Modern Cryptology: An Introduction

Bart Preneel

January 2018

Contents

Cryptology is the science that studies mathematical techniques that provide secrecy, authenticity and related properties for digital information [1]. Historically cryptography dealt mostly with the protection of communications, but in the computer era the protection of stored data has become equally important. More recently cryptography is also capable of protecting information while it is being processed (e.g., searching on or processing of encrypted data). Cryptology also enables to create trust relationships over open networks; more in general, cryptographic protocols allow mutually distrusting parties to achieving a common goal while protecting their own interests. Cryptology is a fundamental enabler for security, privacy and dependability in an on-line society. Cryptographic techniques can be found at the core of computer and network security, of digital identification and digital signatures, digital content management systems, etc. Their applications vary from e-business, m-business, e-health, e-voting and on-line payment systems to wireless protocols and ambient intelligence.

The science of cryptology is almost as old as writing itself; for an historical perspective, the reader is referred to D. Kahn [2] and S. Sing [3]. Until the beginning of the 20th century, the use of cryptology was restricted to kings, generals and diplomats. This all changed with the advent of wireless communication, both for military and business purposes around the mid 1910s. While the first generations of widely used devices were purely mechanical or electromechanical, electronic devices were introduced in the 1960s. For the next decades, cryptology was still restricted to hardware encryption on telecommunication networks and in computer systems; the main users were still governments and industry, in particular the financial and military sector. Around 1990 general purpose hardware became fast enough to allow for software implementations of cryptology and with the explosion of the Internet, cryptography became quickly a tool used by mass-market software; this includes protocols such as Transport Layer Security (TLS) and Secure Shell (SSH) at the transport layer, IPsec at the network layer [**?**] and more recently the WLAN and WPLAN protocols at the data link layer. Simultaneously the success of the GSM network and the deployment of smart cards in the banking sector brought cryptography in the hands of every citizen. An increasing number of European governments is issuing electronic identity cards. In the next decade cryptographic hardware will be integrated into every general purpose processor, while an ever growing number of small processors (hundreds or thousands of devices per user) will bring cryptology *everywhere*.

Without cryptology, securing our information infrastructure would not be possible. While cryptology as an art is very old, it has developed as a science in the last seventy years; most of the open research dates from the last forty years. A significant number of successes has been obtained, and it is clear that cryptology should no longer be the weakest link in our modern security systems. Nevertheless, as a science and engineering discipline, cryptology is still facing some challenging problems.

This article intends to present the state of the art and to offer a perspective on open research issues in the area of cryptographic algorithms. The chapter will cover the principles underlying the design of cryptographic algorithms and protocols and will be restricted to the protection of communicated and stored data. First it discusses algorithms for confidentiality protection that is, the protection against passive eavesdroppers and for authentication that is, the protection against active eavesdroppers, who try to modify information. The chapter concludes with some comments on research challenges.

1 Encryption for Secrecy Protection

The basic idea in cryptography is to apply a complex mathematical transformation to protect the information. When the sender (usually called Alice) wants to convey a message to a recipient (Bob), the sender will apply to the *plaintext* P the mathematical transformation $E()$. This transformation $E()$ is called the encryption algorithm; the result of this transformation is called the *ciphertext* or

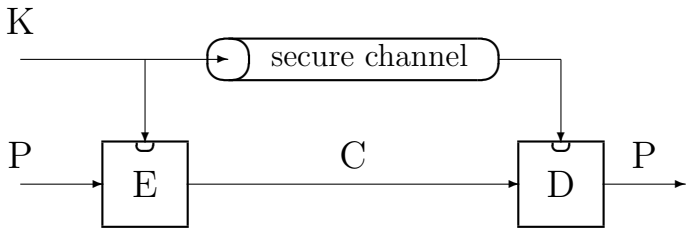


Figure 1: Model for conventional or symmetric encryption

$C = E(P)$. Bob, the recipient, will decrypt C by applying the inverse transformation $D = E^{-1}$, and in this way he recovers P or $P = D(C)$. For a secure algorithm E, the ciphertext C does not make sense to an outsider, such as Eve, who is tapping the connection and who can obtain C, but not obtain any partial information on the corresponding plaintext P.

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

This problem can be solved by introducing into the encryption algorithm $E()$ a secret parameter, the key K. Typically such a key is a binary string of forty to a few thousand bits. A corresponding key K^* is used for the decryption algorithm D. One has thus $C = E_K(P)$ and $P = D_{K^*}(C)$. (See also Fig. 1 which assumes that $K^* = K$.) The transformation strongly depends on the keys: if one uses a wrong key $K^{*\prime} \neq K^*$, then a random plaintext P' rather than the plaintext P is obtained. Now it is possible to publish the encryption algorithm $E()$ and the decryption algorithm $D()$; the security of the system relies only on the secrecy of two short keys, which implies that $E()$ and $D()$ can be evaluated publicly and distributed on a commercial basis. Think of the analogy of a mechanical lock: everyone knows how a lock works, but to open a particular lock, one needs to have a particular key or know the secret combination.[1] In cryptography the assumption that the algorithm should remain secure even if it is known to the opponent is known as "Kerckhoffs's principle". Kerckhoffs was a 19th century Dutch cryptographer who formulated this principle.

A simple example of an encryption algorithm is the Caesar cipher after the Roman emperor who used it. The plaintext is encrypted letter by letter; the ciphertext is obtained by shifting the letters over a fixed number of positions in the alphabet. The secret key indicates the number of positions. It is claimed that Caesar always used the value of three, such that AN EXAMPLE would be encrypted to DQ HADPSOH. Another example is the name of the computer HAL from S. Kubrick's A Space Odyssey (2001), which was obtained by replacing the letters of IBM by their predecessor in the alphabet. This transformation corresponds to a shift over twenty-five positions or minus one position. It is clear that such a system is not secure, since it is easy to try the twenty-six values of the key and to identify the correct plaintext based on the redundancy of the plaintext.

The *simple substitution* cipher replaces a letter by any other letter in the alphabet. For example, the key could be

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
MZNJSOAXFQGYKHLUCTDVWBIPER
```

[1] Note however that Matt Blaze has demonstrated in [5] that many modern locks are easy to attack and that their security relies to a large extent on *security through obscurity* that is, the security of locks relies on the fact that the methods to design and attack locks are not published.

which means that an A is mapped to an M, a B to a Z, and so on; hence, THEEVENING would be encrypted as VXSSBSHFHA. For an alphabet of n letters (in English $n = 26$), there are $n!$ substitutions, which implies that there are $n!$ values for the secret key. Note that even for $n = 26$ trying all keys is not possible since $26! = 403291461126605635584000000 = 4 \cdot 10^{26}$. Even if a fast computer consisting of a million processors that each could try one billion (10^9) keys per second, it would take thousand years to try all the keys. However, it is easy to break this scheme by frequency analysis: in a standard English text, the character E accounts for 12 out of every 100 characters, if spaces are omitted from the plaintext. Hence it is straightforward to deduce that the most common ciphertext character, in this example S corresponds to an E. Consequently, the key space has been reduced by a factor of twenty-six. It is easy to continue this analysis based on lower frequency letters and based on frequent combinations of two (*e.g.,* TH) and three (*e.g.,* THE) letters that are called digrams and trigrams respectively. In spite of the large key length, simple substitution is a very weak cipher, even if the cryptanalyst only has access to the ciphertext. In practice, the cryptanalyst may also know part of the plaintext, for example a standard opening such as DEARSIR.

A second technique applied in cryptology is a *transposition cipher* in which symbols are moved around. For example, the following mapping could be obtained:

