantity is then XORed (Figure 7.9) to $\overline{L_0}$ (previously L_0), resulting in $\overline{L_1}$ (rather than L_1). The same effect follows in the remaining rounds.

The complementation property is normally of no help to a cryptanalyst in known-plaintext exhaustive key search. If an adversary has, for a fixed unknown key K, a chosen-plaintext set of (x, y) data (P_1, C_1), $(\overline{P_1}, C_2)$, then $C_2 = E_K(\overline{P_1})$ implies $\overline{C_2} = E_{\overline{K}}(P_1)$. Checking if the key K with plaintext P_1 yields either C_1 or $\overline{C_2}$ now rules out two keys with one encryption operation, thus reducing the expected number of keys required before success from 2^{55} to 2^{54}. This is not a practical concern.

(ii) Weak keys, semi-weak keys, and fixed points

If subkeys K_1 to K_{16} are equal, then the reversed and original schedules create identical subkeys: $K_1 = K_{16}$, $K_2 = K_{15}$, and so on. Consequently, the encryption and decryption functions coincide. These are called weak keys (and also: *palindromic keys*).

7.88 Definition A DES *weak key* is a key K such that $E_K(E_K(x)) = x$ for all x, i.e., defining an involution. A pair of DES *semi-weak* keys is a pair (K_1, K_2) with $E_{K_1}(E_{K_2}(x)) = x$.

Encryption with one key of a semi-weak pair operates as does decryption with the other.

7.89 Fact DES has four weak keys and six pairs of semi-weak keys.

The four DES weak keys are listed in Table 7.5, along with corresponding 28-bit variables C_0 and D_0 of Algorithm 7.83; here $\{0\}^j$ represents j repetitions of bit 0. Since C_0 and D_0 are all-zero or all-one bit vectors, and rotation of these has no effect, it follows that all subkeys K_i are equal and an involution results as noted above.

The six pairs of DES semi-weak keys are listed in Table 7.6. Note their defining property (Definition 7.88) occurs when subkeys K_1 through K_{16} of the first key, respectively, equal subkeys K_{16} through K_1 of the second. This requires that a 1-bit circular left-shift of each of C_0 and D_0 for the first 56-bit key results in the (C_0, D_0) pair for the second 56-bit key (see Note 7.84), and thereafter left-rotating C_i and D_i one or two bits for the first results in the same value as right-rotating those for the second the same number of positions. The values in Table 7.6 satisfy these conditions. Given any one 64-bit semi-weak key, its paired semi-weak key may be obtained by splitting it into two halves and rotating each half through 8 bits.

7.90 Fact Let E denote DES. For each of the four DES weak keys K, there exist 2^{32} *fixed points* of E_K, i.e., plaintexts x such that $E_K(x) = x$. Similarly, four of the twelve semi-weak keys K each have 2^{32} *anti-fixed points*, i.e., x such that $E_K(x) = \overline{x}$.

The four semi-weak keys of Fact 7.90 are in the upper portion of Table 7.6. These are called *anti-palindromic keys*, since for these $K_1 = \overline{K_{16}}$, $K_2 = \overline{K_{15}}$, and so on.

(iii) DES is not a group

For a fixed DES key K, DES defines a permutation from $\{0,1\}^{64}$ to $\{0,1\}^{64}$. The set of DES keys defines 2^{56} such (potentially different) permutations. If this set of permutations was closed under composition (i.e., given any two keys K_1, K_2, there exists a third key K_3 such that $E_{K_3}(x) = E_{K_2}(E_{K_1}(x))$ for all x) then multiple encryption would be equivalent to single encryption. Fact 7.91 states that this is not the case for DES.

weak key (hexadecimal)	C_0	D_0
`0101 0101 0101 0101`	$\{0\}^{28}$	$\{0\}^{28}$
`FEFE FEFE FEFE FEFE`	$\{1\}^{28}$	$\{1\}^{28}$
`1F1F 1F1F 0E0E 0E0E`	$\{0\}^{28}$	$\{1\}^{28}$
`E0E0 E0E0 F1F1 F1F1`	$\{1\}^{28}$	$\{0\}^{28}$

Table 7.5: *Four DES weak keys.*

C_0	D_0	semi-weak key pair (hexadecimal)	C_0	D_0
$\{01\}^{14}$	$\{01\}^{14}$	`01FE 01FE 01FE 01FE, FE01 FE01 FE01 FE01`	$\{10\}^{14}$	$\{10\}^{14}$
$\{01\}^{14}$	$\{10\}^{14}$	`1FE0 1FE0 0EF1 0EF1, E01F E01F F10E F10E`	$\{10\}^{14}$	$\{01\}^{14}$
$\{01\}^{14}$	$\{0\}^{28}$	`01E0 01E0 01F1 01F1, E001 E001 F101 F101`	$\{10\}^{14}$	$\{0\}^{28}$
$\{01\}^{14}$	$\{1\}^{28}$	`1FFE 1FFE 0EFE 0EFE, FE1F FE1F FE0E FE0E`	$\{10\}^{14}$	$\{1\}^{28}$
$\{0\}^{28}$	$\{01\}^{14}$	`011F 011F 010E 010E, 1F01 1F01 0E01 0E01`	$\{0\}^{28}$	$\{10\}^{14}$
$\{1\}^{28}$	$\{01\}^{14}$	`E0FE E0FE F1FE F1FE, FEE0 FEE0 FEF1 FEF1`	$\{1\}^{28}$	$\{10\}^{14}$

Table 7.6: *Six pairs of DES semi-weak keys (one pair per line).*

7.91 Fact The set of 2^{56} permutations defined by the 2^{56} DES keys is not closed under functional composition. Moreover, a lower bound on the size of the group generated by composing this set of permutations is 10^{2499}.

The lower bound in Fact 7.91 is important with respect to using DES for multiple encryption. If the group generated by functional composition was too small, then multiple encryption would be less secure than otherwise believed.

(iv) Linear and differential cryptanalysis of DES

Assuming that obtaining enormous numbers of known-plaintext pairs is feasible, linear cryptanalysis provides the most powerful attack on DES to date; it is not, however, considered a threat to DES in practical environments. Linear cryptanalysis is also possible in a ciphertext-only environment if some underlying plaintext redundancy is known (e.g., parity bits or high-order 0-bits in ASCII characters).

Differential cryptanalysis is one of the most general cryptanalytic tools to date against modern iterated block ciphers, including DES, Lucifer, and FEAL among many others. It is, however, primarily a chosen-plaintext attack. Further information on linear and differential cryptanalysis is given in §7.8.

7.92 Note (*strength of DES*) The complexity (see §7.2.1) of the best attacks currently known against DES is given in Table 7.7; percentages indicate success rate for specified attack parameters. The 'processing complexity' column provides only an estimate of the expected cost (operation costs differ across the various attacks); for exhaustive search, the cost is in DES operations. Regarding storage complexity, both linear and differential cryptanalysis require only negligible storage in the sense that known or chosen texts can be processed individually and discarded, but in a practical attack, storage for accumulated texts would be required if ciphertext was acquired prior to commencing the attack.

attack method	data complexity		storage complexity	processing complexity
	known	chosen		
exhaustive precomputation	—	1	2^{56}	1 (table lookup)
exhaustive search	1	—	negligible	2^{55}
linear cryptanalysis	2^{43} (85%)	—	for texts	2^{43}
	2^{38} (10%)	—	for texts	2^{50}
differential cryptanalysis	—	2^{47}	for texts	2^{47}
	2^{55}	—	for texts	2^{55}

Table 7.7: *DES strength against various attacks.*

7.93 Remark (*practicality of attack models*) To be meaningful, attack comparisons based on different models (e.g., Table 7.7) must appropriately weigh the feasibility of extracting (acquiring) enormous amounts of chosen (known) plaintexts, which is considerably more difficult to arrange than a comparable number of computing cycles on an adversary's own machine. Exhaustive search with one known plaintext-ciphertext pair (for ciphertext-only, see Example 7.28) and 2^{55} DES operations is significantly more feasible in practice (e.g., using highly parallelized custom hardware) than linear cryptanalysis (LC) requiring 2^{43} known pairs.

While exhaustive search, linear, and differential cryptanalysis allow recovery of a DES key and, therefore, the entire plaintext, the attacks of Note 7.8, which become feasible once about 2^{32} ciphertexts are available, may be more efficient if the goal is to recover only part of the text.

7.5 FEAL

The Fast Data Encipherment Algorithm (FEAL) is a family of algorithms which has played a critical role in the development and refinement of various advanced cryptanalytic techniques, including linear and differential cryptanalysis. FEAL-N maps 64-bit plaintext to 64-bit ciphertext blocks under a 64-bit secret key. It is an N-round Feistel cipher similar to DES (cf. Equations (7.4), (7.5)), but with a far simpler f-function, and augmented by initial and final stages which XOR the two data halves as well as XOR subkeys directly onto the data halves.

FEAL was designed for speed and simplicity, especially for software on 8-bit microprocessors (e.g., chipcards). It uses byte-oriented operations (8-bit addition mod 256, 2-bit left rotation, and XOR), avoids bit-permutations and table look-ups, and offers small code size. The initial commercially proposed version with 4 rounds (FEAL-4), positioned as a fast alternative to DES, was found to be considerably less secure than expected (see Table 7.10). FEAL-8 was similarly found to offer less security than planned. FEAL-16 or FEAL-32 may yet offer security comparable to DES, but throughput decreases as the number of rounds rises. Moreover, whereas the speed of DES implementations can be improved through very large lookup tables, this appears more difficult for FEAL.

Algorithm 7.94 specifies FEAL-8. The f-function $f(A, Y)$ maps an input pair of 32×16 bits to a 32-bit output. Within the f function, two byte-oriented data substitutions (S-boxes) S_0 and S_1 are each used twice; each maps a pair of 8-bit inputs to an 8-bit output

row	column number															
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
S_1																
[0]	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
[1]	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
[2]	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
[3]	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2																
[0]	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
[1]	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
[2]	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
[3]	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3																
[0]	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
[1]	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
[2]	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
[3]	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4																
[0]	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
[1]	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
[2]	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
[3]	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5																
[0]	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
[1]	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
[2]	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
[3]	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6																
[0]	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
[1]	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
[2]	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
[3]	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7																
[0]	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
[1]	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
[2]	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
[3]	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8																
[0]	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
[1]	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
[2]	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
[3]	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table 7.8: *DES S-boxes.*

(see Table 7.9). S_0 and S_1 add a single bit $d \in \{0,1\}$ to 8-bit arguments x and y, ignore the carry out of the top bit, and left rotate the result 2 bits (ROT2):

$$S_d(x,y) = ROT2(x + y + d \bmod 256) \tag{7.6}$$

The key schedule uses a function $f_K(A,B)$ similar to the f-function (see Table 7.9; A_i, B_i, Y_i, t_i, and U_i are 8-bit variables), mapping two 32-bit inputs to a 32-bit output.

	$U \leftarrow f(A,Y)$	$U \leftarrow f_K(A,B)$
$t_1 =$	$(A_0 \oplus A_1) \oplus Y_0$	$A_0 \oplus A_1$
$t_2 =$	$(A_2 \oplus A_3) \oplus Y_1$	$A_2 \oplus A_3$
$U_1 =$	$S_1(t_1, t_2)$	$S_1(t_1, t_2 \oplus B_0)$
$U_2 =$	$S_0(t_2, U_1)$	$S_0(t_2, U_1 \oplus B_1)$
$U_0 =$	$S_0(A_0, U_1)$	$S_0(A_0, U_1 \oplus B_2)$
$U_3 =$	$S_1(A_3, U_2)$	$S_1(A_3, U_2 \oplus B_3)$

Table 7.9: *Output $U = (U_0, U_1, U_2, U_3)$ for FEAL functions f, f_K (Algorithm 7.94).*

As the operations of 2-bit rotation and XOR are both linear, the only nonlinear elementary operation in FEAL is addition mod 256.