```
TRANS          ONI P
POSIT    ⟶    SRTIT
IONS           ASNOO
```

Here the key would indicate where the letters are moved. If letters are grouped in blocks of n (in this example $n = 15$), there are $n!$ different transpositions. Again solving this cipher is rather easy, for example by exploiting digrams and trigrams or by fragments of known plaintexts. In spite of these weaknesses, modern ciphers designed for electronic computers are still based on a combination of several transpositions and substitutions (cf. Sect. 1.1.3).

A large number of improved ciphers has been invented. In the 15th and 16th century, *polyalphabetic substitution* was introduced, which uses $t > 1$ different alphabets. For each letter one of these alphabets is selected based on some simple rule. The complexity of these ciphers was limited by the number of operations that an operator could carry out by hand. None of these manual ciphers is considered to be secure today. With the invention of telegraph and radio communications, more complex ciphers have been developed. The most advanced schemes were based on mechanical or electromechanical systems with rotors, such as the famous Enigma machine used by Germany in World War II and the Hagelin machines. Rotor machines were used between the 1920s and the 1950s. In spite of the increased complexity, most of these schemes were not sufficiently secure in their times One of the weak points was users who did not follow the correct procedures. The analysis of the Lorenz cipher resulted in the development of Colossus, one of the first electronic computers.

A problem not yet been addressed is how Alice and Bob can exchange the secret key. The easy answer is that cryptography does not solve this problem; cryptography only moves the problem and at the same time simplifies the problem. In this case the secrecy of a (large) plaintext has been reduced to that of a *short* key, which can be exchanged beforehand. The problem of exchanging keys is studied in more detail in an area of cryptography that is called *key management*, which will not be discussed in detail in this chapter. See the book by Menezes et al. [1] for an overview of key management techniques.

The branch of science that studies the encryption of information is called *cryptography.* A related branch tries to break encryption algorithms by recovering the plaintext without knowing the key or by deriving the key from the ciphertext and parts of the plaintext; it is called *cryptanalysis.* The term *cryptology* covers both aspects. For more extensive introductions to cryptography, the reader is referred to [7, 1, 8, 9, 10].

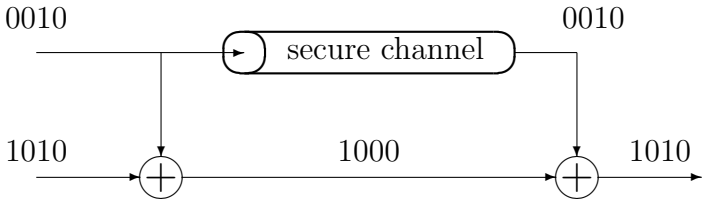


Figure 2: The Vernam scheme or one-time pad

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

1.1 Symmetric Encryption

This section introduces three types of symmetric encryption algorithms: the one-time pad, also known as the Vernam scheme, additive stream ciphers, and block ciphers.

1.1.1 The One-Time Pad or the Vernam Scheme

While the one-time pad has been attributed to G.S. Vernam who demonstrated the system in 1917 [11], recent research by Bellovin [6] showed that F. Miller was the first to propose this scheme in 1871 (during the gold rush in California).

The encryption operation consists of adding a random key bit by bit to the plaintext. The decryption operation subtracts the same key from the ciphertext to recover the plaintext (see Fig. 2). In practice, Vernam stored the keys on paper tapes.

The one-time pad can be formally described as follows. The ith bit of the plaintext, ciphertext, and key stream are denoted with p_i, c_i, and k_i, respectively. The encryption operation can then be written as $c_i = p_i \oplus k_i$. Here $\oplus$ denotes addition modulo 2 or exclusive or. The decryption operation is identical to the encryption or the cipher is an involution. Indeed, $p_i = c_i \oplus k_i = (p_i \oplus k_i) \oplus k_i = p_i \oplus (k_i \oplus k_i) = p_i \oplus 0 = p_i$. Vernam proposed use of a perfectly random key sequence that is, the bit sequence k_i, $i = 1, 2, \ldots$ should consist of a uniformly and identically distributed sequence of bits.

In 1949 C. Shannon, the father of information theory, published his mathematical proof that shows that from observing the ciphertext, the opponent cannot obtain any new information on the plaintext, no matter how much computing power he has [12]. Shannon called this property *perfect secrecy*. The main disadvantage of the one-time pad is that the secret key is exactly as long as the message. C. Shannon also showed that the secret key cannot be shorter if one wants perfect secrecy. Until the late 1980s, the one-time pad was used by diplomats and spies and even for the red-telephone system between Washington and Moscow.

Spies used to carry key pads with random characters: in this case p_i, c_i and k_i are elements from $\mathbb{Z}_{26}$ representing the letters A through Z. The encryption operation is $c_i = (p_i + k_i) \bmod 26$ and the decryption operation is $p_i = (c_i - k_i) \bmod 26$. The keys were written on sheets of paper contained on a pad. The security of the scheme relies on the fact that every page of the pad is used only once, which explains the name one-time pad. Note that during World War II the possession of pads with random characters was sufficient to be convicted as a spy.

The one-time pad was also used for Soviet diplomatic communications. Under the codeword Venona, US cryptologists attempted to break this system in 1943, which seems impossible as it

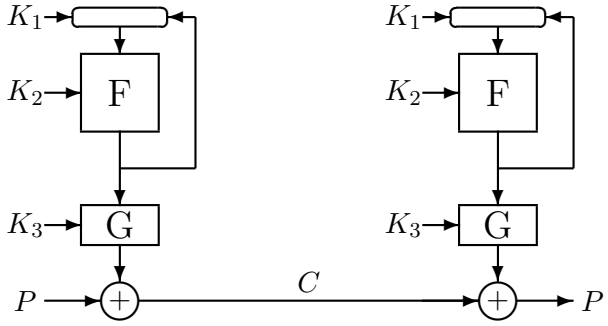


Figure 3: An additive stream cipher consisting of a keyed finite state machine. The initial state depends on K_1, the next state function F depends on K_2 and the output function G depends on K_3. The three keys K_1, K_2, and K_3 are derived from the user key K; this operation is not shown

offers perfect secrecy. However, after two years, it was discovered that the Soviets used their pads twice that is, two messages were encrypted using the same key stream. This error was due to time pressure in the production of the pads. If c and c' are ciphertexts generated with the same pad, one finds that

$$c \oplus c' = (p \oplus k) \oplus (p' \oplus k) = (p \oplus p') \oplus (k \oplus k) = p \oplus p' .$$

This analysis implies that one can deduce from the ciphertext the sum of the corresponding plaintexts. Note that if the plaintexts are written in natural language, their sum $p \oplus p'$ is not uniformly distributed, so it is possible to detect that the correct c and c' have been matched. Indeed, if c and c' would have been encrypted with different keys k and k', the sum $c \oplus c'$ would be equal to $(p \oplus p') \oplus (k \oplus k')$ which is uniformly distributed. By guessing or predicting parts of plaintexts, a clever cryptanalyst can derive most of p and p' from the sum $p \oplus p'$. In practice, the problem was more complex since the plaintexts were encoded with a secret method before encryption and cover names were used to denote individuals. Between 1943 and 1980, approximately three thousand decryptions out of twenty-five thousand messages were obtained. Some of these plaintexts contained highly sensitive information on Soviet spies. The Venona successes were only made public in 1995 and teach an important lesson: in using a cryptosystem errors can be fatal even if the cryptosystem itself is perfectly secure. More details on Venona can be found in the book by Haynes [13]. Section 3.3 discusses another weakness that can occur when implementing the one-time pad.

1.1.2 Additive Stream Ciphers

Additive stream ciphers are ciphers for which the encryption consists of a modulo 2 addition of a key stream to the plaintext. These ciphers try to mimic the one-time pad by replacing the perfectly random key stream by a pseudo-random key stream which is generated from a short key. Here pseudo-random means that the key stream looks random to an observer who has limited computing power. In practice one generates the bit sequence k_i with a keyed finite state machine (see Fig. 3). Such a machine stretches a short secret key K into a much longer key stream sequence k_i. The sequence k_i is eventually periodic. One important but insufficient design criterion for the finite state machine is that the period has to be large (2^{90} is a typical lower bound) because a repeating key stream leads to a very weak scheme; this is known as transmission in depth (cf. the Venona project). The values k_i should have a distribution that is close to uniform; another condition is that there should be no correlations between output strings. Note that cryptanalytic attacks may exploit correlations of less than 1 in 10 million.

Formally, the algorithm generating the sequence k_i can be parameterized with a security parameter. For the security of the stream cipher one requires that the sequence satisfies every polynomial time statistical test for randomness. In this definition polynomial time means that the complexity

of these tests can be described as a polynomial function of the security parameter. Another desirable property is that no polynomial time machine can predict the next bit of the sequence based on the previous outputs with a probability that is significantly better than one-half. An important and perhaps surprising result in theoretical cryptology by A. Yao shows that these two conditions are in fact equivalent [14].

Stream ciphers have been popular in the 20th century: they operate on the plaintext character by character and require no memory for the plaintext or ciphertext, which is convenient and allows for a simple and thus inexpensive implementation. Most of the rotor machines are additive stream ciphers. Between 1960 and 1990, stream ciphers based on Linear Feedback Shift Registers (LFSRs) were very popular. (See for example the book by Rueppel [15].) However, most of these algorithms were trade secrets; every organization used its own cipher, and no standards were published. The most widely used LFSR-based stream ciphers are A5/1 and A5/2, which are implemented in hardware in the GSM phones. The GSM algorithms were kept secret, but they leaked out and were shown to be rather weak [16, 17]. A more secure algorithm (the block cipher KASUMI) was developed around 2000, but its deployment is still not completed. While GSM (2G) has been superseded by 3G and 4G, which are much more secure and much faster, it can still be used as a fall-back network; attackers can trigger fall-back to 2G by jamming 3G and 4G networks. Several sophisticated attacks against LFSR-based stream ciphers are known; see for example [18, 19].

RC4, designed in 1987 by Ron Rivest, is based on completely different principles. RC4 is designed for eight-bit microprocessors and was initially kept as a trade secret. It was posted on the internet in 1994 and was widely used in browsers (TLS protocol). Serious statistical weaknesses have been identified in RC4 [20, 21] that make the algorithm unsuitable for most applications; however, the algorithm still seems to resist attacks that recover the key.

In the 1990s, a large number of very fast stream ciphers has been proposed that are software oriented, suited for 32-bit processors, and that intend to offer a high level of security. However, none of these seemed suitable as standard ciphers. Stream ciphers can be very valuable for encryption with very few hardware gates or for high speed encryption. In 3G the stream cipher SNOW-3G was added an in 4G China proposed in addition the ZUC algorithm. The eSTREAM competition (2004–2008) run by the Network of Excellence ECRYPT has resulted in a portfolio that includes several promising algorithms. For hardware we mention Grain-128 [22] (and evolution of Grain) and Trivium [23] and for software Bernstein's ChaCha20 [24] (an evolution of Salsa20) and Wu's HC-128 [25]. Improving our understanding of these stream ciphers is clearly an important research topic for the years ahead.

1.1.3 Block Ciphers

Block ciphers take a different approach to encryption: the plaintext is divided into larger words of n bits, called *blocks*; typical values for n are 64 and 128. Every block is enciphered in the same way, using a keyed one-way permutation that is, a permutation on the set of n-bit strings controlled by a secret key. The simplest way to encrypt a plaintext using a block cipher is as follows: divide the plaintext into n-bit blocks P_i, and encrypt these block by block. The decryption also operates on individual blocks:

$$C_i = E_K(P_i) \qquad \text{and} \qquad P_i = D_K(C_i)\,.$$

This way of using a block cipher is called the Electronic CodeBook (ECB) mode. Note that the encryption operation does *not* depend on the location in the ciphertext as is the case for additive stream ciphers.

Consider the following attack on a block cipher, the so-called tabulation attack: the cryptanalyst collects ciphertext blocks and their corresponding plaintext blocks which is possible as part of the plaintext is often predictable; these blocks are used to build a large table. With such a table, one can deduce information on other plaintexts encrypted under the same key. In order to

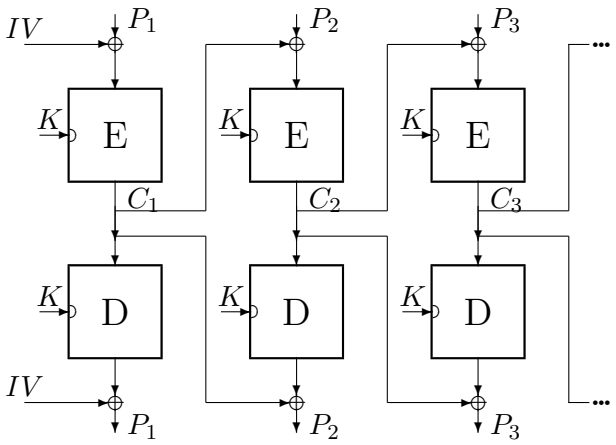


Figure 4: The CBC mode of a block cipher

preclude this attack, the value of n has to be quite large (e.g., 64 or 128). Moreover, the plaintext should not contain any repetitions or other patterns, as these will be leaked to the ciphertext.

This last problem shows that even if n is large, the ECB mode is not suited to encrypt structured plaintexts, such as text and images. This mode should only be used in exceptional cases where the plaintext is random, such for as the encryption of cryptographic keys.[2] There is however an easy way to randomize the plaintext by using the block cipher in a different way. The most popular mode of operation for a block cipher is the Cipher Block Chaining (CBC) mode (see Fig. 4). In this mode the different blocks are coupled by adding modulo 2 to a plaintext block the previous ciphertext block:

$$C_i = E_K(P_i \oplus C_{i-1}) \qquad \text{and} \qquad P_i = D_K(C_i) \oplus C_{i-1}\,.$$

Note that this randomizes the plaintext and hides patterns. To enable the encryption of the first plaintext block ($i = 1$), one defines C_0 as the initial value IV, which should be randomly chosen and transmitted securely to the recipient. By varying this value, one can ensure that the same plaintext is encrypted into a different ciphertext under the same key. The CBC mode allows for random access on decryption: if necessary, one can decrypt only a small part of the ciphertext. A disadvantage of the CBC mode is that the encryption algorithm is serial – one can only use parallelism if one encrypts multiple independent strings simultaneously. A security proof the CBC mode has been provided by Bellare *et al.* [26]; it holds as long as the opponent can only obtain ciphertext corresponding to chosen plaintexts and if the IV is unpredictable and random. Unfortunately this latter condition is often violated in practical implementations. Note that the CBC mode is insecure if the opponent can choose ciphertexts and obtain the corresponding plaintexts. This is not only a theoretical observation but has resulted in the cryptanalysis of several widely deployed protocols such as SSH and TLS [27]. If the length of the plaintext is not a multiple of n bits, the last block has to be padded with an additional string; one of the main problems is that it is very tricky to write constant time verification code for the padding string; even small timing variations leak information on the plaintexts.

One can also use a block cipher to generate a key stream that can be used in an additive stream cipher: in the Output FeedBack (OFB) mode the block cipher is applied iteratively by feeding back the n-bit output to the input; in the CounTeR (CTR) mode one encrypts successive values of a counter. The main advantage of the CTR mode is that it allows for fast implementations using parallelism and pipelining. The main risk of the CTR mode is that one (accidentally) reuses a counter value, for example after a crash, resulting to leakage of plaintext. An alternative stream cipher mode is the Cipher FeedBack (CFB) mode; this mode is slower but it has better

[2] And even then authenticated encryption is strongly recommended.

synchronization properties; as most communication channels are synchronous today, it is rarely used. The modes of operation have been standardized in FIPS 81 [28] (see also [29] which adds the CTR mode) and ISO/IEC 10116 [30].

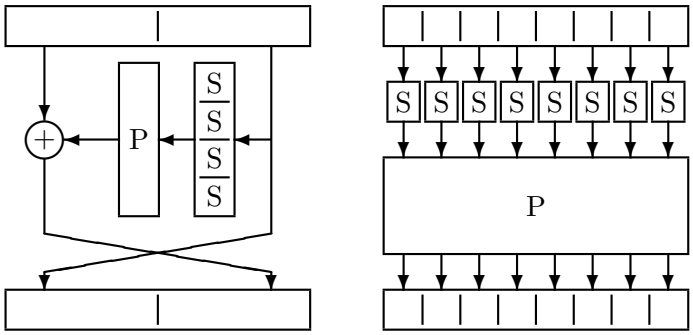
Block ciphers form a very flexible component. They have played an important role in the past 40 years because of the publication of two US government Federal Information Processing Standards (FIPS) for the protection of sensitive but unclassified government information. An important aspect of these standards is that they can be used without paying a license fee.

The first standardized block cipher is the Data Encryption Standard (or DES) of FIPS 46 [31] published in 1977. This block cipher was developed by IBM together with National Security Agency (NSA) in response to a call by the US government. DES represents a remarkable effort to provide a standard for government and commercial use; its impact on both practice and research can hardly be overestimated. DES was widely used in the financial industry. DES has a block size of 64 bits and a key length of 56 bits. (More precisely, the key length is 64 bits, but eight of these are parity bits.) The 56-bit key length was a compromise: it would allow the US government to find the key by brute force that is, by trying all $2^{56} \approx 7 \cdot 10^{16}$ keys one by one but would put a key search beyond limits for an average opponent. However, as hardware got faster, this key length was sufficient only for ten to fifteen years; hence, DES reached the end of its lifetime in 1987-1992 (see Sect. 3.3 for details). The block length of 64 bits is no longer adequate either because there exist matching ciphertext attacks on the modes of operation of an n-bit block cipher that require about $2^{n/2}$ ciphertext blocks [32]. For $n = 64$, these attack require four billion ciphertexts and with a high speed encryption device this number is reached in less than a minute. The DES design was oriented towards mid-1970s hardware. For example, it uses a sixteen-round Feistel structure (see Fig. 5) which implies that the hardware for encryption and decryption is identical. Each round consists of non-linear substitutions from six bits to four bits followed by some bit permutations or transpositions. The performance of DES in software is suboptimal; for example DES runs at 40 cycles/byte on a Pentium III, which corresponds to 200 Mbit/s for a clock frequency of 1 GHz.

In 1978, one year after the publication of the DES standard, an improved variant of DES was proposed: triple-DES consists of three iterations of DES: $E_{K_1}(D_{K_2}(E_{K_3}(x)))$. Only in 1999 this variant has been included into the third revision of FIPS 46 [31]. The choice of a decryption for the middle operation is motivated by backward compatibility: indeed, choosing $K_1 = K_2 = K_3$ results in single DES. Three-key triple DES has a 168-bit key, but the security level corresponds to a key of approximately 100 bits. Initially two-key triple DES was proposed, with $K_3 = K_1$: its security level is about 80 to 90 bits.[3] On first examination, the double-DES key length of 112 bits appears sufficient; however, it has been shown that the security level of double-DES is approximately 70 bits. For an overview of these attacks see [1]. The migration from DES to triple-DES in the financial sector was started in 1986, but it is progressing slowly and has taken 20 years to complete. Triple-DES has the disadvantage that it is rather slow (115 cycles/byte on a Pentium III) and that the block length is still limited to 64 bits.

In 1997 the US government decided to replace DES by the Advanced Encryption Standard (AES). AES is a block cipher with a 128-bit block length and key lengths of 128, 192, and 256 bits. An open call for algorithms was issued; fifteen candidates were submitted by the deadline of June 1998. After the first round, five finalists remained and in October 2000 it was announced that the Rijndael algorithm, designed by the Belgian cryptographers Vincent Rijmen and Joan Daemen, was the winner. The FIPS standard was published in November 2001 [33]. It may not be a coincidence that the U.S. Department of Commerce Bureau of Export Administration (BXA) relaxed export restrictions for US companies in September 2000. Note that otherwise it would have been illegal to export AES software from the US to Belgium. In 2003, the US government announced that it would

[3] If 2^t known plaintexts are available, the security level of two-key triple DES against key recovery is $120 - t$ bits; in financial applications $t \approx 6 \ldots 10$, which explains why the banks plan to keep using two-key triple DES for the next decade.



Feistel cipher SP network

Figure 5: One round of a Feistel cipher (left) and of a Substitution-Permutation (SP) network. Here S represents a substitution and P a permutation, which can be a bit permutation or an affine mapping. In a Feistel cipher a complex operation on the right part is added to the left part; it has the advantage that the decryption operation is equal to the encryption operation, which simplifies hardware implementations. The substitution operation does not need to be invertible here. In an SP network all the bits are updated in every round, which guarantees faster diffusion of information; the substitution and permutation operations need to be invertible

also allow the use of AES for secret data, and even for top secret data; the latter application requires key lengths of 192 or 256 bits. Rijndael is a rather elegant and mathematical design. Among the five finalists it offered the best combination of security, performance, efficiency, implementability, and flexibility. AES allows for efficient implementations on 32-bit architectures (at the time of selection AES-128 ran at 15 cycles/byte on a Pentium III but later implementations were developed that were twice as fast), but also for compact implementations on 8-bit smart cards. Moreover, hardware implementations of AES offer good trade-offs between size and speed. In 2010 Intel has decided to add the AES-NI instructions to its processors; on the latest Kaby Lake processors AES-128 now runs at 0.66 cycles/byte, which means that encryption is now blazingly fast. AMD has followed Intel's example and a growing number of embedded processors are adding instructions for AES.

AES consists of a Substitution-Permutation (SP) network with ten rounds for a 128-bit key, twelve rounds for a 192-bit key and fourteen rounds for a 256-bit key (see Fig. 5). Each round consists of non-linear substitutions (from eight bits to eight bits) followed by some affine transformations, which move around the information. Note that Rijndael also supports 192-bit and 256-bit block lengths, but these have not been included in the AES standard. For a complete description of the AES and its design, see [34].

There exist many other block ciphers; a limited number of these has been included in products, such as Camellia, members of the CAST family, FEAL, Gost, IDEA, KASUMI (used in mobile networks), Present and Skipjack. For more details the reader is referred to the cryptographic literature.

1.2 Public-Key Encryption

The main problem left unsolved by symmetric cryptography is the key distribution problem. Especially in a large network it is not feasible to distribute keys between all user pairs. In a network with t users, there are $t(t-1)/2$ such pairs; hence even for one thousand users approximately half a million keys are needed. An alternative is to manage all keys in a central entity that shares a secret key with every user. However, this entity then becomes a single point of failure and an attractive target of attack. A much more elegant solution to the key management problem is offered by public-key cryptography invented in 1976, independently by W. Diffie and M. Hellman [35] and by R. Merkle [36].

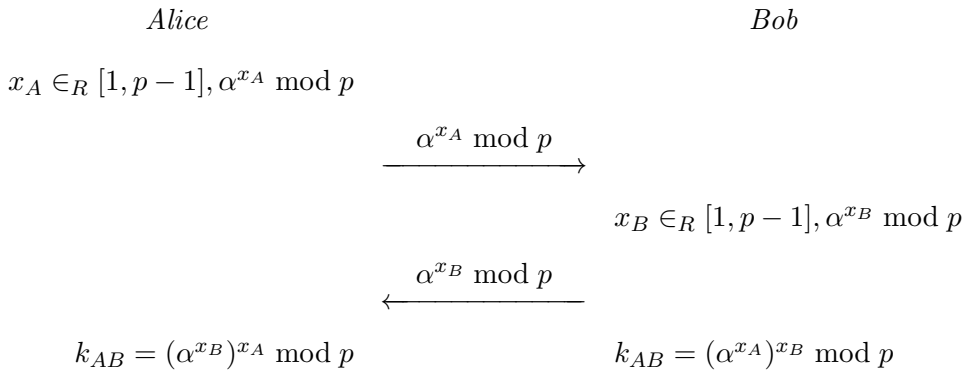


Figure 6: The Diffie-Hellman protocol

1.2.1 Public-Key Agreement

A public-key agreement protocol allows two parties who have never met to agree on a secret key by way of a public conversation. Diffie and Hellman showed how to achieve this goal using the concept of *commutative one-way functions*. A *one-way function* is a function that is easy to compute but hard to invert. For example, in a block cipher, the ciphertext has to be a one-way function of the plaintext and the key: it is easy to compute the ciphertext from the plaintext and the key, but given the plaintext and the ciphertext it should be hard to recover the key otherwise the block cipher would not be secure. Similarly it can be shown that the existence of pseudo-random string generators, as used in additive stream ciphers, implies the existence of one-way functions. A *commutative one-way function* is a one-way function for which the result is the same independent of the order of the evaluation: for a function $f(\cdot, \cdot)$ with two arguments $f(f(z, x), y) = f(f(z, y), x)$.

The candidate commutative one-way function proposed by Diffie and Hellman is $f(\alpha, x) = \alpha^x \bmod p$; here p is a large prime number (large means 2048 bits or more), $x \in [1, p-1]$, and α is a generator $\bmod p$, which means that $\alpha^0, \alpha, \alpha^2, \alpha^3, \ldots, \alpha^{p-2} \bmod p$ run through all values between 1 and $p-1$. For technical reasons, it is required that p is a safe prime, which means that $(p-1)/2$ is a prime number as well. The Diffie-Hellman protocol works as follows (see also Fig. 6).

- Alice and Bob agree on a prime number p and a generator $\alpha \bmod p$.
- Alice picks a value x_A uniformly at random in the interval $[1, p-1]$, computes $y_A = \alpha^{x_A} \bmod p$ and sends this to Bob;
- Bob picks a value x_B uniformly at random in the interval $[1, p-1]$, computes $y_B = \alpha^{x_B} \bmod p$ and sends this to Alice;
- On receipt of y_B, Alice checks that $1 \leq y_B \leq p-2$ and computes $k_{AB} = y_B^{x_A} \bmod p = \alpha^{x_A x_B} \bmod p$.
- On receipt of y_A, Bob checks that $1 \leq y_A \leq p-2$ and computes $k_{BA} = y_A^{x_B} \bmod p = \alpha^{x_B x_A} \bmod p$.
- Alice and Bob compute the secret key as $h(k_{AB}) = h(k_{BA})$, with $h()$ a hash function or MDC (see Sect. 2.1).