7.94 Algorithm Fast Data Encipherment Algorithm (FEAL-8)

INPUT: 64-bit plaintext $M = m_1 \dots m_{64}$; 64-bit key $K = k_1 \dots k_{64}$.
OUTPUT: 64-bit ciphertext block $C = c_1 \dots c_{64}$. (For decryption, see Note 7.96.)

1. (key schedule) Compute sixteen 16-bit subkeys K_i from K using Algorithm 7.95.
2. Define $M_L = m_1 \cdots m_{32}$, $M_R = m_{33} \cdots m_{64}$.
3. $(L_0, R_0) \leftarrow (M_L, M_R) \oplus ((K_8, K_9),\ (K_{10}, K_{11}))$. (XOR initial subkeys.)
4. $R_0 \leftarrow R_0 \oplus L_0$.
5. For i from 1 to 8 do: $L_i \leftarrow R_{i-1},\ R_i \leftarrow L_{i-1} \oplus f(R_{i-1}, K_{i-1})$. (Use Table 7.9 for $f(A,Y)$ with $A = R_{i-1} = (A_0, A_1, A_2, A_3)$ and $Y = K_{i-1} = (Y_0, Y_1)$.)
6. $L_8 \leftarrow L_8 \oplus R_8$.
7. $(R_8, L_8) \leftarrow (R_8, L_8) \oplus ((K_{12}, K_{13}),\ (K_{14}, K_{15}))$. (XOR final subkeys.)
8. $C \leftarrow (R_8, L_8)$. (Note the order of the final blocks is exchanged.)

7.95 Algorithm FEAL-8 key schedule

INPUT: 64-bit key $K = k_1 \dots k_{64}$.
OUTPUT: 256-bit extended key (16-bit subkeys K_i, $0 \le i \le 15$).

1. (initialize) $U^{(-2)} \leftarrow 0,\ U^{(-1)} \leftarrow k_1 \dots k_{32},\ U^{(0)} \leftarrow k_{33} \dots k_{64}$.
2. $U \stackrel{\text{def}}{=} (U_0, U_1, U_2, U_3)$ for 8-bit U_i. Compute $K_0, \dots, K_{15}$ as i runs from 1 to 8:
   (a) $U \leftarrow f_K(U^{(i-2)}, U^{(i-1)} \oplus U^{(i-3)})$. ($f_K$ is defined in Table 7.9, where A and B denote 4-byte vectors (A_0, A_1, A_2, A_3), (B_0, B_1, B_2, B_3).)
   (b) $K_{2i-2} = (U_0, U_1),\quad K_{2i-1} = (U_2, U_3),\quad U^{(i)} \leftarrow U$.

7.96 Note (*FEAL decryption*) Decryption may be achieved using Algorithm 7.94 with the same key K and ciphertext $C = (R_8, L_8)$ as the plaintext input M, but with the key schedule reversed. More specifically, subkeys $((K_{12}, K_{13}), (K_{14}, K_{15}))$ are used for the initial XOR (step 3), $((K_8, K_9), (K_{10}, K_{11}))$ for the final XOR (step 7), and the round keys are used from K_7 back to K_0 (step 5). This is directly analogous to decryption for DES (Note 7.84).

7.97 Note (*FEAL-N*) FEAL with 64-bit key can be generalized to N-rounds, N even. $N = 2^x$ is recommended; $x = 3$ yields FEAL-8 (Algorithm 7.94). FEAL-N uses $N + 8$ sixteen-bit subkeys: $K_0, \ldots, K_{N-1}$, respectively, in round i; $K_N, \ldots, K_{N+3}$ for the initial XOR; and $K_{N+4}, \ldots K_{N+7}$ for the final XOR. The key schedule of Algorithm 7.95 is directly generalized to compute keys K_0 through K_{N+7} as i runs from 1 to $(N/2) + 4$.

7.98 Note (*FEAL-NX*) Extending FEAL-N to use a 128-bit key results in FEAL-NX, with altered key schedule as follows. The key is split into 64-bit halves (K_L, K_R). K_R is partitioned into 32-bit halves (K_{R1}, K_{R2}). For $1 \le i \le (N/2) + 4$, define $Q_i = K_{R1} \oplus K_{R2}$ for $i \equiv 1 \bmod 3$; $Q_i = K_{R1}$ for $i \equiv 2 \bmod 3$; and $Q_i = K_{R2}$ for $i \equiv 0 \bmod 3$. The second argument $(U^{(i-1)} \oplus U^{(i-3)})$ to f_K in step 2a of Algorithm 7.95 is replaced by $U^{(i-1)} \oplus U^{(i-3)} \oplus Q_i$. For $K_R = 0$, FEAL-NX matches FEAL-N with K_L as the 64-bit FEAL-N key K.

7.99 Example (*FEAL test vectors*) For hex plaintext $M =$ `00000000 00000000` and hex key $K =$ `01234567 89ABCDEF`, Algorithm 7.95 generates subkeys $(K_0, \ldots, K_7) =$ `DF3BCA36 F17C1AEC 45A5B9C7 26EBAD25`, $(K_8, \ldots, K_{15}) =$ `8B2AECB7 AC509D4C 22CD479B A8D50CB5`. Algorithm 7.94 generates FEAL-8 ciphertext $C =$ `CEEF2C86 F2490752`. For FEAL-16, the corresponding ciphertext is $C' =$ `3ADE0D2A D84D0B6F`; for FEAL-32, $C'' =$ `69B0FAE6 DDED6B0B`. For 128-bit key (K_L, K_R) with $K_L = K_R = K$ as above, M has corresponding FEAL-8X ciphertext $C''' =$ `92BEB65D 0E9382FB`. □

7.100 Note (*strength of FEAL*) Table 7.10 gives various published attacks on FEAL; LC and DC denote linear and differential cryptanalysis, and times are on common personal computers or workstations.

attack method	data complexity known	data complexity chosen	storage complexity	processing complexity
FEAL-4 – LC	5	—	30K bytes	6 minutes
FEAL-6 – LC	100	—	100K bytes	40 minutes
FEAL-8 – LC	2^{24}			10 minutes
FEAL-8 – DC		2^7 pairs	280K bytes	2 minutes
FEAL-16 – DC	—	2^{29} pairs		2^{30} operations
FEAL-24 – DC	—	2^{45} pairs		2^{46} operations
FEAL-32 – DC	—	2^{66} pairs		2^{67} operations

Table 7.10: *FEAL strength against various attacks.*

7.6 IDEA

The cipher named IDEA (International Data Encryption Algorithm) encrypts 64-bit plaintext to 64-bit ciphertext blocks, using a 128-bit input key K. Based in part on a novel generalization of the Feistel structure, it consists of 8 computationally identical rounds followed by an output transformation (see Figure 7.11). Round r uses six 16-bit subkeys $K_i^{(r)}$, $1 \le i \le 6$, to transform a 64-bit input X into an output of four 16-bit blocks, which are input to the next round. The round 8 output enters the output transformation, employing four additional subkeys $K_i^{(9)}$, $1 \le i \le 4$ to produce the final ciphertext $Y = (Y_1, Y_2, Y_3, Y_4)$. All subkeys are derived from K.

A dominant design concept in IDEA is mixing operations from three different algebraic groups of 2^n elements. The corresponding group operations on sub-blocks a and b of bitlength $n = 16$ are bitwise XOR: $a \oplus b$; addition mod 2^n: $(a+b)$ AND `0xFFFF`, denoted $a \boxplus b$; and (modified) multiplication mod 2^n+1, with $0 \in \mathbb{Z}_{2^n}$ associated with $2^n \in \mathbb{Z}_{2^n+1}$: $a \odot b$ (see Note 7.104).

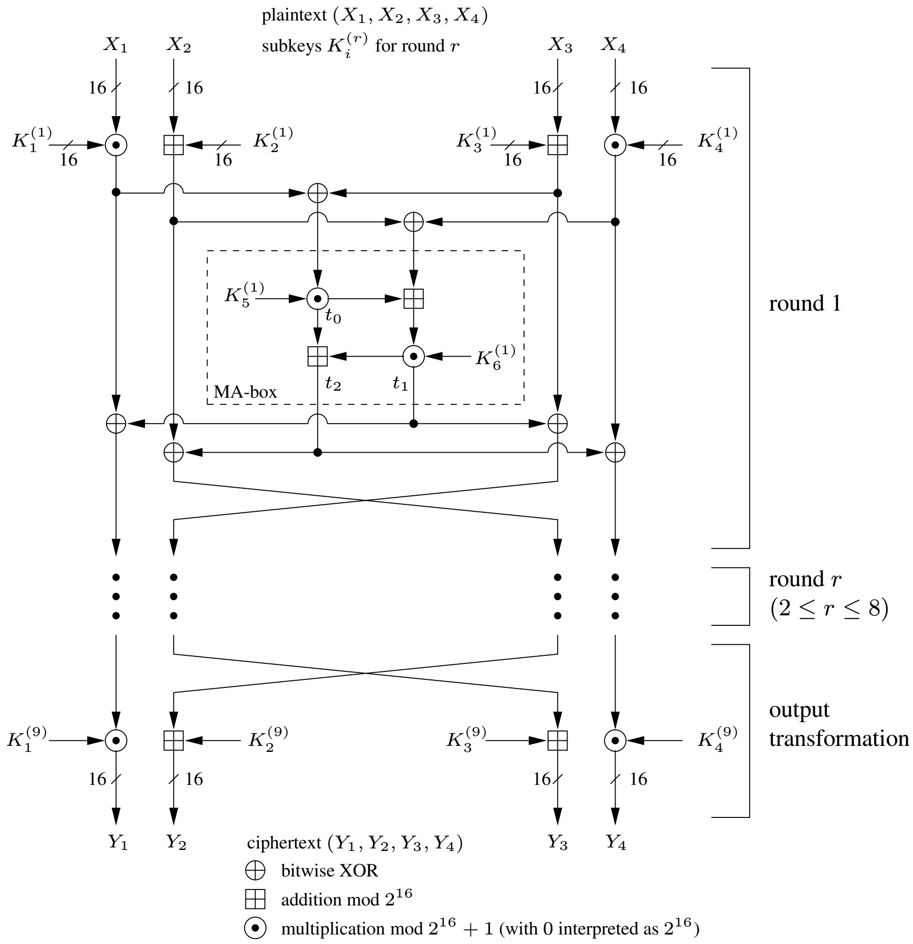

Figure 7.11: *IDEA computation path.*

7.101 Algorithm IDEA encryption

INPUT: 64-bit plaintext $M = m_1 \dots m_{64}$; 128-bit key $K = k_1 \dots k_{128}$.
OUTPUT: 64-bit ciphertext block $Y = (Y_1, Y_2, Y_3, Y_4)$. (For decryption, see Note 7.103.)

1. (key schedule) Compute 16-bit subkeys $K_1^{(r)}, \dots, K_6^{(r)}$ for rounds $1 \le r \le 8$, and $K_1^{(9)}, \dots, K_4^{(9)}$ for the output transformation, using Algorithm 7.102.
2. $(X_1, X_2, X_3, X_4) \leftarrow (m_1 \dots m_{16}, m_{17} \dots m_{32}, m_{33} \dots m_{48}, m_{49} \dots m_{64})$, where X_i is a 16-bit data store.
3. For round r from 1 to 8 do:
   (a) $X_1 \leftarrow X_1 \odot K_1^{(r)}$, $X_4 \leftarrow X_4 \odot K_4^{(r)}$, $X_2 \leftarrow X_2 \boxplus K_2^{(r)}$, $X_3 \leftarrow X_3 \boxplus K_3^{(r)}$.
   (b) $t_0 \leftarrow K_5^{(r)} \odot (X_1 \oplus X_3)$, $t_1 \leftarrow K_6^{(r)} \odot (t_0 \boxplus (X_2 \oplus X_4))$, $t_2 \leftarrow t_0 \boxplus t_1$.
   (c) $X_1 \leftarrow X_1 \oplus t_1$, $X_4 \leftarrow X_4 \oplus t_2$, $a \leftarrow X_2 \oplus t_2$, $X_2 \leftarrow X_3 \oplus t_1$, $X_3 \leftarrow a$.
4. (output transformation) $Y_1 \leftarrow X_1 \odot K_1^{(9)}$, $Y_4 \leftarrow X_4 \odot K_4^{(9)}$, $Y_2 \leftarrow X_3 \boxplus K_2^{(9)}$, $Y_3 \leftarrow X_2 \boxplus K_3^{(9)}$.