It is easy to see that the commutativity implies that $k_{AB} = k_{BA}$, hence Alice and Bob obtain a common value. Eve, who is eavesdropping the communication, only observes $y_A = \alpha^{x_A} \bmod p$ and $y_B = \alpha^{x_B} \bmod p$; there is no obvious way for her to obtain $k_{AB} = (\alpha^{x_B})^{x_A} \bmod p$. If Eve could compute discrete logarithms that is, derive x_A from y_A and/or x_B from y_B, she could of course

also derive k_{AB}. However, if p is large, this problem is believed to be difficult (cf. infra). Eve could try to find another way to compute k_{AB} from y_A and y_B. So far, no efficient algorithm has been found to solve this problem which is stated as the Diffie-Hellman assumption: it is hard to solve the Diffie-Hellman problem that is, to deduce $(\alpha^{x_B})^{x_A} \bmod p$ from $\alpha^{x_A} \bmod p$ and $\alpha^{x_B} \bmod p$. If the Diffie-Hellman assumption holds, the Diffie-Hellman protocol results in a common secret between Alice and Bob after a public conversation. It is clear from the above discussion that the Diffie-Hellman problem cannot be harder than the discrete logarithm problem. It is known that for a very large class of prime numbers the two problems are equivalent.

It is very important to check that $y_A, y_B \notin \{0, 1, p-1\}$: if not, Eve could modify y_A and y_B to one of these values and ensure in this way that $k_{AB} \in \{0, 1, p-1\}$. However, the Diffie-Hellman protocol has another problem: how does Alice know that she is talking to Bob or vice versa? In the famous person-in-the-middle-attack, Eve sets up a conversation with Alice which results in the key k_{AE} and with Bob which results in the key k_{BE}. Eve now shares a key with both Alice and Bob; she can decrypt all messages received from Alice, read them, and re-encrypt them for Bob and vice-versa. Alice or Bob are unable to detect this attack; they believe that they share a common secret only known to the two of them. This attack shows that the common secret can only be established between two parties if there is an authentic channel that is, a channel on which the information can be linked to the sender and the information cannot be modified. The conclusion is that the authenticity of the values y_A and y_B has to be established, by linking them to Alice and Bob respectively. One way to achieve this goal is to read these values or hash values of these values (see Sect. 2.1) over the phone; this solution works if Alice and Bob know each other's voices or if they trust the phone system to connect them to the right person.

The Diffie-Hellman problem has another limitation: Alice and Bob can only agree on a secret key, but Alice cannot use this protocol directly to tell Bob to meet her tonight at 9 pm. Alice can of course use the common key k_{AB} to encrypt this message using the AES algorithm in the CTR-mode. We will explain in the next section how public-key encryption can overcome this limitation.

1.2.2 Public-Key Encryption

The key idea behind public-key encryption is the concept of *trapdoor one-way functions* [35]. Trapdoor one-way functions are one-way functions with an additional property: given some extra information, the trapdoor, it becomes possible to invert the one-way function.

With such functions Bob can send a secret message to Alice without the need for prior arrangement of a secret key. Alice chooses a trapdoor one-way function with public parameter P_A, that is Alice's public key and with secret parameter S_A that is, Alice's secret key. Alice makes her public key widely available. For example, she can put it on her home page, but it can also be included in special directories. Anyone who wants to send some confidential information to Alice computes the ciphertext as the image of the plaintext under the trapdoor one-way function using the parameter P_A. Upon receipt of this ciphertext, Alice recovers the plaintext by using her trapdoor information S_A (see Fig. 7). An attacker, who does not know S_A, sees only the image of the plaintext under a one-way function and will not be able to recover the plaintext. The conditions which a public-key encryption algorithm has to satisfy are:

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

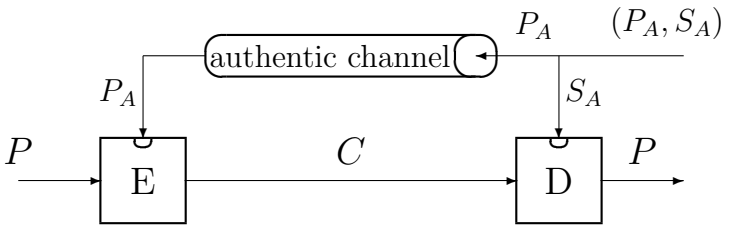


Figure 7: Model for public-key or asymmetric encryption

Note that if a person wants to send a message to Alice, that individual has to know Alice's public key P_A and has to be sure that this key really belongs to Alice and not to Eve, since only the owner of the corresponding secret key will be able to decrypt the ciphertext. Public keys do not need a secure channel for their distribution, but they do need an authentic channel. As the keys for encryption and decryption are different, and Alice and Bob have different information, public-key algorithms are also known as *asymmetric algorithms.*

Designing a secure public-key encryption algorithm is apparently a difficult problem. From the large number of proposals, only a few have survived. The most popular algorithm is the RSA algorithm [37], which was named after its inventors R.L. Rivest, A. Shamir, and L. Adleman. RSA was published in 1978; the patent on RSA has expired in 2000. The security of RSA is based on the fact that it is relatively easy to find two large prime numbers and to multiply these while factoring their product is not feasible with the current algorithms and computers. The RSA algorithm can be described as follows:

key generation: Find two prime numbers p and q with at least one 1024 bits and compute their product, the modulus $n = p \cdot q$. Compute the Carmichael function $\lambda(n)$, which is defined as the least common multiple of $p-1$ and $q-1$. In other words, $\lambda(n)$ is the smallest integer which is a multiple of both $p-1$ and $q-1$. Choose an encryption exponent e, which is at least 32 to 64 bits long and which is relatively prime to $\lambda(n)$ that is, has no common divisors with $\lambda(n)$. Compute the decryption exponent as $d = e^{-1} \bmod \lambda(n)$ using Euclid's algorithm. The public key consists of the pair (e, n), and the secret key consists of the decryption exponent d or the pair (p, q);

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

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

The prime factors p and q or the secret decryption exponent d are the trapdoor that allows the inversion of the function $f(x) = x^e \bmod n$. Indeed, it can be shown that $f(x)^d \bmod n = x^{ed} \bmod n = x$. Note that the RSA function is the dual of the Diffie-Hellman function $f(x) = \alpha^x \bmod p$, which has a fixed base and a variable exponent.

Without explaining the mathematical background of the algorithm, it can be seen that the security of the RSA algorithm depends on the factoring problem. Indeed, if an attacker can factor n, he can find $\lambda(n)$, derive d from e and decrypt any message. However, to decrypt a ciphertext it is sufficient to extract modular eth roots. Note that it is not known whether it is possible to extract eth roots without knowing p and q. The RSA problem is the extraction of random modular eth roots since this corresponds to the decryption of arbitrary ciphertexts. Cryptographers believe that the RSA problem is hard; this assumption is known as the RSA assumption. It is easy to see that the RSA problem cannot be harder than factoring the modulus. Some indication has been found that the two problems may not be equivalent.

For special arguments, the RSA problem is easy. For example -1, 0 and 1 are always fixed points for the RSA encryption function and for small arguments, $P^e < n$ and extracting modular eth root simplifies to extracting natural eth roots, which is an easy problem. However, the RSA assumption

states that extracting *random* modular eth roots is hard, which means that the challenge ciphertext needs to be uniformly distributed. Such a uniform distribution can be achieved by transforming the plaintext with a randomizing transform. A large number of such transforms is known and many of these are *ad hoc* so there is no reason to believe that they should be effective. In 1993, Bellare and Rogaway published a new transform under the name Optimal Asymmetric Encryption (OAEP) together with a security proof [38]. This proof shows that an algorithm that can decrypt a challenge ciphertext without knowing the secret key, can be transformed into an algorithm that computes a random modular eth root. The proof is in the random oracle model, which means that the hash functions used in the OAEP construction are assumed to be perfectly random. However, seven years later Shoup pointed out that the proof was wrong [39]; the error has been corrected for by Fujisaki *et al.* in [40], but the resulting reduction is not meaningful that is, the coupling between the two problems is not very tight in this new proof (except when e is small). Currently the cryptographic community believes that the best way of using RSA is the RSA-KEM mode [41], which is a *hybrid* mode in which RSA is only used to transfer a session key while the plaintext is encrypted using a symmetric algorithm with this key. It is interesting to note that it has taken more than 20 years before cryptographers have understood how RSA should be used properly for encryption.

1.2.3 The Factoring and the Discrete Logarithm Problem

The more complex properties of public-key cryptography seem to require some 'high level' mathematical structure; most public-key algorithms are based on problems from algebraic number theory. While these number theoretic problems are believed to be difficult, it should be noted that there is no mathematical proof that shows that these problems are hard. Moreover, since the invention of public-key cryptography, significant progress has been made in solving concrete instances. This evolution is due to a combination of more sophisticated algorithms with progress in hardware and parallel processing. Table 1 summarizes the progress made in factoring over the past 50 years. It is believed that the discrete logarithm problem mod$p$ is about as difficult as the factoring problem for the same size of modulus. This equivalence only holds if p satisfies certain conditions; a sufficient condition is that p is a safe prime as defined above.

Table 1: Progress of factorization records for products of two random prime numbers. One MIPS year (MY) is the equivalent of a computation during one full year at a sustained speed of one Million Instructions Per Second, which corresponds roughly to the speed of a VAX 11/780

year	# digits	# bits	computation
1964	20	66	
1974	45	150	0.001 MY
1983	50	166	
1984	71	236	0.1 MY
1991	100	332	7 MY
1992	110	365	75 MY
1993	120	399	835 MY
1994	129	429	5000 MY
1996	130	432	1000 MY
1999	140	465	2000 MY
1999	155	512	8400 MY
2003	174	576	
2005	200	663	
2009	232	768	2000 AMD Opteron Core 2.2 GHz Years

The best known algorithm to factor an RSA modulus N is the General Number Field Sieve. It

has been used in all factoring records since 1996 and has a heuristic asymptotic complexity

$$O\left(\exp\left[(1.923+o(1))\cdot(\ln N)^{1/3}\cdot(\ln\ln N)^{2/3}\right]\right).$$

Note that this asymptotic expression should be used with care; extrapolations can only be made in a relatively small range due to the $o(1)$ term. Lenstra and Verheul provide an interesting study on the selection of RSA key sizes [42]; see also the most recent ECRYPT report [47]. Currently it is believed that factoring a 1024-bit RSA modulus (308 digit) modulus requires 2^{74} steps. With special hardware proposed by Shamir and Tromer [43], the following cost estimates have been provided: with an investment of US$ ten million, a 1024-bit modulus can be factored in one year (the initial R&D cost is US$ twenty million). A 768-bit modulus can be factored for US$ five thousand in ninety-five days, and a 512-bit modulus can be factored with a US$ ten thousand device in ten minutes. Note that these cost estimates do not include the linear algebra step at the end; while this step takes additional time and effort, it should not pose any unsurmountable problems. Nevertheless, these estimates show that for 5 to 10 year security, an RSA modulus of 3072 bits is recommended.

1.2.4 Basing Public-Key Cryptology on Other Problems

There has been a large number of proposals for other public key encryption algorithms. Many of these have been broken, the most notable example being the class of knapsack systems. The most important alternative to RSA is the ElGamal scheme, which extends the Diffie-Hellman scheme to public-key encryption. In particular, the group of integers mod$p$ can also be replaced by a group defined by an elliptic curve over a finite field, as proposed by Miller and Koblitz in the mid-eighties. Elliptic curve cryptosystems allow for shorter key sizes that is, a 1024-bit RSA key corresponds to a 148-bit elliptic curve key, but the operations on an elliptic curve are more complex (see [44, 45, 46, 47]). Hyperelliptic curves form an extension of elliptic curves; they have a comparable performance and security if one selects a genus equal to 2.

Other alternatives are schemes based on multivariate polynomials over finite fields, coding theory (McEliece), lattice-based systems such as NTRU and LWE, and isogenies between elliptic curves. The key advantages of these schemes is that they are believed to be secure against quantum computers (cf. Sect. 3.3.3). For this reason NIST has started in 2017 an open competition for so-called post-quantum public key algorithms; this competition is expected to be completed around 2024. Currently these schemes are not yet mature enough for deployment. It is a little worrying that our digital economy relies to a large extent on the claimed difficulty of a few problems in algebraic number theory.