7.102 Algorithm IDEA key schedule (encryption)

INPUT: 128-bit key $K = k_1 \dots k_{128}$.
OUTPUT: 52 16-bit key sub-blocks $K_i^{(r)}$ for 8 rounds r and the output transformation.

1. Order the subkeys $K_1^{(1)} \dots K_6^{(1)}, K_1^{(2)} \dots K_6^{(2)}, \dots, K_1^{(8)} \dots K_6^{(8)}, K_1^{(9)} \dots K_4^{(9)}$.
2. Partition K into eight 16-bit blocks; assign these directly to the first 8 subkeys.
3. Do the following until all 52 subkeys are assigned: cyclic shift K left 25 bits; partition the result into 8 blocks; assign these blocks to the next 8 subkeys.

The key schedule of Algorithm 7.102 may be converted into a table which lists, for each of the 52 keys blocks, which 16 (consecutive) bits of the input key K form it.

7.103 Note (*IDEA decryption*) Decryption is achieved using Algorithm 7.101 with the ciphertext Y provided as input M, and the same encryption key K, but the following change to the key schedule. First use K to derive all encryption subkeys $K_i^{(r)}$; from these compute the decryption subkeys $K'^{(r)}_i$ per Table 7.11; then use $K'^{(r)}_i$ in place of $K_i^{(r)}$ in Algorithm 7.101. In Table 7.11, $-K_i$ denotes the additive inverse (mod 2^{16}) of K_i: the integer $u = (2^{16} - K_i)$ AND `0xFFFF`, $0 \le u \le 2^{16} - 1$. K_i^{-1} denotes the multiplicative inverse (mod $2^{16} + 1$) of K_i, also in $\{0, 1, \dots, 2^{16} - 1\}$, derivable by the Extended Euclidean algorithm (Algorithm 2.107), which on inputs $a \ge b \ge 0$ returns integers x and y such that $ax + by = \gcd(a, b)$. Using $a = 2^{16} + 1$ and $b = K_i$, the gcd is always 1 (except for $K_i = 0$, addressed separately) and thus $K_i^{-1} = y$, or $2^{16} + 1 + y$ if $y < 0$. When $K_i = 0$, this input is mapped to 2^{16} (since the inverse is defined by $K_i \odot K_i^{-1} = 1$; see Note 7.104) and $(2^{16})^{-1} = 2^{16}$ is then defined to give $K_i^{-1} = 0$.

7.104 Note (*definition of* $\odot$) In IDEA, $a \odot b$ corresponds to a (modified) multiplication, modulo $2^{16}+1$, of unsigned 16-bit integers a and b, where $0 \in \mathbb{Z}_{2^{16}}$ is associated with $2^{16} \in \mathbb{Z}^*_{2^{16}+1}$ as follows:[2] if $a = 0$ or $b = 0$, replace it by 2^{16} (which is $\equiv -1 \bmod 2^{16} + 1$) prior to modular multiplication; and if the result is 2^{16}, replace this by 0. Thus, $\odot$ maps two 16-bit inputs to a 16-bit output. Pseudo-code for $\odot$ is as follows (cf. Note 7.105, for ordinary

[2]Thus the operands of $\odot$ are from a set of cardinality 2^{16} ($\mathbb{Z}^*_{2^{16}+1}$) as are those of $\oplus$ and $\boxplus$.

round r	$K'^{(r)}_1$	$K'^{(r)}_2$	$K'^{(r)}_3$	$K'^{(r)}_4$	$K'^{(r)}_5$	$K'^{(r)}_6$
$r=1$	$(K_1^{(10-r)})^{-1}$	$-K_2^{(10-r)}$	$-K_3^{(10-r)}$	$(K_4^{(10-r)})^{-1}$	$K_5^{(9-r)}$	$K_6^{(9-r)}$
$2 \le r \le 8$	$(K_1^{(10-r)})^{-1}$	$-K_3^{(10-r)}$	$-K_2^{(10-r)}$	$(K_4^{(10-r)})^{-1}$	$K_5^{(9-r)}$	$K_6^{(9-r)}$
$r=9$	$(K_1^{(10-r)})^{-1}$	$-K_2^{(10-r)}$	$-K_3^{(10-r)}$	$(K_4^{(10-r)})^{-1}$	—	—

Table 7.11: *IDEA decryption subkeys $K'^{(r)}_i$ derived from encryption subkeys $K_i^{(r)}$.*

multiplication mod $2^{16}+1$), for c a 32-bit unsigned integer: if $(a=0)$ $r \leftarrow$ (`0x10001` $-\, b$) (since $2^{16}b \equiv -b$), elseif $(b=0)$ $r \leftarrow$ (`0x10001` $-\, a$) (by similar reasoning), else $\{c \leftarrow ab;\ r \leftarrow ((c$ `AND 0xFFFF`$) - (c >> 16))$; if $(r<0)$ $r \leftarrow$ (`0x10001` $+\, r)\}$, with return value (r `AND 0xFFFF`) in all 3 cases.

7.105 Note (*implementing* $ab \bmod 2^n+1$) Multiplication mod $2^{16}+1$ may be efficiently implemented as follows, for $0 \le a, b \le 2^{16}$ (cf. §14.3.4). Let $c = ab = c_0 \cdot 2^{32} + c_H \cdot 2^{16} + c_L$, where $c_0 \in \{0,1\}$ and $0 \le c_L, c_H < 2^{16}$. To compute $c' = c \bmod (2^{16}+1)$, first obtain c_L and c_H by standard multiplication. For $a = b = 2^{16}$, note that $c_0 = 1$, $c_L = c_H = 0$, and $c' = (-1)(-1) = 1$, since $2^{16} \equiv -1 \bmod (2^{16}+1)$; otherwise, $c_0 = 0$. Consequently, $c' = c_L - c_H + c_0$ if $c_L \ge c_H$, while $c' = c_L - c_H + (2^{16}+1)$ if $c_L < c_H$ (since then $-2^{16} < c_L - c_H < 0$).

7.106 Example (*IDEA test vectors*) Sample data for IDEA encryption of 64-bit plaintext M using 128-bit key K is given in Table 7.12. All entries are 16-bit values displayed in hexadecimal. Table 7.13 details the corresponding decryption of the resulting 64-bit ciphertext C under the same key K. □

128-bit key $K=(1,2,3,4,5,6,7,8)$							64-bit plaintext $M=(0,1,2,3)$			
r	$K_1^{(r)}$	$K_2^{(r)}$	$K_3^{(r)}$	$K_4^{(r)}$	$K_5^{(r)}$	$K_6^{(r)}$	X_1	X_2	X_3	X_4
1	0001	0002	0003	0004	0005	0006	00f0	00f5	010a	0105
2	0007	0008	0400	0600	0800	0a00	222f	21b5	f45e	e959
3	0c00	0e00	1000	0200	0010	0014	0f86	39be	8ee8	1173
4	0018	001c	0020	0004	0008	000c	57df	ac58	c65b	ba4d
5	2800	3000	3800	4000	0800	1000	8e81	ba9c	f77f	3a4a
6	1800	2000	0070	0080	0010	0020	6942	9409	e21b	1c64
7	0030	0040	0050	0060	0000	2000	99d0	c7f6	5331	620e
8	4000	6000	8000	a000	c000	e001	0a24	0098	ec6b	4925
9	0080	00c0	0100	0140	—	—	11fb	ed2b	0198	6de5

Table 7.12: *IDEA encryption sample: round subkeys and ciphertext (X_1, X_2, X_3, X_4).*

7.107 Note (*security of IDEA*) For the full 8-round IDEA, other than attacks on weak keys (see page 279), no published attack is better than exhaustive search on the 128-bit key space. The security of IDEA currently appears bounded only by the weaknesses arising from the relatively small (compared to its keylength) blocklength of 64 bits.

	$K = (1,2,3,4,5,6,7,8)$						$C =$ (11fb,ed2b,0198,6de5)			
r	$K'^{(r)}_1$	$K'^{(r)}_2$	$K'^{(r)}_3$	$K'^{(r)}_4$	$K'^{(r)}_5$	$K'^{(r)}_6$	X_1	X_2	X_3	X_4
1	fe01	ff40	ff00	659a	c000	e001	d98d	d331	27f6	82b8
2	fffd	8000	a000	cccc	0000	2000	bc4d	e26b	9449	a576
3	a556	ffb0	ffc0	52ab	0010	0020	0aa4	f7ef	da9c	24e3
4	554b	ff90	e000	fe01	0800	1000	ca46	fe5b	dc58	116d
5	332d	c800	d000	fffd	0008	000c	748f	8f08	39da	45cc
6	4aab	ffe0	ffe4	c001	0010	0014	3266	045e	2fb5	b02e
7	aa96	f000	f200	ff81	0800	0a00	0690	050a	00fd	1dfa
8	4925	fc00	fff8	552b	0005	0006	0000	0005	0003	000c
9	0001	fffe	fffd	c001	—	—	0000	0001	0002	0003

Table 7.13: *IDEA decryption sample: round subkeys and variables* (X_1, X_2, X_3, X_4).

7.7 SAFER, RC5, and other block ciphers

7.7.1 SAFER

SAFER K-64 (Secure And Fast Encryption Routine, with 64-bit key) is an iterated block cipher with 64-bit plaintext and ciphertext blocks. It consists of r identical rounds followed by an output transformation. The original recommendation of 6 rounds was followed by a recommendation to adopt a slightly modified key schedule (yielding SAFER SK-64, which should be used rather than SAFER K-64 – see Note 7.110) and to use 8 rounds (maximum $r = 10$). Both key schedules expand the 64-bit external key into $2r+1$ subkeys each of 64-bits (two for each round plus one for the output transformation). SAFER consists entirely of simple byte operations, aside from byte-rotations in the key schedule; it is thus suitable for processors with small word size such as chipcards (cf. FEAL).

Details of SAFER K-64 are given in Algorithm 7.108 and Figure 7.12 (see also page 280 regarding SAFER K-128 and SAFER SK-128). The XOR-addition stage beginning each round (identical to the output transformation) XORs bytes 1, 4, 5, and 8 of the (first) round subkey with the respective round input bytes, and respectively adds (mod 256) the remaining 4 subkey bytes to the others. The XOR and addition (mod 256) operations are interchanged in the subsequent addition-XOR stage. The S-boxes are an invertible byte-to-byte substitution using one fixed 8-bit bijection (see Note 7.111). A linear transformation f (the *Pseudo-Hadamard Transform*) used in the 3-level linear layer was specially constructed for rapid diffusion. The introduction of additive key biases in the key schedule eliminates weak keys (cf. DES, IDEA). In contrast to Feistel-like and many other ciphers, in SAFER the operations used for encryption differ from those for decryption (see Note 7.113). SAFER may be viewed as an SP network (Definition 7.79).

Algorithm 7.108 uses the following definitions (L, R denote left, right 8-bit inputs):

1. $f(L,R) = (2L + R,\ L + R)$. Addition here is mod 256 (also denoted by $\boxplus$);
2. tables S and S_{inv}, and the constant table for *key biases* $B_i[j]$ as per Note 7.111.

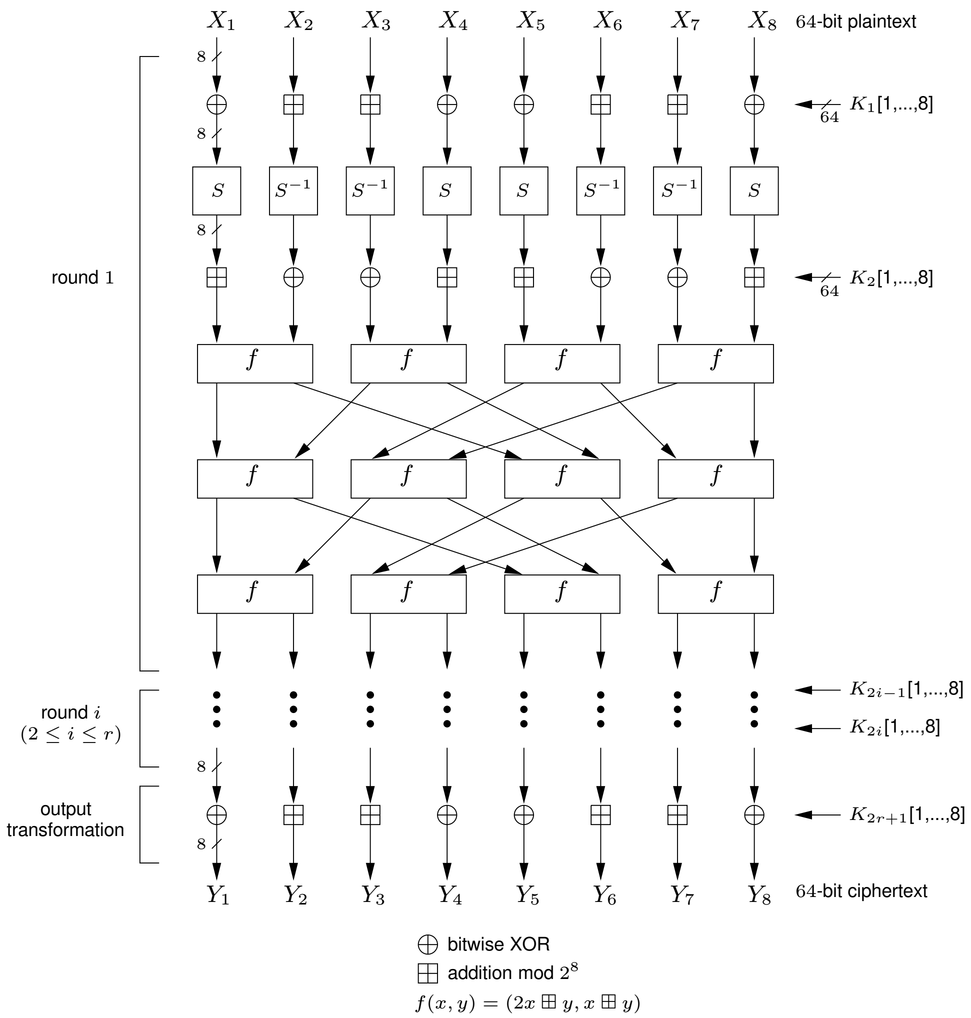

Figure 7.12: *SAFER K-64 computation path (r rounds).*

7.108 Algorithm SAFER K-64 encryption (r rounds)

INPUT: r, $6 \le r \le 10$; 64-bit plaintext $M = m_1 \cdots m_{64}$ and key $K = k_1 \cdots k_{64}$.
OUTPUT: 64-bit ciphertext block $Y = (Y_1, \ldots, Y_8)$. (For decryption, see Note 7.113.)

1. Compute 64-bit subkeys $K_1, \ldots, K_{2r+1}$ by Algorithm 7.109 with inputs K and r.
2. $(X_1, X_2, \ldots, X_8) \leftarrow (m_1 \cdots m_8,\ m_9 \cdots m_{16},\ \ldots,\ m_{57} \cdots m_{64})$.
3. For i from 1 to r do: (XOR-addition, S-box, addition-XOR, and 3 linear layers)
   (a) For $j = 1, 4, 5, 8$: $X_j \leftarrow X_j \oplus K_{2i-1}[j]$.
   For $j = 2, 3, 6, 7$: $X_j \leftarrow X_j \boxplus K_{2i-1}[j]$.
   (b) For $j = 1, 4, 5, 8$: $X_j \leftarrow S[X_j]$. For $j = 2, 3, 6, 7$: $X_j \leftarrow S_{\mathrm{inv}}[X_j]$.
   (c) For $j = 1, 4, 5, 8$: $X_j \leftarrow X_j \boxplus K_{2i}[j]$. For $j = 2, 3, 6, 7$: $X_j \leftarrow X_j \oplus K_{2i}[j]$.
   (d) For $j = 1, 3, 5, 7$: $(X_j, X_{j+1}) \leftarrow f(X_j, X_{j+1})$.
   (e) $(Y_1, Y_2) \leftarrow f(X_1, X_3)$, $(Y_3, Y_4) \leftarrow f(X_5, X_7)$,
   $(Y_5, Y_6) \leftarrow f(X_2, X_4)$, $(Y_7, Y_8) \leftarrow f(X_6, X_8)$.
   For j from 1 to 8 do: $X_j \leftarrow Y_j$.
   (f) $(Y_1, Y_2) \leftarrow f(X_1, X_3)$, $(Y_3, Y_4) \leftarrow f(X_5, X_7)$,
   $(Y_5, Y_6) \leftarrow f(X_2, X_4)$, $(Y_7, Y_8) \leftarrow f(X_6, X_8)$.
   For j from 1 to 8 do: $X_j \leftarrow Y_j$. (This mimics the previous step.)
4. (output transformation):
   For $j = 1, 4, 5, 8$: $Y_j \leftarrow X_j \oplus K_{2r+1}[j]$. For $j = 2, 3, 6, 7$: $Y_j \leftarrow X_j \boxplus K_{2r+1}[j]$.

7.109 Algorithm SAFER K-64 key schedule

INPUT: 64-bit key $K = k_1 \cdots k_{64}$; number of rounds r.
OUTPUT: 64-bit subkeys $K_1, \ldots, K_{2r+1}$. $K_i[j]$ is byte j of K_i (numbered left to right).

1. Let $R[i]$ denote an 8-bit data store and let $B_i[j]$ denote byte j of B_i (Note 7.111).
2. $(R[1], R[2], \ldots, R[8]) \leftarrow (k_1 \cdots k_8,\ k_9 \cdots k_{16},\ \ldots,\ k_{57} \cdots k_{64})$.
3. $(K_1[1], K_1[2], \ldots, K_1[8]) \leftarrow (R[1], R[2], \ldots, R[8])$.
4. For i from 2 to $2r + 1$ do: (rotate key bytes left 3 bits, then add in the bias)
   (a) For j from 1 to 8 do: $R[j] \leftarrow (R[j] \hookleftarrow 3)$.
   (b) For j from 1 to 8 do: $K_i[j] \leftarrow R[j] \boxplus B_i[j]$. (See Note 7.110.)

7.110 Note (*SAFER SK-64 – strengthened key schedule*) An improved key schedule for Algorithm 7.108, resulting in SAFER SK-64, involves three changes as follows. (i) After initializing the $R[i]$ in step 1 of Algorithm 7.109, set $R[9] \leftarrow R[1] \oplus R[2] \oplus \cdots \oplus R[8]$. (ii) Change the upper bound on the loop index in step 4a from 8 to 9. (iii) Replace the iterated line in step 4b by: $K_i[j] \leftarrow R[((i + j - 2) \bmod 9) + 1] \boxplus B_i[j]$. Thus, key bytes $1, \ldots, 8$ of $R[\cdot]$ are used for K_1; bytes $2, \ldots, 9$ for K_2; bytes $3, \ldots 9, 1$ for K_3, etc. Here and originally, $\boxplus$ denotes addition mod 256. No attack against SAFER SK-64 better than exhaustive key search is known.

7.111 Note (*S-boxes and key biases in SAFER*) The S-box, inverse S-box, and key biases for Algorithm 7.108 are constant tables as follows. $g \leftarrow 45$. $S[0] \leftarrow 1$, $S_{\mathrm{inv}}[1] \leftarrow 0$. for i from 1 to 255 do: $t \leftarrow g \cdot S[i-1] \bmod 257$, $S[i] \leftarrow t$, $S_{\mathrm{inv}}[t] \leftarrow i$. Finally, $S[128] \leftarrow 0$, $S_{\mathrm{inv}}[0] \leftarrow 128$. (Since g generates $\mathbb{Z}_{257}^*$, $S[i]$ is a bijection on $\{0, 1, \ldots, 255\}$. (Note that $g^{128} \equiv 256 \pmod{257}$, and associating 256 with 0 makes S a mapping with 8-bit input and output.) The *additive key biases* are 8-bit constants used in the key schedule (Algorithm 7.109), intended to behave as random numbers, and defined $B_i[j] = S[S[9i+j]]$ for i from 2 to $2r+1$ and j from 1 to 8. For example: $B_2 = (22, 115, 59, 30, 142, 112, 189, 134)$ and $B_{13} = (143, 41, 221, 4, 128, 222, 231, 49)$.

7.112 Remark (*S-box mapping*) The S-box of Note 7.111 is based on the function $S(x) = g^x \bmod 257$ using a primitive element $g = 45 \in \mathbb{Z}_{257}$. This mapping is nonlinear with respect to both $\mathbb{Z}_{257}$ arithmetic and the vector space of 8-tuples over $\mathbb{F}_2$ under the XOR operation. The inverse S-box is based on the base-g logarithm function.

7.113 Note (*SAFER K-64 decryption*) For decryption of Algorithm 7.108, the same key K and subkeys K_i are used as for encryption. Each encryption step is undone in reverse order, from last to first. Begin with an input transformation (XOR-subtraction stage) with key K_{2r+1} to undo the output transformation, replacing modular addition with subtraction. Follow with r decryption rounds using keys K_{2r} through K_1 (two-per-round), inverting each round in turn. Each starts with a 3-stage inverse linear layer using $f_{\rm inv}(L, R) = (L - R,\ 2R - L)$, with subtraction here mod 256, in a 3-step sequence defined as follows (to invert the byte-permutations between encryption stages):
Level 1 (for $j = 1, 3, 5, 7$): $(X_j, X_{j+1}) \leftarrow f_{\rm inv}(X_j, X_{j+1})$.
Levels 2 and 3 (each): $(Y_1, Y_2) \leftarrow f_{\rm inv}(X_1, X_5)$, $(Y_3, Y_4) \leftarrow f_{\rm inv}(X_2, X_6)$, $(Y_5, Y_6) \leftarrow f_{\rm inv}(X_3, X_7)$, $(Y_7, Y_8) \leftarrow f_{\rm inv}(X_4, X_8)$; for j from 1 to 8 do: $X_j \leftarrow Y_j$.
A subtraction-XOR stage follows (replace modular addition with subtraction), then an inverse substitution stage (exchange S and S^{-1}), and an XOR-subtraction stage.

7.114 Example (*SAFER test vectors*) Using 6-round SAFER K-64 (Algorithm 7.108) on the 64-bit plaintext $M = (1, 2, 3, 4, 5, 6, 7, 8)$ with the key $K = (8, 7, 6, 5, 4, 3, 2, 1)$ results in the ciphertext $C = (200, 242, 156, 221, 135, 120, 62, 217)$, written as 8 bytes in decimal. Using 6-round SAFER SK-64 (Note 7.110) on the plaintext M above with the key $K = (1, 2, 3, 4, 5, 6, 7, 8)$ results in the ciphertext $C = (95, 206, 155, 162, 5, 132, 56, 199)$. □

7.7.2 RC5

The RC5 block cipher has a word-oriented architecture for variable word sizes $w = 16, 32$, or 64 bits. It has an extremely compact description, and is suitable for hardware or software. The number of rounds r and the key byte-length b are also variable. It is successively more completely identified as RC5–w, RC5–w/r, and RC5–w/r/b. RC5-32/12/16 is considered a common choice of parameters; $r = 12$ rounds are recommended for RC5–32, and $r = 16$ for RC5–64.