1.2.5 Applying Public-Key Cryptology

The main advantage of public-key algorithms is the simplified key management; deployment of cryptology on the internet largely relies on public-key mechanisms (*e.g.,* TLS, IPsec and SSH [**?**]). An important question is how authentic copies of the public keys can be distributed; this problem will be briefly discussed in Sect. 2.2. The main disadvantages are the larger keys (typically 64 to 512 bytes) and the slow performance: both in software and hardware public-key encryption algorithms are two to three orders of magnitude slower than symmetric algorithms. For example, a 3072-bit exponentiation with a 16-bit exponent (RSA encryption) takes 162,000 cycles or 422 cycles/byte on a 2017 Intel Xeon Kaby Lake with four 3 GHz cores; a decryption with a 3072-bit exponent takes 8.5 million cycles or 22 000 cycles/byte. This speed should be compared to 0.66 to 7.5 cycles/byte for AES. Because of the large difference in performance, the large block length which influences error propagation and the security reasons indicated above, one always employs *hybrid* systems: the public-key encryption scheme is used to establish a secret key, which is then used in a fast symmetric algorithm.

2 Hashing and Signatures for Authentication

In digital communications, the implicit authentication created by recognition of the handwriting, signature, or voice disappears. Electronic information becomes much more vulnerable to falsification as the physical coupling between information and its bearer is lost. Information authentication includes two main aspects:

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

Other aspects which can be important are the timeliness of the information, the sequence of messages, and the destination of information. These aspects can be accounted for by using sequence numbers and time stamps in the messages and by including addressing information in the data. One may also want to deal with streams or with fragmentation.

Until the mid 1980s, it was widely believed that encryption with a symmetric algorithm of a plaintext was sufficient for protecting its authenticity. If a certain ciphertext resulted after decryption in a *meaningful* plaintext, it had to be created by someone who knew the key, and therefore it must be authentic. However, a few counterexamples are sufficient to refute this claim. If a block cipher is used in ECB mode, an attacker can easily reorder the blocks. For any additive stream cipher, including the one-time pad, an opponent can always modify a plaintext bit even without knowing whether a zero has been changed to a one or vice versa. The concept of meaningful information implicitly assumes that the information contains redundancy, which allows a distinction of genuine information from arbitrary plaintext. However, one can envisage applications where the plaintext contains very little or no redundancy, for example, the encryption of keys. The separation between secrecy and authentication has also been clarified by public-key cryptography: anyone who knows Alice's public key can send her a confidential message, and therefore Alice has no idea who has actually sent this message.

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

2.1 Symmetric Authentication

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

- either the hash result is communicated over an authentic channel (*e.g.,* it can be read over the phone or protected with a digital signature, cf. Sect. 2.2). It is then sufficient to use a hash function without a secret parameter, which is also known as a Manipulation Detection Code or MDC;
- or the hash function uses a secret parameter (the key) and is then called a Message Authentication Code or MAC algorithm.

2.1.1 MDCs

If an additional authentic channel is available, MDCs can provide authenticity without requiring secret keys. Moreover an MDC is a flexible primitive, which can be used for a variety of other cryptographic applications. An MDC has to satisfy the following conditions:

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

An MDC satisfying these three conditions is called a *collision resistant* hash function.

For a strong hash function with an n-bit result, solving the first two problems requires about 2^n evaluations of the hash function. This implies that $n = 90 \ldots 100$ is sufficient (cf. Sect. 3.3); larger values of n are required if one can attack multiple targets in parallel. However, finding collisions is substantially easier that finding preimages or second preimages. With high probability a set of hash results corresponding to $2^{n/2}$ inputs contains a collision, which implies that collision resistant hash functions need a hash result of 224 to 256 bits. This last property is also known as the *birthday paradox* based on the following observation: within a group of twenty-three persons the probability that there are two persons with the same birthday is about 50 percent. The reason is that a group of this size contains $23 \cdot 22/2 = 253$ different pairs of persons, which is rather larger compared to the 365 days in a year. The birthday paradox plays an essential role in the security of many cryptographic primitives. Note that not all applications need collision resistant hash functions; sometimes preimage resistance or second preimage resistance is sufficient.

The most efficient hash functions are dedicated hash function designs. The hash functions MD4 and MD5 with a 128-bit hash result are no longer recommended. For the devastating attacks by Wang *et al.*, see [48]). The most popular hash function today is SHA-1, but by building on an attack in 2005 published by Wang et al. *et al.* [49], collisions have been found in 2017 by Stevens et al. [50]; the complexity is about 2^{63} which is much less than 2^{80} for a birthday attack. A short term alternative is RIPEMD-160. Recent additions to the SHA-family include the SHA-2 family (SHA-224, SHA-256, SHA-384, SHA-512, SHA512/224, and SHA-512/256 see FIPS 180-4 [51] for the latest version of this standard). After an open competition between 2008 and 2012, NIST has standardized the SHA-3 family [52] based on the Keccak design of Bertoni et al. [53]. In addition to versions compatible in output with the SHA-2 family, the SHA-3 family also proposes XOFs, or eXtendible Output Functions; these functions do not produce a fixed output length. The ISO standard on dedicated hash functions (ISO/IEC 10118-3) contains RIPEMD-128, RIPEMD-160, SHA-1, SHA-2 and Whirlpool [54]. Part 2 of this standard specifies hash functions based on a block cipher while Part 4 specifies hash functions based on modular arithmetic. SHA-3 will be added in the near future.

2.1.2 MAC Algorithms

MAC algorithms have been used since the 1970s for electronic transactions in the banking environment. They require the establishment of a secret key between the communicating parties. The MAC value corresponding to a message is a complex function of every bit of the message and every bit of the key; it should be infeasible to derive the key from observing a number of text/MAC pairs or to compute or predict a MAC without knowing the secret key. In practice most MAC algorithms are Pseudo-Random Function (PRF) families; this implies that they cannot be distinguished from a family of random mappings.

A MAC algorithm is used as follows (cf. Fig. 8): Alice computes for her message P the value $\mathrm{MAC}_K(P)$ and appends this MAC to the message (here MAC is an abbreviation of MAC result).

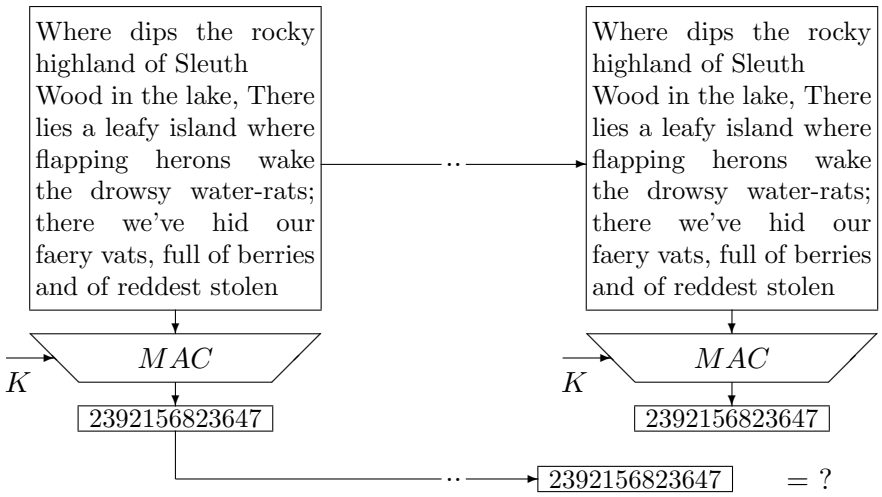


Figure 8: Using a Message Authentication Code for data authentication

Bob recomputes the value of $\mathrm{MAC}_K(P)$ based on the received message P, and verifies whether it matches the received MAC result. If the answer is positive, he accepts the message as authentic that is, as a genuine message from Alice. Eve, the active eavesdropper, can modify the message P to P', but she is not able to compute the corresponding value $\mathrm{MAC}_K(P')$ since she is not privy to the secret key K. For a secure MAC algorithm, the best Eve can do is guess the MAC result. In that case, Bob can detect the modification with high probability: for an m-bit MAC result Eve's probability of success is only $1/2^m$. The value of m lies typically between 32 and 96.

A common way to compute a MAC is to encrypt the message with a block cipher using the CBC mode which is another use of a block cipher and to keep only part of the bits of the last block as the MAC. However, Knudsen has shown that this approach is less secure than previously believed [55]. The recommended approach to use CBC-MAC consists of super-encrypting the final block with a different key, that may be derived from the first key. This scheme is known as EMAC; a security proof for EMAC has been provided by Petrank and Rackoff in [56]. Note that this scheme can be optimized by dropping the one but last encryption; this scheme is known as LMAC. NIST has decided to standardize CMAC [60]: this construction XORs a derived key before the last encryption. Almost all CBC-MAC variants are vulnerable to a birthday type attack that requires only $2^{n/2}$ known text-MAC pairs [61]. Another popular MAC algorithm is HMAC [62], which derives a MAC algorithm from a hash function such as SHA-1 [51]; an alternative for HMAC is MDx-MAC [61]. A large number of MAC algorithms has been standardized in ISO/IEC 9797 [63].

For data authentication, the equivalent of the one-time pad exists which implies that a MAC algorithm can be designed that is unconditionally secure in the sense that the security of the MAC algorithm is independent of the computing power of the opponent. The requirement is again that the secret key is used only once. The basic idea of this approach is due to G.J. Simmons [9] who defined authentication codes and Carter and Wegman [64, 65] who used the term universal hash functions. The first ideas date back to the 1970s. It turns out that these algorithms can be computationally extremely efficient since the properties required from this primitive are combinatorial rather than cryptographic. The combinatorial property that is required is that if one takes the average over a key, the function values or pairs of function values need to be distributed almost uniformly. This property is much easier to achieve than cryptographic properties which require that it should be hard to recover the key from input-output pairs which is a much stronger requirement than a close to uniform distribution. Recent constructions are therefore one order of magnitude faster than other cryptographic primitives, such as encryption algorithms and hash functions, and achieve speeds up to 1-2 cycles/byte on a Pentium III for messages longer than 256 bytes (*e.g.,* UMAC [66] The most widely used schemes are all variants of the polynomial hash function (see

[69]). The key consists of two n-bit words denoted with K_1 and K_2. The plaintext P is divided into t n-bit words, denoted with P_1 through P_t. The MAC value, which consists of a single n-bit word, is computed based on a simple polynomial evaluation:

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

where addition and multiplication are to be computed in $GF(2^n)$, the finite field with 2^n elements. It can be proved that the probability of creating another valid message/MAC pair is upper bounded by $t/2^n$. A practical choice is $n = 64$ which results in a 128-bit key. For messages up to one megabyte, the success probability of a forgery is less than $1/2^{47}$. For every message a new key K_1 is required. Note that some designers suggest that K_2 can be reused, but this is not recommended. The keys K_1 and K_2 can be generated from a short initial key using an additive stream cipher or a pseudo-random function, but then the unconditional security is lost. However, it can be argued that it is easier to understand the security of this scheme than that of a computationally secure MAC algorithm. An even better way to use universal hash functions is to apply a pseudo-random function to its output concatenated with a value that occurs only once, for example, a serial number or a large random number. The GMAC polynomial hash function defined over $GF(2^{128})$ is standardized by NIST [67]; Intel has added in 2010 a specific instruction (PCLMULQDQ) to optimize this function. GMAC has the weakness that it reuses K_2 for every message – the motivation for this unfortunate decision was that the first implementations used precomputed tables with the values of $(K_2)^i$. As a consequence, GMAC is rather fragile. An alternative using arithmetic in the prime field $GF(2^{130} - 5)$ is Poly1305-AES [68]) that is being deployed with ChaCha20; it does not require a dedicated instruction to improve its performance.

2.1.3 Authenticated Encryption