Algorithm 7.115 specifies RC5. Plaintext and ciphertext are blocks of bitlength $2w$. Each of r rounds updates both w-bit data halves, using 2 subkeys in an input transformation and 2 more for each round. The only operations used, all on w-bit words, are addition mod 2^w ($\boxplus$), XOR ($\oplus$), and rotations (left $\hookleftarrow$ and right $\hookrightarrow$). The XOR operation is linear, while the addition may be considered nonlinear depending on the metric for linearity. The data-dependent rotations featured in RC5 are the main nonlinear operation used: $x \hookleftarrow y$ denotes cyclically shifting a w-bit word left y bits; the rotation-count y may be reduced mod w (the low-order $\lg(w)$ bits of y suffice). The key schedule expands a key of b bytes into $2r + 2$ subkeys K_i of w bits each. Regarding packing/unpacking bytes into words, the byte-order is *little-endian*: for $w = 32$, the first plaintext byte goes in the low-order end of A, the fourth in A's high-order end, the fifth in B's low order end, and so on.

7.115 Algorithm RC5 encryption (w-bit wordsize, r rounds, b-byte key)

INPUT: $2w$-bit plaintext $M = (A, B)$; r; key $K = K[0] \dots K[b-1]$.
OUTPUT: $2w$-bit ciphertext C. (For decryption, see Note 7.117.)

1. Compute $2r+2$ subkeys $K_0, \dots, K_{2r+1}$ by Algorithm 7.116 from inputs K and r.
2. $A \leftarrow A \boxplus K_0$, $B \leftarrow B \boxplus K_1$. (Use addition modulo 2^w.)
3. For i from 1 to r do: $A \leftarrow ((A \oplus B) \hookleftarrow B) \boxplus K_{2i}$, $B \leftarrow ((B \oplus A) \hookleftarrow A) \boxplus K_{2i+1}$.
4. The output is $C \leftarrow (A, B)$.

7.116 Algorithm RC5 key schedule

INPUT: word bitsize w; number of rounds r; b-byte key $K[0] \dots K[b-1]$.
OUTPUT: subkeys $K_0, \dots, K_{2r+1}$ (where K_i is w bits).

1. Let $u = w/8$ (number of bytes per word) and $c = \lceil b/u \rceil$ (number of words K fills). Pad K on the right with zero-bytes if necessary to achieve a byte-count divisible by u (i.e., $K[j] \leftarrow 0$ for $b \le j \le c \cdot u - 1$). For i from 0 to $c-1$ do: $L_i \leftarrow \sum_{j=0}^{u-1} 2^{8j}$ $K[i \cdot u + j]$ (i.e., fill L_i low-order to high-order byte using each byte of $K[\cdot]$ once).
2. $K_0 \leftarrow P_w$; for i from 1 to $2r+1$ do: $K_i \leftarrow K_{i-1} \boxplus Q_w$. (Use Table 7.14.)
3. $i \leftarrow 0$, $j \leftarrow 0$, $A \leftarrow 0$, $B \leftarrow 0$, $t \leftarrow \max(c, 2r+2)$. For s from 1 to $3t$ do:
   (a) $K_i \leftarrow (K_i \boxplus A \boxplus B) \hookleftarrow 3$, $A \leftarrow K_i$, $i \leftarrow i + 1 \bmod (2r+2)$.
   (b) $L_j \leftarrow (L_j \boxplus A \boxplus B) \hookleftarrow (A \boxplus B)$, $B \leftarrow L_j$, $j \leftarrow j + 1 \bmod c$.
4. The output is $K_0, K_1, \dots, K_{2r+1}$. (The L_i are not used.)

7.117 Note (*RC5 decryption*) Decryption uses the Algorithm 7.115 subkeys, operating on ciphertext $C = (A, B)$ as follows (subtraction is mod 2^w, denoted $\boxminus$). For i from r down to 1 do: $B \leftarrow ((B \boxminus K_{2i+1}) \hookrightarrow A) \oplus A$, $A \leftarrow ((A \boxminus K_{2i}) \hookrightarrow B) \oplus B$. Finally $M \leftarrow (A \boxminus K_0, B \boxminus K_1)$.

w :	16	32	64
P_w :	B7E1	B7E15163	B7E15162 8AED2A6B
Q_w :	9E37	9E3779B9	9E3779B9 7F4A7C15

Table 7.14: *RC5 magic constants (given as hex strings).*

7.118 Example (*RC5–32/12/16 test vectors*) For the hexadecimal plaintext M = `65C178B2 84D197CC` and key K = `5269F149 D41BA015 2497574D 7F153125`, RC5 with $w = 32$, $r = 12$, and $b = 16$ generates ciphertext C = `EB44E415 DA319824`. □

7.7.3 Other block ciphers

LOKI'91 (and earlier, LOKI'89) was proposed as a DES alternative with a larger 64-bit key, a matching 64-bit blocksize, and 16 rounds. It differs from DES mainly in key-scheduling and the f-function. The f-function of each round uses four identical 12-to-8 bit S-boxes, 4 input bits of which select one of 16 functions, each of which implements exponentiation with a fixed exponent in a different representation of GF(2^8). While no significant exploitable weaknesses have been found in LOKI'91 when used for encryption, related-key attacks (see page 281) are viewed as a certificational weakness.

Khufu and Khafre are DES-like ciphers which were proposed as fast software-oriented alternatives to DES. They have 64-bit blocks, 8×32 bit S-boxes, and a variable number of rounds (typically 16, 24, or 32). Khufu keys may be up to 512 bits. Khafre keys have bitlength that is a multiple of 64 (64 and 128-bit keys are typical); 64 key bits are XORed onto the data block before the first and thereafter following every 8 rounds. Whereas a DES round involves eight 6-to-4 bit S-boxes, one round of Khufu involves a single 8-to-32 bit table look-up, with a different S-box for every 8 rounds. The S-boxes are generated pseudorandomly from the user key. Khafre uses fixed S-boxes generated pseudorandomly from an initial S-box constructed from random numbers published by the RAND corporation in 1955. Under the best currently known attacks, 16-round Khufu and 24-round Khafre are each more difficult to break than DES.

7.8 Notes and further references

§7.1

The extensive and particularly readable survey by Diffie and Hellman [347], providing a broad introduction to cryptography especially noteworthy for its treatment of Hagelin and rotor machines and the valuable annotated bibliography circa 1979, is a source for much of the material in §7.2, §7.3, and §7.4 herein. Aside from the appearance of DES [396] in the mid 1970s and FEAL [884] later in the 1980s, prior to 1990 few fully-specified serious symmetric block cipher proposals were widely available or discussed. (See Chapter 15 for Pohlig and Hellman's 1978 discrete exponentiation cipher.) With the increasing feasibility of exhaustive search on 56-bit DES keys, the period 1990-1995 resulted in a large number of proposals, beginning with PES [728], the preliminary version of IDEA [730]. The *Fast Software Encryption* workshops (Cambridge, U.K., Dec. 1993; Leuven, Belgium, Dec. 1994; and again Cambridge, Feb. 1996) were a major stimulus and forum for new proposals.

The most significant cryptanalytic advances over the 1990-1995 period were Matsui's linear cryptanalysis [796, 795], and the differential cryptanalysis of Biham and Shamir [138] (see also [134, 139]). Extensions of these included the differential-linear analysis by Langford and Hellman [741], and the truncated differential analysis of Knudsen [686]. For additional background on linear cryptanalysis, see Biham [132]; see also Matsui and Yamagishi [798] for a preliminary version of the method. Additional background on differential cryptanalysis is provided by many authors including Lai [726], Lai, Massey, and Murphy [730], and Coppersmith [271]; although more efficient 6-round attacks are known, Stinson [1178] provides detailed examples of attacks on 3-round and 6-round DES. Regarding both linear and differential cryptanalysis, see also Knudsen [684] and Kaliski and Yin [656].

§7.2

Lai [726, Chapter 2] provides an excellent concise introduction to block ciphers, including a lucid discussion of design principles (recommended for all block cipher designers). Regarding text dictionary and matching ciphertext attacks (Note 7.8), see Coppersmith, Johnson, and Matyas [278]. Rivest and Sherman [1061] provide a unified framework for randomized encryption (Definition 7.3); a common example is the use of random "salt" appended to passwords prior to password encryption in some operating systems (§10.2.3). Fact 7.9 is due to Shannon [1121], whose contributions are many (see below).

The four basic modes of operation (including k-bit OFB feedback) were originally defined specifically for DES in 1980 by FIPS 81 [398] and in 1983 by ANSI X3.106 [34], while ISO 8732 [578] and ISO/IEC 10116 [604], respectively, defined these modes for general 64-bit and general n-bit block ciphers, mandating n-bit OFB feedback (see also Chapter 15). Brassard [192] gives a concise summary of modes of operation; Davies and Price [308] provide a comprehensive discussion, including OFB cycling (Note 7.24; see also Jueneman [643] and Davies and Parkin [307]), and a method for encrypting incomplete CBC final blocks without data expansion, which is important if plaintext must be encrypted and returned into its original store. See Voydock and Kent [1225] for additional requirements on IVs. Recommending $r = s$ for maximum strength, ISO/IEC 10116 [604] specifies the CFB variation of Example 7.19, and provides extensive discussion of properties of the various modes. The counter mode (Example 7.23) was suggested by Diffie and Hellman [347].

The 1977 exhaustive DES key search machine (Example 7.27) proposed by Diffie and Hellman [346] contained 10^6 DES chips, with estimated cost US$20 million (1977 technology) and 12-hour expected search time; Diffie later revised the estimate upwards one order of magnitude in a BNR Inc. report (US$50 million machine, 2-day expected search time, 1980 technology). Diffie and Hellman noted the feasibility of a ciphertext-only attack (Example 7.28), and that attempting to preclude exhaustive search by changing DES keys more frequently, at best, doubles the expected search time before success.

Subsequently Wiener [1241] provided a gate-level design for a US$1 million machine (1993 technology) using 57 600 DES chips with expected success in 3.5 hours. Each chip contains 16 pipelined stages, each stage completing in one clock tick at 50 MHz; a chip with full pipeline completes a key test every 20 nanoseconds, providing a machine $57\,600 \times 50$ times faster than the 1142 years noted in FIPS 74 [397] as the time required to check 2^{55} keys if one key can be tested each microsecond. Comparable key search machines of equivalent cost by Eberle [362] and Wayner [1231] are, respectively, 55 and 200 times slower, although the former does not require a chip design, and the latter uses a general-purpose machine. Wiener also noted adaptations of the ECB known-plaintext attack to other 64-bit modes (CBC, OFB, CFB) and 1-bit and 8-bit CFB.

Even and Goldreich [376] discuss the unicity distance of cascade ciphers under known-plaintext attack (Fact 7.35), present a generalized time-memory meet-in-the-middle trade-off (Note 7.38), and give several other concise results on cascades, including that under reasonable assumptions, the number of permutations realizable by a cascade of L random cipher stages is, with high probability, 2^{Lk}.

Diffie and Hellman [346] noted the meet-in-the-middle attack on double encryption (Fact 7.33), motivating their recommendation that multiple encipherment, if used, should be at least three-fold; Hoffman [558] credits them with suggesting E-E-E triple encryption with three independent keys. Merkle's June 1979 thesis [850] explains the attack on two-key triple-encryption of Fact 7.39 (see also Merkle and Hellman [858]), and after noting Tuchman's proposal of two-key E-D-E triple encryption in a June 1978 conference talk (*National Computer Conference*, Anaheim, CA; see also [1199]), recommended that E-D-E be used with three independent keys: $E_{K3}(E_{K2}^{-1}(E_{K1}(x)))$. The two-key E-D-E idea, adopted in ANSI X9.17 [37] and ISO 8732 [578], was reportedly conceived circa April 1977 by Tuchman's colleagues, Matyas and Meyer. The attack of Fact 7.40 is due to van Oorschot and Wiener [1206]. See Coppersmith, Johnson, and Matyas [278] for a proposed construction for a triple-DES algorithm. Other techniques intended to extend the strength of DES include the *DESX* proposal of Rivest as analyzed by Kilian and Rogaway [672], and the work of Biham and Biryukov [133].

Hellman [549] proposes a time-memory tradeoff for exhaustive key search on a cipher with $N = 2^m$ ciphertexts requiring a chosen-plaintext attack, $O(N^{2/3})$ time and $O(N^{2/3})$ space after an $O(N)$ precomputation; search time can be reduced somewhat by use of Rivest's suggestion of distinguished points (see Denning [326, p.100]). Kusuda and Matsumoto [722] recently extended this analysis. Fiat and Naor [393] pursue time-memory tradeoffs for more general functions. Amirazizi and Hellman [25] note that time-memory tradeoff with constant time-memory product offers no asymptotic cost advantage over exhaustive search; they examine tradeoffs between time, memory, and parallel processing, and using standard parallelization techniques, propose under a simplified model a search machine architecture for which doubling the machine budget (cost) increases the solution rate fourfold. This approach may be applied to exhaustive key search on double-encryption, as can the parallel collision search technique of van Oorschot and Wiener [1207, 1208]; see also Quisquater and Delescaille [1017, 1018].

Regarding Note 7.41, see Biham [131] (and earlier [130]) as well as Coppersmith, Johnson, and Matyas [278]. Biham's analysis on DES and FEAL shows that, in many cases, the use of intermediate data as feedback into an intermediate stage reduces security. 15 years earlier, reflecting on his chosen-plaintext attack on two-key triple-encryption, Merkle [850, p.149] noted "multiple encryption with any cryptographic system is liable to be much less secure than a system designed originally for the longer key".

Maurer and Massey [822] formalize Fact 7.42, where "break" means recovering plaintext from ciphertext (under a known-plaintext attack) or recovering the key; the results hold also for chosen-plaintext and chosen-ciphertext attack. They illustrate, however, that the earlier result and commonly-held belief proven by Even and Goldreich [376] – that a cascade is as strong as any of its component ciphers – requires the important qualifying (and nonpractical) assumption that an adversary will not exploit statistics of the underlying plaintext; thus, the intuitive result is untrue for most practical ciphertext-only attacks.

§7.3

Kahn [648] is the definitive historical reference for classical ciphers and machines up to 1967, including much of §7.3 and the notes below. The selection of classical ciphers presented largely follows Shannon's lucid 1949 paper [1121]. Standard references for classical cryptanalysis include Friedman [423], Gaines [436], and Sinkov [1152]. More recent books providing expository material on classical ciphers, machines, and cryptanalytic examples include Beker and Piper [84], Meyer and Matyas [859], Denning [326], and Davies and Price [308].

Polyalphabetic ciphers were invented circa 1467 by the Florentine architect Alberti, who devised a cipher disk with a larger outer and smaller inner wheel, respectively indexed by plaintext and ciphertext characters. Letter alignments defined a simple substitution, modified by rotating the disk after enciphering a few words. The first printed book on cryptography, *Polygraphia*, written in 1508 by the German monk Trithemius and published in 1518, contains the first *tableau* – a square table on 24 characters listing all shift substitutions for a fixed ordering of plaintext alphabet characters. Tableau rows were used sequentially to substitute one plaintext character each for 24 letters, where-after the same tableau or one based on a different alphabet ordering was used. In 1553 Belaso (from Lombardy) suggested using an easily changed key (and key-phrases as memory aids) to define the fixed alphabetic (shift) substitutions in a polyalphabetic substitution. The 1563 book of Porta (from Naples) noted the ordering of tableau letters may define arbitrary substitutions (vs. simply shifted alphabets).

Various polyalphabetic auto-key ciphers, wherein the key changes with each message (the alteration depending on the message), were explored in the 16th century, most significantly by the Frenchman B. de Vigenère. His 1586 book *Traicté des Chiffres* proposed the combined use of a mixed tableau (mixed alphabet on both the tableau top and side) and an auto-keying technique (cf. Example 7.61). A single character served as a priming key to select the tableau row for the first character substitution, where-after the ith plaintext character determined the alphabet (tableau row) for substituting the next. The far less secure simple Vigenère cipher (Definition 7.53) is incorrectly attributed to Vigenère.

The Playfair cipher (Example 7.51), popularized by L. Playfair in England circa 1854 and invented by the British scientist C. Wheatstone, was used as a British field cipher [648, p.6]. J. Mauborgne (see also the Vernam and PURPLE ciphers below) is credited in 1914 with the first known solution of this digram cipher.

The Jefferson cylinder was designed by American statesman T. Jefferson, circa 1790-1800. In 1817, fellow American D. Wadsworth introduced the principle of plaintext and ciphertext alphabets of different lengths. His disk (cf. Alberti above) implemented a cipher similar to Trithemius' polyalphabetic substitution, but wherein the various alphabets were brought into play irregularly in a plaintext-dependent manner, foreshadowing both the polyalphabetic ciphers of later 20th century rotor machines, and the concept of chaining. The inner disk had 26 letters while the outer had an additional 7 digits; one full revolution of the larger caused the smaller to advance 7 characters into its second revolution. The driving disk was always turned in the same clockwise sense; when the character revealed through an aperture in the plaintext disk matched the next plaintext character, that visible through a corresponding ciphertext aperture indicated the resulting ciphertext. In 1867, Wheatstone displayed an independently devised similar device thereafter called the *Wheatstone disc*, receiving greater attention although less secure (having disks of respectively 26 and 27 characters, the extra character a plaintext space).

Vernam [1222] recorded his idea for telegraph encryption in 1917; a patent filed in September 1918 was issued July 1919. Vernam's device combined a stream of plaintext (5-bit Baudot coded) characters, via XOR, with a keystream of 5-bit (key) values, resulting in the *Vernam cipher* (a term often used for related techniques). This, the first polyalphabetic substitution automated using electrical impulses, had period equal to the length of the key stream; each 5-bit key value determined one of 32 fixed mono-alphabetic substitutions. Credit for the actual *one-time system* goes to J. Mauborgne (U.S. Army) who, after seeing Vernam's device with a repeated tape, realized that use of a random, non-repeated key improved security. While Vernam's device was a commercial failure, a related German system engineered by W. Kunze, R. Schauffler, and E. Langlotz was put into practice circa 1921-1923 for German diplomatic communications; their encryption system, which involved manually adding a key string to decimal-coded plaintext, was secured by using as the numerical key a random non-repeating decimal digit stream – the original *one-time pad*. Pads of 50 numbered sheets were used, each with 48 five-digit groups; no pads were repeated aside for one identical pad for a communicating partner, and no sheet was to be used twice; sheets were destroyed once used. The Vernam cipher proper, when used as a one-time system, involves only 32 alphabets, but provides more security than rotor machines with a far greater number of alphabets because the latter eventually repeat, whereas there is total randomness (for each plaintext character) in selecting among the 32 Vernam alphabets.

The matrix cipher of Example 7.52 was proposed in 1929 by Hill [557], providing a practical method for polygraphic substitution, albeit a linear transformation susceptible to known-plaintext attack. Hill also recognized that using an involution as the encryption mapping allowed the same function to provide decryption. Recent contributions on homophonic substitution include Günther [529] and Jendal, Kuhn, and Massey [636].

Among the unrivalled cryptanalytic contributions of the Russian-born American Friedman is his 1920 Riverbank Publication no.22 [426] on cryptanalysis using the index of coincidence. Friedman coined the term *cryptanalysis* in 1920, using it in his 1923 book *Elements of Cryptanalysis* [425], a 1944 expansion of which, *Military Cryptanalysis* [423], remains highly recommended. The method of Kasiski (from West Prussia) was originally published in 1863; see Kahn [648, pp.208-213] for a detailed example. The discussion on IC and MR follows that of Denning [326], itself based on Sinkov [1152]. Fact 7.75 follows from a standard expectation computation weighted by κ_p or κ_r depending on whether the second of a pair of randomly selected ciphertext characters is from the same ciphertext alphabet or one of the $t-1$ remaining alphabets. The values in Table 7.1 are from Kahn [648], and vary somewhat over time as languages evolve.

Friedman teaches how to cryptanalyze running-key ciphers in his (circa 1918) Riverbank Publication no.16, *Methods for the Solution of Running-Key Ciphers*; the two basic techniques are outlined by Diffie and Hellman [347]. The first is a *probable word* attack wherein an attacker guesses an (e.g., 10 character) word hopefully present in underlying text, and subtracts that word (mod 26) from all possible starting locations in the ciphertext in hopes of finding a recognizable 10-character result, where-after the guessed word (as either partial running-key or plaintext) might be extended using context. Probable-word attacks also apply to polyalphabetic substitution. The second technique is based on the fact that each ciphertext letter c results from a pair of plaintext/running-key letters (m_i, m'_i), and is most likely to result from such pairs wherein both m_i and m'_i are high-frequency characters; one isolates the highest-probability pairs for each such ciphertext character value c, makes trial assumptions, and attempts to extend apparently successful guesses by similarly decrypting adjacent ciphertext characters; see Denning [326, p.83] for a partial example. Diffie and Hellman [347] note Fact 7.59 as an obvious method that is little-used (modern ciphers being more convenient); their suggestion that use of four iterative running keys is unbreakable follows from English being 75% redundant. They also briefly summarize various *scrambling* techniques (encryption via analog rather than digital methods), noting that analog scramblers are sometimes used in practice due to lower bandwidth and cost requirements, although such known techniques appear relatively insecure (possibly an inherent characteristic) and their use is waning as digital networks become prevalent.

Denning [326] tabulates digrams into high, medium, low, and rare classes. Konheim [705, p.24] provides transition probabilities $p(t|s)$, the probability that the next letter is t given that the current character is s in English text, in a table also presented by H. van Tilborg [1210]. Single-letter distributions in plaintext languages other than English are given by Davies and Price [308]. The letter frequencies in Figure 7.5, which should be interpreted only as an estimate, were derived by Meyer and Matyas [859] using excerpts totaling 4 million characters from the 1964 publication: W. Francis, *A Standard Sample of Present-Day Edited American English for Use with Digital Computers*, Linguistics Dept., Brown University, Providence, Rhode Island, USA. Figure 7.6 is based on data from Konheim [705, p.19] giving an estimated probability distribution of 2-grams in English, derived from a sample of size 67 320 digrams.

See Shannon [1122] and Cover and King [285] regarding redundancy and Fact 7.67. While not proven in any concrete manner, Fact 7.68 is noted by Friedman [424] and generally accepted. Unicity distance was defined by Shannon [1121]. Related issues are discussed in detail in various appendices of Meyer and Matyas [859]. Fact 7.71 and the random cipher model are due to Shannon [1121]; see also Hellman [548].

Diffie and Hellman [347] give an instructive overview of rotor machines (see also Denning [326]), and note their use in World War II by the Americans in their highest level system, the British, and the Germans (Enigma); they also give Fact 7.63 and the number of characters required under ciphertext-only and known-plaintext attacks (Note 7.66). Beker and Piper [84] provide technical details of the Hagelin M-209, as does Kahn [648, pp.427-431] who notes its remarkable compactness and weight: 3.25 x 5.5 x 7 inches and 6 lb. (including case); see also Barker [74], Morris [906], and Rivest [1053]. Davies and Price [308] briefly discuss the Enigma, noting it was cryptanalyzed during World War II in Poland, France, and then in the U.K. (Bletchley Park); see also Konheim [705].