It was well understood in the 1980s that performance benefits could be achieved by combining confidentiality with data authentication. Unfortunately many *ad hoc* constructions were broken and the security requirements were poorly understood. Today it is clear that one should never protect confidentiality without also providing data authentication; this operation is called authenticated encryption. There may be settings (such as industrial control systems) where data authentication is required without the need for confidentiality).

A straightforward solution is to use separate techniques for encryption and authentication. The analysis of the security is rather subtle, but it is clear that independent keys are required for both operations. Moreover, the preferred option is Encrypt-then-MAC: apply the MAC algorithm to the ciphertext since this order of operations protects the encryption algorithm against chosen ciphertext attacks. SSH uses Encrypt-and-MAC: the MAC is computed on the plaintext and appended to the ciphertext. TLS uses MAC-then-Encrypt: the MAC is computed on the plaintext and encrypted together with the plaintext. The last two choices have resulted in practical chosen ciphertext attacks [27].

A more efficient solution is to combine data encryption and data authentication in a single operation with a single key. The first secure solutions were IAPM [57], XECB and XCBC [58] and the more efficient OCB [59] variants that improved over IAPM; unfortunately these modes are covered by patents, which impeded their adoption. Subsequently NIST decided to standardize two authenticated encryption schemes that are less efficient but not encumbered by patents and which use the CTR mode of encryption in combination with a MAC algorithm: CCM [70] uses CMAC and GCM [67] uses GMAC. CCM has the disadvantage that CMAC cannot be parallelized, while the GCM mode is brittle – attacks are known under nonce reuse or shortening of the MAC output.

In view of this the CAESAR competition[4] has been launched in 2012 to search for more efficient

[4] https://competitions.cr.yp.to/caesar.html

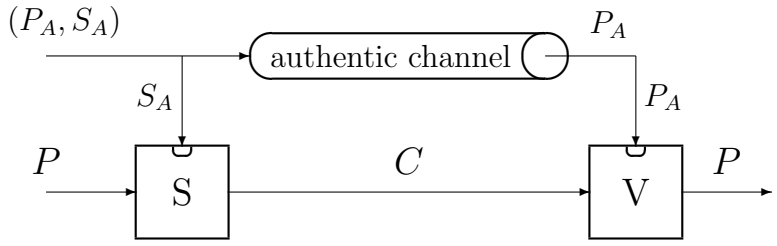


Figure 9: A digital signature scheme with message recovery based on a trapdoor one-way permutation; S and V denote the signing and verification operation respectively

and more robust solutions for authenticated encryption. From the 54 submissions, 14 have made it to the third round where they are being benchmarked against the OCB mode. It is expected that the final portfolio will be announced in 2018.

2.2 Digital Signatures

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

We now explain how the RSA algorithm can be used to create digital signatures with message recovery. The RSA mapping is a bijection, more specifically a trapdoor one-way permutation. If Alice wants to sign some information P intended for Bob, she adds some redundancy to the information, resulting in $\tilde{P}$ and decrypts the resulting text with her secret key. This operation can only be carried out by Alice. Upon receipt of the signature, Bob encrypts it using Alice's public key and verifies that the information $\tilde{P}$ has the prescribed redundancy. If so, he accepts the signature on P as valid. Such a digital signature which is a signature with message recovery requires the following condition on the public-key system: $E_{P_A}(D_{S_A}(\tilde{P})) = \tilde{P}$. Anyone who knows Alice's public key can verify the signature and recover the message from the signature.

Note that if the redundancy is left out, any person can pick a random ciphertext C^* and claim that Alice has signed $P^* = C^{*\,e} \bmod n$. It is not clear that P^* is a meaningful message which will require some extra tricks, but it shows why redundancy is essential. A provably secure way to add the redundancy is PSS-R [71]; however, in practice other constructions are widely used, and most of them combine a hash function with a digital signature scheme.

If Alice wants to sign very long messages, digital signature schemes with message recovery result in signatures that are as long as the message. Moreover, signing with a public-key system is a relatively slow operation. In order to solve these problems, Alice does not sign the information itself but rather the hash result of the information computed with an MDC (see also Fig. 10). This approach corresponds to the use of an MDC to replace the authenticity of a large text by that of a short hash value (cf. Sect. 2.1). The signature now consists of a single block which is appended to the information. This type of signature scheme is sometimes called a digital signature with appendix. In order to verify such a signature, Bob recomputes the MDC of the message and encrypts the signature with Alice's public key. If both operations give the same result, Bob accepts the signature as valid. MDCs used in this way need to be collision resistant: if Alice can find two

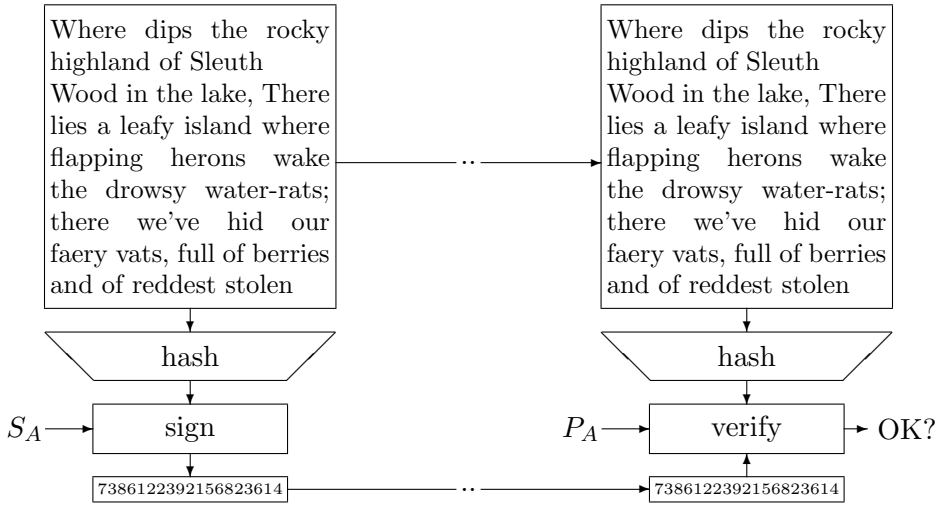


Figure 10: A digital signature scheme with appendix

different messages (P, P') with the same hash result, she can sign P and later claim to have signed P' (P and P' will have the same signature!).

Note that there exist other signature schemes with appendix, such as the DSA from FIPS 186 [72] which are not derived immediately from a public-key encryption scheme. For these schemes it is possible to define a signing operation using the secret key and a verification operation using the public key without referring to decryption and encryption operations. The security of the DSA is based on the discrete logarithm problem as the Diffie-Hellman scheme. There also exists an elliptic-curve based variant of DSA called ECDSA. There are many more digital signature schemes than public-key encryption schemes. Other digital signature schemes include ESIGN, Fiat-Shamir, Guillou-Quisquater and Schnorr.

2.2.1 Certificates

Digital signatures move the problem of authenticating data to the problem of authenticating a link between an entity and its public key. This problem can be simplified using digital *certificates*. A certificate is a digital signature of a third party on an entity's name, its public key, and additional data, such algorithms and parameters, key usage, begin and end of validity period. It corresponds roughly to an electronic identity. The verification of a certificate requires the public key of the trusted third party, and in this way the problem of authenticating data will be replaced by authenticating a single public key. In principle authenticating this key is very easy: it can be published in a newspaper, listed on a web page, or added in a browser. The infrastructure that provides public keys is called a Public Key Infrastructure (PKI). Deploying such an infrastructure in practice is rather complex since it is necessary to consider revocation of certificates, upgrades of keys, multiple trusted parties, and integration of the public key management with the application (see for example [73]).

3 Analysis and Design of Cryptographic Algorithms

In this section we compare several approaches in cryptography. Next we describe the typical phases in the life of a cryptographic algorithm. Finally we discuss why, in spite of the progress in cryptology, weak cryptographic algorithms are still in use and which problems cryptographers are facing in the next decades.

3.1 Different Approaches in Cryptography

Modern cryptology follows several approaches: the information theoretic approach, the complexity theoretic approach, the bounded-storage approach, the quantum cryptology approach and the system based approach. These approaches differ in the assumptions about the capabilities of an opponent, in the definition of a cryptanalytic success, and in the notion of security.

From the viewpoint of the cryptographer, the most desirable are unconditionally secure algorithms. This approach is also known as the *information theoretic* approach. It was developed in the seminal work of C. Shannon in 1943 and published a few years later [12]. This approach offers a perfect solution since the security can be proved independent of the computational power of the opponent, and the security will not erode over time. However, few such schemes exist that are secure in this model; examples are the one-time pad (Sect. 1.1) and the polynomial MAC algorithm (Sect. 2.1.2). While they are computationally extremely efficient, the cost in terms of key material may be prohibitively large. For most applications users have to live with schemes that offer only conditional security.

A second approach is to reduce the security of a cryptographic algorithm to that of other well known difficult problems or to that of other cryptographic primitives. The *complexity theoretic* approach starts from an abstract model for computation and assumes that the opponent has limited computing power within this model [74]. The most common models used are Turing machines and RAM models. This approach was started in cryptology by Yao [14] in 1982. It has many positive sides and has certainly contributed towards moving cryptology from an art to a science:

- It forces the formulation of exact definitions and the clear statement of security properties and assumptions, which may seem trivial, but it has taken the cryptographic community a long time to define what secure encryption, secure message authentication, secure digital signatures, and secure authenticated encryption are. It turns out that there are many variants for these definitions depending on the power of the opponent and the goals, and it takes a substantial effort to establish the relationship between them. For more complex primitives, such as e-payment, e-voting, interactions with more than two parties, and in general multi-party computation establishing correct definitions is even more complex.

- The complexity theoretic approach results in formal reductions. It can be formally proven that if a particular object exists, another object exists as well. For example, one-way functions imply digital signature schemes, or the existence of pseudo-random permutations, which corresponds to a secure block cipher, is sufficient to prove that the CBC mode with secret and random initial value is a 'secure' mode for encryption assuming that the opponent can choose plaintexts and observe the corresponding ciphertexts. The terms secure have very specific definitions as explained above. These reductions often work by contradiction. For example, it can be shown that if an opponent can find a weakness in CBC-encryption using a chosen plaintext attack, the implication is that there is a weakness in the underlying block cipher which allows a distinction of this block cipher from a pseudo-random permutation. Once the proofs are written down, any person can verify them which is very important, as it turns out that some of these proofs have very subtle errors.

The complexity theoretic approach also has some limitations:

- Many cryptographic applications need building blocks, such are one-way functions, one-way permutations, collision resistant compression functions, and pseudo-random functions, which cannot be reduced to other primitives. In terms of the existence of such primitives, complexity theory has only very weak results: it is not known whether one-way functions exist. In non-uniform complexity, which corresponds to Boolean circuits, the best result proved thus far is that there exist functions which are twice as hard to invert as to compute, which is far

too weak to be of any use in cryptography [75]. To some extent cryptographers need to rely on number theoretic assumptions, such as the assumptions underlying the security of the Diffie-Hellman protocol and RSA. For the others, we rely on unproven assumptions on functions, such as RC4, DES, AES, RIPEMD-160, SHA-1, SHA-2, and SHA-3 which means that even when there are strong reductions, the foundations of the reductions may well be weak.

- Sometimes the resulting scheme is much more expensive in terms of computation or memory than a scheme without security proof; however, in the last decade a substantial effort has been made to improve the efficiency of the constructions for which there exists a security proof.
- The security proof may not be very efficient. Some reductions are only asymptotic or are very weak. For example for RSA-OAEP, if the security property is violated that is, a random ciphertext can be decrypted, the security assumption, in this case the computation of a modular eth root is infeasible, is only violated with a very small probability. Many reductions require the random oracle assumption, that states that the hash function used in the scheme behaves as a perfectly random function. It has been shown that this approach can result in security proofs for insecure schemes; however, these reductions still have some value as a heuristic in the sense that these schemes are typically better than schemes for which no reduction in the random oracle model is known.

The term concrete complexity has been coined by Bellare and Rogaway to denote security proofs with concrete reductions focusing on efficiency that is, without asymptotics and hidden constants. For an overview of complexity theoretic results in cryptology, the reader is referred to the work of Goldwasser and Bellare [76] and Goldreich [77].

There has also been some interest in the bounded-storage model where it is assumed that the storage capacity of an adversary is limited (see for example [78]). This model can be considered to be part of the information theoretic approach if one imposes that only limited information is available to the adversary or part of the complexity theoretic approach if space rather than time is considered to be limited.