The Japanese PURPLE cipher, used during World War II, was a polyalphabetic cipher cryptanalyzed August 1940 [648, p.18-23] by Friedman's team in the U.S. Signal Intelligence Service, under (Chief Signal Officer) Mauborgne. The earlier RED cipher used two rotor arrays; preceding it, the ORANGE system implemented a vowels-to-vowels, consonants-to-consonants cipher using sets of rotors.

§7.4

The concept of *fractionation*, related to product ciphers, is noted by Feistel [387], Shannon [1121], and Kahn [648, p.344] who identifies this idea in an early product cipher, the WWI German *ADFGVX* field cipher. As an example, an encryption function might operate on a block of $t = 8$ plaintext characters in three stages as follows: the first substitutes two symbols for each individual character; the second transposes (mixes) the substituted symbols among themselves; the third re-groups adjacent resulting symbols and maps them back to the plaintext alphabet. The action of the transposition on partial (rather than complete) characters contributes to the strength of the principle.

Shannon [1121, §5 and §23-26] explored the idea of the product of two ciphers, noted the principles of confusion and diffusion (Remark 1.36), and introduced the idea of a *mixing transformation* F (suggesting a preliminary transposition followed by a sequence of alternating substitution and simple linear operations), and combining ciphers in a product using an intervening transformation F. Transposition and substitution, respectively, rest on the principles of diffusion and confusion. Harpes, Kramer, and Massey [541] discuss a general model for iterated block ciphers (cf. Definition 7.80).

The name *Lucifer* is associated with two very different algorithms. The first is an SP network described by Feistel [387], which employs (bitwise nonlinear) 4×4 invertible S-boxes; the second, closely related to DES (albeit significantly weaker), is described by Smith [1160] (see also Sorkin [1165]). Principles related to both are discussed by Feistel, Notz, and Smith [388]; both are analyzed by Biham and Shamir [138], and the latter in greater detail by Ben-Aroya and Biham [108] whose extension of differential cryptanalysis allows, using 2^{36} chosen plaintexts and complexity, attack on 55% of the key space in Smith's Lucifer – still infeasible in practice, but illustrating inferiority to DES despite the longer 128-bit key.

Feistel's product cipher Lucifer [387], instantiated by a blocksize $n = 128$, consists of an unspecified number of alternating substitution and permutation (transposition) stages, using a fixed (unpublished) n-bit permutation P and 32 parallel identical S-boxes each effecting a mapping S_0 or S_1 (fixed but unpublished bijections on $\{0,1\}^4$), depending on the value of one key bit; the unpublished key schedule requires 32-bits per S-box stage. Each stage operates on all n bits; decryption is by stage-wise inversion of P and S_i.

The structure of so-called Feistel ciphers (Definition 7.81) was first introduced in the Lucifer algorithm of Smith [1160], the direct predecessor of DES. This 16-round algorithm with 128-bit key operates on alternating half-blocks of a 128-bit message block with a simplified f function based on two published invertible 4×4 bit S-boxes S_0 and S_1 (cf. above). Feistel, Notz, and Smith [388] discuss both the abstract Feistel cipher structure (suggesting its use with non-invertible S-boxes) and SP networks based on invertible (distinct) S-boxes. Suggestions for SP networks include the use of single key bits to select one of two mappings (a fixed bijection or its inverse) from both S-boxes and permutation boxes; decryption then uses a reversed key schedule with complemented key. They also noted the multi-round *avalanche effect* of changing a single input bit, subsequently pursued by Kam and Davida [659] in relation to SP networks and S-boxes having a *completeness* property: for every pair of bit positions i, j, there must exist at least two input blocks x, y which differ only in bit i and whose outputs differ in at least bit j. More simply, a function is *complete* if each output bit depends on all input bits. Webster and Tavares [1233] proposed the more stringent *strict avalanche criterion*: whenever one input bit is changed, every output bit must change with probability 1/2.

DES resulted from IBM's submission to the 1974 U.S. National Bureau of Standards (NBS) solicitation for encryption algorithms for the protection of computer data. The original specification is the 1977 U.S. Federal Information Processing Standards Publication 46 [396], reprinted in its entirety as Appendix A in Meyer and Matyas [859]. DES is now specified in FIPS 46–2, which succeeded FIPS 46–1; the same cipher is defined in the American standard ANSI X3.92 [33] and referred to as the Data Encryption Algorithm (DEA). Differences between FIPS 46/46–1 and ANSI X3.92 included the following: these earlier FIPS required that DES be implemented in hardware and that the parity bits be used for parity; ANSI X3.92 specifies that the parity bits *may* be used for parity. Although no purpose was stated by the DES designers for the permutations IP and IP^{-1}, Preneel et al. [1008] provided some evidence of their cryptographic value in the CFB mode.

FIPS 81 [398] specifies the common modes of operation. Davies and Price [308] provide a comprehensive discussion of both DES and modes of operation; see also Diffie and Hellman [347], and the extensive treatment of Meyer and Matyas [859]. The survey of Smid and Branstad [1156] discusses DES, its history, and its use in the U.S. government. Test vectors for various modes of DES, including the ECB vectors of Example 7.86, may be found in ANSI X3.106 [34]. Regarding exhaustive cryptanalysis of DES and related issues, see also the notes under §7.2.

The 1981 publication FIPS 74 [397] notes that DES is not (generally) commutative under two keys, and summarizes weak and semi-weak keys using the term *dual keys* to include both (weak keys being self-dual); see also Davies [303] and Davies and Price [308]. Coppersmith [268] noted Fact 7.90; Moore and Simmons [900] pursue weak and semi-weak DES keys and related phenomena more rigorously.

The 56-bit keylength of DES was criticized from the outset as being too small (e.g., see Diffie and Hellman [346], and p.272 above). Claims which have repeatedly arisen and been denied (e.g., see Tuchman [1199]) over the past 20 years regarding built-in weaknesses of DES (e.g., trap-door S-boxes) remain un-substantiated. Fact 7.91 is significant in that if the permutation group were closed under composition, DES would fall to a known-plaintext attack requiring 2^{28} steps – see Kaliski, Rivest, and Sherman [654], whose cycling experiments provided strong evidence against this. Campbell and Wiener [229] prove the fact conclusively (and give the stated lower bound), through their own cycling experiments utilizing collision key search and an idea outlined earlier by Coppersmith [268] for establishing a lower bound on the group size; they attribute to Coppersmith the same result (in unpublished work), which may also be deduced from the cycle lengths published by Moore and Simmons [901].

Countless papers have analyzed various properties of DES; Davies and Price [308, pp.73-75] provide a partial summary to 1987. Subsequent to the discovery of differential cryptanalysis (DC) by Biham and Shamir, Coppersmith [271] explains how DES was specifically designed 15 years earlier to counter DC, citing national security concerns regarding the design team publishing neither the attack nor design criteria; then gives the (relevant) design criteria – some already noted by others, e.g., see Hellman et al. [552] – for DES S-boxes and the permutation P, explaining how these preclude DC. Coppersmith notes elements of DC were present in the work of den Boer [322], followed shortly by Murphy [913]. DES was not, however, specifically designed to preclude linear cryptanalysis (LC); Matsui [797] illustrates the order of the 8 DES S-boxes, while a strong (but not optimal) choice against DC, is relatively weak against LC, and that DES can be strengthened (vs. DC and LC) by carefully re-arranging these. Despite Remark 7.93, a DES key has actually been recovered by Matsui [795] using LC under experimental conditions (using 2^{43} known-plaintext pairs from randomly generated plaintexts, and 2^{43} complexity running twelve 99 MHz machines over 50 days); such a result remains to be published for exhaustive search or DC.

Ben-Aroya and Biham [108] note that often suggestions to redesign DES, some based on design criteria and attempts to specifically resist DC, have resulted in (sometimes far) weaker systems, including the RDES (randomized DES) proposal of Koyama and Terada [709], which fall to variant attacks. The lesson is that in isolation, individual design principles do not guarantee security.

DES alternatives are sought not only due to the desire for a keylength exceeding 56 bits, but also because its bit-oriented operations are inconvenient in conventional software implementations, often resulting in poor performance; this makes triple-DES less attractive. Regarding fast software implementations of DES, see Shepherd [1124], Pfitzmann and Aßmann [970], and Feldmeier and Karn [391].

§7.5

FEAL stimulated the development of a sequence of advanced cryptanalytic techniques of unparalleled richness and utility. While it appears to remain relatively secure when iterated a sufficient number of rounds (e.g., 24 or more), this defeats its original objective of speed. FEAL-4 as presented at Eurocrypt'87 (Abstracts of Eurocrypt'87, April 1987) was found to have certain vulnerabilities by den Boer (unpublished Eurocrypt'87 rump session talk), resulting in Shimizu and Miyaguchi [1126] (or see Miyaguchi, Shiraishi, and Shimizu [887]) increasing FEAL to 8 rounds in the final proceedings. In 1988 den Boer [322] showed FEAL-4 vulnerable to an adaptive chosen plaintext attack with 100 to 10 000 plaintexts. In 1990, Gilbert and Chassé [455] devised a chosen-plaintext attack (called a statistical meet-in-the-middle attack) on FEAL-8 requiring 10 000 pairs of plaintexts, the bitwise XOR of each pair being selected to be an appropriate constant (thus another early variant of differential cryptanalysis).

FEAL-N with N rounds, and its extension FEAL-NX with 128-bit key (Notes 7.97 and 7.98) were then published by Miyaguchi [884] (or see Miyaguchi et al. [885]), who nonetheless opined that chosen-plaintext attacks on FEAL-8 were not practical threats. However, improved chosen-plaintext attacks were subsequently devised, as well as known-plaintext attacks. Employing den Boer's G function expressing linearity in the FEAL f-function, Murphy [913] defeated FEAL-4 with 20 chosen plaintexts in under 4 hours (under 1 hour for most keys) on a Sun 3/60 workstation. A statistical method of Tardy-Corfdir and Gilbert [1187] then allowed a known-plaintext attack on FEAL-4 (1000 texts; or 200 in an announced improvement) and FEAL-6 ($2 \times 10\,000$ texts), involving linear approximation of FEAL S-boxes. Thereafter, the first version of linear cryptanalysis (LC) introduced by Matsui and Yamagishi [798] allowed known-plaintext attack of FEAL-4 (5 texts, 6 minutes on a 25MHz 68040 processor), FEAL-6 (100 texts, 40 minutes), and FEAL-8 (2^{28} texts, in time equivalent to exhaustive search on 50-bit keys); the latter betters the 2^{38} texts required for FEAL-8 by Biham and Shamir [136] in their known-plaintext conversion of differential cryptanalysis (DC). Biham and Shamir [138, p.101] later implemented a DC chosen-plaintext attack recovering FEAL-8 keys in two minutes on a PC using 128 chosen pairs, the program requiring 280K bytes of storage. Biham [132] subsequently used LC to defeat FEAL-8 with 2^{24} known-plaintexts in 10 minutes on a personal computer. Ohta and Aoki [943] suggest that FEAL-32 is as secure as DES against DC, while FEAL-16 is as secure as DES against certain restricted forms of LC.

Differential-linear cryptanalysis was introduced by Langford and Hellman [741], combining linear and differential cryptanalysis to allow a reduced 8-round version of DES to be attacked with fewer chosen-plaintexts than previous attacks. Aoki and Ohta [53] refined these ideas for FEAL-8 yielding a differential-linear attack requiring only 12 chosen texts and 35 days of computer time (cf. Table 7.10).

Test vectors for FEAL-N and FEAL-NX (Example 7.99) are given by Miyaguchi [884]. The DC attack of Biham and Shamir [137], which finds FEAL-N subkeys themselves, is equally as effective on FEAL-NX. Biham [132] notes that an LC attack on FEAL-N is possible with less than 2^{64} known plaintexts (and complexity) for up to $N = 20$. For additional discussion of properties of FEAL, see Biham and Shamir [138, §6.3].