Quantum cryptology does not work based on any computational assumptions, but rather it starts from the assumption that quantum physics provides a complete model of the physical world. It relies on concepts such as the Heisenberg uncertainty principle and quantum entanglement. Quantum cryptography provides a means for two parties to exchange a common secret key over an authenticated channel. The first experimental results have been obtained in the early 1990s [79] and ten years later several companies offer encryption products based on quantum cryptography. While this offers a fascinating approach, quantum cryptology needs authentic channels which seems to make them not very useful to secure large open networks such as the internet. Indeed, without public-key cryptography, the only way to achieve an authentic channel is by establishing prior secrets manually; this seems rather impractical for a large network. Quantum networks consist of point-to-point connections, which opens the door to attackers that go after the link nodes. It is also questionable whether the security of the implementations can be formally reduced to the laws of physics (as currently known). Moreover, most implementations today have practical limitations. They typically allow for low speeds only although high speed links have been demonstrated recently; and the communication distances are limited. For high speed communication, one still needs to stretch the short keys with a classical stream cipher; of course, the resulting scheme then depends on standard cryptographic assumptions which to some extent defeats the purpose of using quantum cryptology in the first place. Moreover, it can be expected that the current implementations are vulnerable to side channel attacks as discussed below.

In view of the limitations of the above approaches, modern cryptology still has to rely on the *system-based* or *practical approach.* This approach tries to produce practical solutions for

building blocks such as one-way functions, pseudo-random bit generators or stream ciphers, pseudo-random functions, and pseudo-random permutations. The security estimates are based on the best algorithm known to break the system and on realistic estimates of the necessary computing power or dedicated hardware to carry out the algorithm. By trial and error procedures, several *cryptanalytic principles* have emerged, and it is the goal of the designer to avoid attacks based on these principles. The second aspect is to design *building blocks with provable properties*, and to assemble such basic building blocks to design cryptographic primitives. The complexity theoretic approach has also improved our understanding of the requirements to be imposed on these building blocks. In this way, this approach is also evolving towards a more scientific approach. Nevertheless, it seems likely that for the next decades cryptographers still need to rely the security of some concrete functions such as AES, SHA-3, and their successors.

3.2 Life Cycle of a Cryptographic Algorithm

This section discusses the life cycle of a cryptographic algorithm, evaluates the impact of open competitions, and compares the use of public versus secret algorithms.

A cryptographic algorithm usually starts with a new idea of a cryptographer and a first step should be an evaluation of the security properties and of the cryptographic algorithm in which the cryptographer tries to determine whether or not the scheme is secure for the intended applications. If the scheme is unconditionally secure, one has to write the proofs and provide convincing arguments why the model is correct and matches the application. For computational security, it is again important to write down security proofs and check these for subtle flaws. Moreover, a cryptographer has to assess whether the assumptions behind the proofs are realistic. For the system-based approach, it is important to prove partial results and write down arguments which should convince others of the security of the algorithm. Often such cryptographic algorithms have security parameters, such as the number of steps, the size of the key. It is then important to give lower bounds for these parameters and to indicate the value of the parameters which corresponds to a certain security level.

The next step is the publication of the algorithm at a conference, in a journal, or in an Internet Request for Comment (RFC) which hopefully results in an *independent* evaluation of the algorithm. Often more or less subtle flaws are then discovered by other researchers. These flaws can vary from small errors in proofs to complete security breaks. Depending on the outcome, this evaluation can lead to a small fix of the scheme or to abandoning the idea altogether. Sometimes such weaknesses can be found in real-time when the author is presenting his ideas at a conference, but often evaluating a cryptographic algorithm is a time consuming task; for example, the design effort of the Data Encryption Standard (DES) has been more than seventeen man-years and the open academic evaluation since that time has taken a much larger effort. While no good estimates are available for AES and SHA-3, one can expect that these efforts are much larger. Cryptanalysis is quite destructive; in this respect it differs from usual scientific activities, even when proponents of competing theories criticize each other.

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

As will be explained below, even if no new security weaknesses are found, the security of a cryptographic algorithm degrades over time; if the algorithm is not parameterized, the moment will come when it has to be taken out of service or when its parameters need to be upgraded and often such upgrades are not planned for.

3.2.1 Open Competitions

In the past several open competitions have been launched to stimulate this open and independent evaluation process: such competitions were organized in the US for the selection of DES (1974), AES (1997) and SHA-3 (2008), in Europe by the RIPE (1988-1992) [**?**] NESSIE (2000-2003) [41] and eSTREAM (2004-2008) projects and in Japan by the CRYPTREC project (2000-present) [**?**]. While each of these competitions has slightly different goals, it is clear that they have advanced the state of the art and have helped to develop better solutions. The main disadvantage of most of these competitions is that the results are not much better than the state of the art at the time that the competition started which is a direct consequence of the fact that the designs cannot be modified too much during the process. However, it turns out that the lessons learned during these competitions have been very valuable.

3.2.2 Public versus Secret Algorithms

The open and independent evaluation process described above offers a strong argument for publishing the details of a cryptographic algorithm including the design rationale and full evaluation performed by the designers. Publishing the algorithm opens it up for public scrutiny and is the best way to guarantee that it is as strong as claimed. Note that a public algorithm should not be confused with a public-key algorithm. Published algorithms can be standardized, and will be available from more than one source.

Nevertheless, certain governments and organizations prefer to keep their algorithms secret. These groups argue that obtaining the algorithm raises an additional barrier for the attacker. Moreover, governments or organizations may want to protect their know-how on the design of cryptographic algorithms. Note, however, that obtaining a description of the algorithm is often not harder than just bribing a single person. An unacceptable excuse for using a secret algorithm is the desire to hide the fact that an insecure algorithm is being used. The use of secret algorithms is acceptable in closed systems provided that sufficient experience and resources are available for *independent* evaluation and re-evaluation of the algorithm.

3.3 Insecure Versus Secure Algorithms

Today secure cryptographic algorithms are publicly available that offer good performance at an acceptable cost. Nevertheless, one finds in many applications insecure cryptographic algorithms. This section explains some of the technical reasons for this problem. We will not focus on other explanations such as plain incompetence (a 'do-it-yourself' attitude), political reasons (national security and law enforcement) and economic reasons (cost of upgrade too high compared to economic benefits).

3.3.1 Brute Force Attacks Become Easier over Time

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

A typical brute force attack against an encryption or MAC algorithm is an exhaustive key search which is equivalent to breaking into a safe by trying all the combinations of the lock. The lock should be designed such that a brute force attack is not feasible in a reasonable amount of time. This attack requires only a few known plaintext/ciphertext or plaintext/MAC pairs which can always be obtained in practice. It can be precluded by increasing the key length. Note that

adding one bit to the key doubles the time for exhaustive key search. It should also be guaranteed that the key is selected uniformly at random in the key space.

On a standard PC, trying a single key for a typical algorithm requires between 0.01 and 1 μs depending on the complexity of the algorithm. For example, a 40-bit key can be recovered in 0.1-10 days. If a Local Area Network (LAN) with 1000 machines can be used, the key can be recovered in seconds to minutes. For a 56-bit key such as DES, a key search requires a few months if several thousand machines are available as has been demonstrated in the first half of 1997. However, if dedicated hardware is used, a different picture emerges. M. Wiener has designed in 1993 a US$ 1 million hardware DES key search machine that can recover a 56-bit DES key in about three hours [80]. If such a machine would be built in 2018, it would be about 100 000 times faster and thus recovering a 56-bit key would take 0.1 second on average. In 1998, a US$ 250,000 machine called Deep Crack was built that could find a 56-bit DES key in about 4 days [81]; the design which required 50 percent of the cost has been made available for free.

These numbers makes one wonder how the US government could publish in 1977 a block cipher with a 56-bit key. However, it should be taken into account that a variant of Moore's law formulated in 1967 [82] states that computers will double their speed every 18 months which implies that in 1977, recovering a 56-bit key with a US$ 10 million hardware machine would take about 20 days; such a machine was clearly only feasible for very large organizations including the US government. This discussion also explains the initial controversy over the DES key length.

Experts believe that Moore's law will be holding for at least another ten years which means that if data needs to be protected for ten years against an opponent with a budget of US$ 20 million, a key length of at least ninety bits is needed, which corresponds to the security level of three-key triple-DES (see Sect. 1.1.3). However, as the cost of increasing the key size is quite low, it is advisable to design new algorithms with variable key size from 128 to 256 bits. Indeed, searching for the smallest AES key (128 bits) is a factor 2^{72} times more expensive than finding a DES key. Even with a key search machine of US$ 10^{12}, it would take in 2024 one million years to recover a 128-bit key. This calculation shows that if symmetric algorithms with 128-bit keys are used, it may no longer necessary to worry about exhaustive key search for the next decades.

3.3.2 Shortcut Attacks Become more Efficient

Many algorithms are less secure than suggested by the size of their external parameters. It is often possible to find more effective attacks than trying all keys (see for example the discussion on two-key and three-key triple-DES in Sect. 1.1.3). Assessing the strength of an algorithm requires cryptanalytic skills and experience as well as hard work. During the last decades, powerful new tools have been developed which includes differential cryptanalysis [83], which analyzes the propagation of differences through cryptographic algorithms; linear cryptanalysis [84], which is based on the propagation of bit correlations; fast correlation attacks on stream ciphers [19]; and algebraic attacks on stream ciphers [18]. For example, the FEAL block cipher with eight rounds; it was published in 1987 and can now be broken with only ten chosen plaintexts. Similar progress has been made in the area of public key cryptology, for example, with attacks based on lattice reduction [85] and factoring methods discussed in Sect. 1.2.2.

3.3.3 New Attack Models

The largest threats however originate from new attack models. One of these models is a quantum computer. Another new threat is the exploitation of side channel attacks and attacks based on fault injection.

Feynman realized in 1982 that computers based on the principles of quantum physics would be able to achieve exponential parallelism: having n components in the quantum computer would allow it to perform 2^n computations in parallel. Note that in a classical computer, having n computers can