§7.6

The primary reference for IDEA is Lai [726]. A preliminary version introduced by Lai and Massey [728] was named PES (Proposed Encryption Standard). Lai, Massey, and Murphy [730] showed that a generalization (see below) of differential cryptanalysis (DC) allowed recovery of PES keys, albeit requiring all 2^{64} possible ciphertexts (cf. exhaustive search of 2^{128} operations). Minor modifications resulted in IPES (Improved PES): in stage r, $1 \leq r \leq 9$, the group operations keyed by $K_2^{(r)}$ and $K_4^{(r)}$ ($\boxplus$ and $\odot$ in Figure 7.11) were reversed from PES; the permutation on 16-bit blocks after stage r, $1 \leq r \leq 9$, was altered; and necessary changes were made in the decryption (but not encryption) key schedule. IPES was commercialized under the name IDEA, and is patented (see Chapter 15).

The ingenious design of IDEA is supported by a careful analysis of the interaction and algebraic incompatibilities of operations across the groups $(\mathbb{F}_2{}^n, \oplus)$, $(\mathbb{Z}_{2^n}, \boxplus)$, and $(\mathbb{Z}^*_{2^n+1}, \odot)$. The design of the MA structure (see Figure 7.11) results in IDEA being "complete" after a single round; for other security properties, see Lai [726]. Regarding mixing operations from different algebraic systems, see also the 1974 examination by Grossman [522] of transformations arising by alternating mod 2^n and mod 2 addition ($\oplus$), and the use of arithmetic modulo $2^{32} - 1$ and $2^{32} - 2$ in MAA (Algorithm 9.68).

Daemen [292, 289] identifies several classes of so-called *weak keys for IDEA*, and notes a small modification to the key schedule to eliminate them. The largest is a class of 2^{51} keys for which membership can be tested in two encryptions plus a small number of computations, whereafter the key itself can be recovered using 16 chosen plaintext-difference encryptions, on the order of 2^{16} group operations, plus 2^{17} key search encryptions. The probability of a randomly chosen key being in this class is $2^{51}/2^{128} = 2^{-77}$. A smaller number of weak key blocks were observed earlier by Lai [726], and dismissed as inconsequential. The analysis of Meier [832] revealed no attacks feasible against full 8-round IDEA, and supports the conclusion of Lai [726] that IDEA appears to be secure against DC after 4 of its 8 rounds (cf. Note 7.107). Daemen [289] also references attacks on reduced-round variants of IDEA. While linear cryptanalysis (LC) can be applied to any iterated block cipher, Harpes, Kramer, and Massey [541] provide a generalization thereof; IDEA and SAFER K-64 are argued to be secure against this particular generalization.

Lai, Massey, and Murphy [730] (see also Lai [726]) generalized DC to apply to *Markov ciphers* (which they introduced for this purpose; DES, FEAL, and LOKI are all examples under the assumption of independent round keys) including IDEA; broadened the notion of a *difference* from that based on $\oplus$ to: $\Delta X = X \otimes (X^*)^{-1}$ where $\otimes$ is a specified group operation and $(X^*)^{-1}$ is the group inverse of an element X^*; and defined an *i-round differential* (as opposed to an i-round characteristic used by Biham and Shamir [138] on DES) to be a pair (α, β) such that two distinct plaintexts with difference $\Delta X = \alpha$ results in a pair of round i outputs with difference β.

Decimal values corresponding to Tables 7.12 and 7.13 may be found in Lai [726]. A table-based alternative for multiplication mod $2^{16}+1$ (cf. Note 7.104) is to look up the anti-log of $\log_\alpha(a) + \log_\alpha(b) \bmod 2^{16}$, relative to a generator α of $\mathbb{Z}^*_{2^{16}+1}$; the required tables, however, are quite large.

§7.7

Massey [787] introduced SAFER K-64 with a 64-bit key and initially recommended 6 rounds, giving a reference implementation and test vectors (cf. Example 7.114). It is not patented. Massey [788] then published SAFER K-128 (with a reference implementation), differing only in its use of a non-proprietary (and backwards compatible) key schedule accommodating 128-bit keys, proposed by a Singapore group; 10 rounds were recommended (12 maximum). Massey [788] gave further justification for design components of SAFER K-64. Vaudenay [1215] showed SAFER K-64 is weakened if the S-box mapping (Remark 7.112) is replaced by a random permutation.

Knudsen [685] proposed the modified key schedule of Note 7.110 after finding a weakness in 6-round SAFER K-64 that, while not of practical concern for encryption (with 2^{45} chosen plaintexts, it finds 8 bits of the key), permitted collisions when using the cipher for hashing. This and a subsequent certificational attack on SAFER K-64 by S. Murphy (to be published) lead Massey ("Strengthened key schedule for the cipher SAFER", posted to the USENET newsgroup sci.crypt, September 9 1995) to advise adoption of the new key schedule, with the resulting algorithm distinguished as SAFER SK-64 with 8 rounds recommended (minimum 6, maximum 10); an analogous change to the 128-bit key schedule yields SAFER SK-128 for which 10 rounds remain recommended (maximum 12). A new variant of DC by Knudsen and Berson [687] using *truncated differentials* (building on Knudsen [686]) yields a certificational attack on 5-round SAFER K-64 with 2^{45} chosen plaintexts; the attack, which does not extend to 6 rounds, indicates that security is less than argued by Massey [788], who also notes that preliminary attempts at linear cryptanalysis of SAFER were unsuccessful.

RC5 was designed by Rivest [1056], and published along with a reference implementation. The magic constants of Table 7.14 are based on the golden ratio and the base of natural logarithms. The data-dependent rotations (which vary across rounds) distinguish RC5 from iterated ciphers which have identical operations each round; Madryga [779] proposed an earlier (less elegant) cipher involving data-dependent rotations. A preliminary examination by Kaliski and Yin [656] suggested that, while variations remain to be explored, standard linear and differential cryptanalysis appear impractical for RC5–32 (64-bit blocksize) for $r = 12$: their differential attacks on 9 and 12 round RC5 require, respectively, 2^{45}, 2^{62} chosen-plaintext pairs, while their linear attacks on 4, 5, and 6-round RC5–32 require, respectively, 2^{37}, 2^{47}, 2^{57} known plaintexts. Both attacks depend on the number of rounds and the blocksize, but not the byte-length of the input key (since subkeys are recovered directly). Knudsen and Meier [689] subsequently presented differential attacks on RC5 which improved on those of Kaliski and Yin by a factor up to 512, and showed that RC5 has so-called *weak keys* (independent of the key schedule) for which these differential attacks perform even better.

LOKI was introduced by Brown, Pieprzyk, and Seberry [215] and renamed LOKI'89 after the discovery of weaknesses lead to the introduction of LOKI'91 by Brown et al. [214]. Knudsen [682] noted each LOKI'89 key fell into a class of 16 equivalent keys, and the differential cryptanalysis of Biham and Shamir [137] was shown to be effective against reduced-round versions. LOKI'91 failed to succumb to differential analysis by Knudsen [683]; Tokita et al. [1193] later confirmed the optimality of Knudsen's characteristics, suggesting that LOKI'89 and LOKI'91 were resistant to both ordinary linear and differential cryptanalysis. However, neither should be used for hashing as originally proposed (see Knudsen [682]) or in other modes (see Preneel [1003]). Moreover, both are susceptible to *related-key attacks* (Note 7.6), popularized by Biham [128, 129]; but see also the earlier ideas of Knudsen [683]. Distinct from these are *key clustering attacks* (see Diffie and Hellman [347, p.410]), wherein a cryptanalyst first finds a key "close" to the correct key, and then searches a cluster of "nearby" keys to find the correct one.

8×32 bit S-boxes first appeared in the Snefru hash function of Merkle [854]; here such fixed S-boxes created from random numbers were used in its internal encryption mapping. Regarding large S-boxes, see also Gordon and Retkin [517], Adams and Tavares [7], and Biham [132]. Merkle [856] again used 8×32 S-boxes in Khufu and Khafre (see also §15.2.3(viii)). In this 1990 paper, Merkle gives a chosen-plaintext differential attack defeating 8 rounds of Khufu (with secret S-box). Regarding 16-round Khafre, a DC attack by Biham and Shamir [138, 137] requires somewhat over 1500 chosen plaintexts and one hour on a personal computer, and their known-plaintext differential attack requires $2^{37.5}$ plaintexts; for 24-round Khafre, they require 2^{53} chosen plaintexts or $2^{58.5}$ known plaintexts. Khufu with 16 rounds was examined by Gilbert and Chauvaud [456], who gave an attack using 2^{43} chosen plaintexts and about 2^{43} operations.

CAST is a design procedure for a family of DES-like ciphers, featuring fixed $m \times n$ bit S-boxes ($m < n$) based on bent functions. Adams and Tavares [7] examine the construction of large S-boxes resistant to differential cryptanalysis, and give a partial example (with 64-bit blocklength and 8×32 bit S-boxes) of a CAST cipher. CAST ciphers have variable keysize and numbers of rounds. Rijmen and Preneel [1049] presented a cryptanalytic technique applicable to Feistel ciphers with non-surjective round functions (e.g., LOKI'91 and an example CAST cipher), noting cases where 6 to 8 rounds is insufficient.

Blowfish is a 16-round DES-like cipher due to Schneier [1093], with 64-bit blocks and keys of length up to 448 bits. The computationally intensive key expansion phase creates eighteen 32-bit subkeys plus four 8×32 bit S-boxes derived from the input key (cf. Khafre above), for a total of 4168 bytes. See Vaudenay [1216] for a preliminary analysis of Blowfish.

3-WAY is a block cipher with 96-bit blocksize and keysize, due to Daemen [289] and introduced by Daemen, Govaerts, and Vandewalle [290] along with a reference C implementation and test vectors. It was designed for speed in both hardware and software, and to resist differential and linear attacks. Its core is a 3-bit nonlinear S-box and a linear mapping representable as polynomial multiplication in $\mathbb{Z}_2^{12}$.

SHARK is an SP-network block cipher due to Rijmen et al. [1048] (coordinates for a reference implementation are given) which may be viewed as a generalization of SAFER, employing highly nonlinear S-boxes and the idea of MDS codes (cf. Note 12.36) for diffusion to allow a small number of rounds to suffice. The block ciphers BEAR and LION of Anderson and Biham [30] are 3-round unbalanced Feistel networks, motivated by the earlier construction of Luby and Rackoff [776] (see also Maurer [816] and Lucks [777]) which provides a provably secure (under suitable assumptions) block cipher from pseudorandom functions using a 3-round Feistel structure. SHARK, BEAR, and LION all remain to be subjected to independent analysis in order to substantiate their conjectured security levels.

SKIPJACK is a classified block cipher whose specification is maintained by the U.S. National Security Agency (NSA). FIPS 185 [405] notes that its specification is available to organizations entering into a Memorandum of Agreement with the NSA, and includes interface details (e.g., it has an 80-bit secret key). A public report contains results of a preliminary security evaluation of this 64-bit block cipher ("SKIPJACK Review, Interim Report, The SKIPJACK Algorithm", 1993 July 28, by E.F. Brickell, D.E. Denning, S.T. Kent, D.P. Maher, and W. Tuchman). See also Roe [1064, p.312] regarding curious results on the cyclic closure tests on SKIPJACK, which give evidence related to the size of the cipher keyspace.

GOST 28147-89 is a Soviet government encryption algorithm with a 32-round Feistel structure and unspecified S-boxes; see Charnes et al. [241].

RC2 is a block cipher proprietary to RSA Data Security Inc. (as is the stream cipher RC4). WAKE is a block cipher due to Wheeler [1237] employing a key-dependent table, intended for fast encryption of bulk data on processors with 32-bit words. TEA (Tiny Encryption Algorithm) is a block cipher proposed by Wheeler and Needham [1238].