speed up the computation with a factor up to n. A few years later, Deutsch realized that a general purpose quantum computer could be developed on which any physical process could be modeled, at least in principle. However, the first interesting application of quantum computers outside physics was proposed by Shor in 1994 [86] who showed that quantum computers were perfectly suitable to number theoretic problems, such as factoring and discrete logarithm problems (over finite fields and over elliptic and hyperelliptic curves). This result implies that if a large quantum computer could be built, all widely deployed public-key systems would be completely insecure. Building quantum computers however is a huge challenge. A quantum computer maintains state in a set of qubits. A qubit can hold a one, or a zero, or a superposition of one and zero; this superposition is the essence of the exponential parallelism, but it can be disturbed by outside influences, called decoherence. The first quantum computer with two qubits was built in 1998. In 2002 a 7-bit quantum computer was used to factor 15 [87]. In 2018 the record is a quantum computer with 49 bits; the main challenge is how to use quantum error correction to avoid decoherence. Experts are divided on the question whether sufficiently powerful quantum computers can be built in the next ten to fifteen years, but no one seems to expect a breakthrough in the next five years. For symmetric cryptography, quantum computers are less of a threat since they can reduce the time to search a $2n$-bit key to the time to search an n-bit key, using Grover's algorithm [88]. Hence doubling the key length (from 128 to 256 bits) offers an adequate protection. Collision search on a quantum computer may reduce from $2^{n/2}$ steps to $2^{n/3}$ steps [88], so it is sufficient to increase the number of bits of a hash result from 256 to 384.

Cryptographic algorithms have to be implemented in hardware or software, but it should not be overlooked that software runs on hardware. The opponent can try to make life easier by obtaining information from the hardware implementation, rather than trying to break the cryptographic algorithm using fast computers or clever mathematics. Side channel attacks have been known for a long time in the classified community; these attacks exploit information on the time to perform a computation [89], on the power consumption [90], or on the electromagnetic radiation [91, 92] to extract information on the plaintext or even the secrets used in the computation. A very simple side channel attack on the one-time pad (see Sect. 1.1) exploits the fact that if there is a logical zero on the line, this can be the result of $0 \oplus 0$ or $1 \oplus 1$. If the device implementing the scheme is not properly designed, the two cases may result in different electrical signals which leaks immediately information on half of the plaintext bits.

Protecting implementations against side channel attacks is notoriously difficult and requires a combination of countermeasures at the hardware level such as adding noise, special logic, decoupling power source, at the algorithmic level, for example, blinding and randomization and at the protocol level, for example, frequent key updates (see [93]). While many countermeasures have been published, many fielded systems are still vulnerable. This vulnerability is in part due to cost reasons and delays in upgrades and also due to the development of ever more sophisticated attacks. Developing efficient implementations which offer a high security level against side channel attacks is an important research challenge for the coming years.

The most powerful attacks induce errors in the computations, for example, by varying clock frequency or power level or by applying light flashes. Such attacks can be devastating because small changes in inputs during a cryptographic calculation typically reveal the secret key material [94]. Protecting against these attacks is non-trivial since it requires continuous verification of all calculations, which should also include a check on the verifications, and even this countermeasure may not be sufficient as has been pointed out in [95]. It is also clear that pure cryptographic measures will never be sufficient to protect against such attacks.

4 Conclusions

This chapter only covers part of the issues in modern cryptology since the discussion is restricted to an overview of cryptographic algorithms. Other problems solved in cryptography include entity authentication, timestamping, sharing of secrets, electronic cash, and cryptocurrencies. Many interesting problems are studied under the concept of computing on encrypted data and secure multi-party computation, such as electronic elections and the generation and verification of digital signatures in a distributed way. An important aspect is the underlying key management infrastructure which ensures that secret and public keys can be established and maintained throughout the system in a secure way which is where cryptography meets the constraints of the real world.

This article has demonstrated that while cryptology has made significant progress as a science, there are some interesting research challenges ahead both from a theoretic and from a practical viewpoint. Progress needs to be made in theoretical foundations, such as how to prove that a problem is hard, the development of new cryptographic algorithms, such as new public-key cryptosystems which do not depend on algebraic number theory and light-weight or low footprint secret-key cryptosystems, and secure and efficient implementations of cryptographic algorithms. Hopefully this will allow the cryptographic community to build schemes that offer long term security at a reasonable cost.

References

[1] Menezes, A.J., van Oorschot, P.C., and Vanstone, S., *Handbook of Applied Cryptography,* CRC Press, 1997.

[2] Kahn, D., *The Codebreakers. The Story of Secret Writing,* New York, NY: MacMillan, 1967.

[3] Singh, S., *The Code Book. The Science of Secrecy from Ancient Egypt to Quantum Cryptography,* Anchor, 2000.

[4] Stallings, W., *Cryptography and Network Security,* 7th edition, Pearson, 2016.

[5] Blaze, M., "Rights amplification in master-keyed mechanical locks." *IEEE Security & Privacy*, Vol. 1, No. 2, 2003, pp. 24–32.

[6] Bellovin, S.M., "Frank Miller: Inventor of the One-Time Pad." *Cryptologia*, Vol. 35, No. 3, 2011, pp. 203–222.

[7] Koblitz, N., *A Course in Number Theory and Cryptography,* Berlin: Springer-Verlag, 1987.

[8] *State of the Art in Applied Cryptography, Lecture Notes in Computer Science 1528*, B. Preneel and V. Rijmen (eds.), Berlin: Springer-Verlag, 1998.

[9] *Contemporary Cryptology: The Science of Information Integrity,* G.J. Simmons (ed.), New York, NY: IEEE Press, 1991.

[10] Stinson, D., *Cryptography. Theory and Practice,* 3rd edition, CRC Press, 2005.

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

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

[13] Haynes, J.E., and Klehr, H., *Venona. Decoding Soviet Espionage in America,* Yale University Press, 1999.

[14] Yao, A.C., "Theory and applications of trapdoor functions." In *Proc. 23rd IEEE Symposium on Foundations of Computer Science*, pp. 80–91, Washington DC: IEEE Computer Society, 1982.

[15] Rueppel, R.A., *Analysis and Design of Stream Ciphers,* Berlin: Springer-Verlag, 1986.

[16] Biryukov, A., Shamir, A., and Wagner, D., "Real time cryptanalysis of A5/1 on a PC." In *Fast Software Encryption, Lecture Notes in Computer Science 1978,* Schneier, B. (ed.), pp. 1–18, Berlin: Springer-Verlag, 2001.

[17] Barkan, E., Biham, E., and Keller N. "Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication." *Journal of Cryptology*, Vo. 21, No. 3, 2008, pp. 392–429 (extended version of the Crypto 2006 paper).

[18] Courtois, N., and Meier, W., "Algebraic attacks on stream ciphers with linear feedback." In *Advances in Cryptology, Proc. Eurocrypt'03, Lecture Notes in Computer Science 2656,* Biham, E. (ed.), pp. 345–359, Berlin: Springer-Verlag, 2003.

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

[20] Fluhrer, S., Mantin, I., and Shamir, A., "Weaknesses in the key scheduling algorithm of RC4." In *Selected Areas in Cryptography, Lecture Notes in Computer Science 2259,* Vaudenay, S. and Youssef, A. (eds.), pp. 1–24, Berlin: Springer-Verlag, 2001.

[21] AlFardan, N.J., Bernstein, D.J. Paterson, K.G., Poettering, B. and Schuldt J.C.N., "On the Security of RC4 in TLS." *USENIX Security Symposium 2013*, pp. 305–320.

[22] Ågren, M., Hell, M., Johansson, T., and Meier, W. "A New Version of Grain-128 with Authentication." *Paper presented at Symmetric Key Encryption Workshop 2011*, Lyngby, Denmark.

[23] De Cannière, C. and Preneel, B., "Trivium." In *The eSTREAM Finalists, Lecture Notes in Computer Science 4986,* Robshaw M.J.B.. and Billet O. (ed.s), pp. 244–266, Berlin: Springer-Verlag, 2008.

[24] Nir, Y. and Langley, A., "ChaCha20 and Poly1305 for IETF Protocols." *Request for Comments 7539 (Informational)*, May 2015.

[25] Wu, H. "The Stream Cipher HC-128." In *The eSTREAM Finalists, Lecture Notes in Computer Science 4986,* Robshaw M.J.B.. and Billet O. (ed.s), pp. 39–47, Berlin: Springer-Verlag, 2008.

[26] Bellare, M., Desai, A., Jokipii, E., and Rogaway, P., "A concrete security treatment of symmetric encryption." In *Proc. 3th Annual Symposium on Foundations of Computer Science, FOCS '97*, pp. 394–403, Washington DC: IEEE Computer Society, 1997.

[27] Canvel, B., Hiltgen, A.P., Vaudenay S., and Vuagnoux, M., "Password Interception in a SSL/TLS Channel." In *Advances in Cryptology, Proc. Crypto'03, Lecture Notes in Computer Science 2729,* Boneh, D. (ed.), pp. 583–599, Berlin: Springer-Verlag, 2003.

[28] FIPS 81, *DES Modes of Operation,* Federal Information Processing Standard, NBS, US Dept. of Commerce, December 1980.

[29] NIST, *SP 800-38A Recommendation for Block Cipher Modes of Operation – Methods and Techniques,* December 2001.

[30] ISO/IEC 10116, *Information technology – Security techniques – Modes of operation for an n-bit block cipher*, 2017.

[31] FIPS 46, *Data Encryption Standard,* Federal Information Processing Standard, NBS, U.S. Dept. of Commerce, January 1977 (revised as FIPS 46-1, 1988; FIPS 46-2, 1993, FIPS 46-3, 1999).

[32] Knudsen, L.R., *Block Ciphers – Analysis, Design and Applications,* PhD thesis, Aarhus University, Denmark, 1994.

[33] FIPS 197, *Advanced Encryption Standard,* Federal Information Processing Standard, NIST, U.S. Dept. of Commerce, November 26, 2001.

[34] Daemen, J., and Rijmen, V., *The Design of Rijndael. AES – The Advanced Encryption Standard,* Berlin: Springer-Verlag, 2001.

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

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

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

[38] Bellare, M., and Rogaway, P., "Random oracles are practical: A paradigm for designing efficient protocols." In *Proc. ACM Conference on Computer and Communications Security*, pp. 62–73, New York, NY: ACM Press, 1993.

[39] Shoup, V., "OAEP reconsidered." In *Advances in Cryptology, Proc. Crypto'01, Lecture Notes in Computer Science 2139,* Kilian, J. (ed.), pp. 239–259, Berlin: Springer-Verlag, 2001.

[40] Fujisaki, E., Okamoto, T., Pointcheval, D., and Stern, J., "RSA-OAEP is secure under the RSA assumption." In *Advances in Cryptology, Proc. Crypto'01, Lecture Notes in Computer Science 2139,* Kilian, J. (ed.), pp. 260–274, Berlin: Springer-Verlag, 2001.

[41] NESSIE, `http://www.cryptonessie.org`.

[42] Lenstra, A.K., and Verheul, E.R., "Selecting cryptographic key sizes." *Journal of Cryptology*, Vol. 14, 2001, pp. 255–293.

[43] Shamir, A., and Tromer, E., "Factoring large numbers with the TWIRL device." In *Advances in Cryptology, Proc. Crypto'03, Lecture Notes in Computer Science 2729,* Boneh, D. (ed.), pp. 1–26, Berlin: Springer-Verlag, 2003.

[44] Cohen,H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K., and Vercauteren, F., *Handbook of Elliptic and Hyperelliptic Curve Cryptography,* Cohen, H., and Frey, G. (eds.), Chapman & Hall/CRC, 2005.

[45] Blake, I.F., Seroussi, G., and Smart, N.P., *Elliptic Curves in Cryptography,* Cambridge University Press, 1999.

[46] Hankerson, D., Menezes, A., and Vanstone, S., *Guide to Elliptic Curve Cryptography*, Berlin: Springer-Verlag, 2004.

[47] Abdalla, M., Cid, C., Gierlichs, B., Hulsing, A., Luykx, A., Paterson, K.G., Preneel, B. Sadeghi A.-R., Spies, T., Stam, M., Ward, M., Warinschi, B., and Watson, G., *"Algorithms, Key Size and Protocols Report."* Smart, N.P. (ed.), ECRYPT – Coordination & Support Action, October 2016, `http://www.ecrypt.eu.org/csa/documents/D5.2-AlgKeySizeProt-1.0.pdf`

[48] Wang, X., and Yu, H., "How to break MD5 and other hash functions." In *Advances in Cryptology, Proc. Eurocrypt'05, Lecture Notes in Computer Science 3494,* R. Cramer (ed.), pp. 19–35, Berlin: Springer-Verlag, 2005.

[49] Wang, X., Lin, Y.L., and Yu, H., "Finding collisions in the full SHA-1." In *Advances in Cryptology, Proc. Crypto'05, Lecture Notes in Computer Science 3621,* Shoup, V. (ed.), pp. 17–36, Berlin: Springer-Verlag, 2005.

[50] Stevens, M., Bursztein, E., Karpman P., Albertini, A., and Markov, Y., " The First Collision for Full SHA-1." In *Advances in Cryptology, Proc. Crypto'17, Part I, Lecture Notes in Computer Science 10401,* Katz, J. and Shacham, H. (eds.), pp. 570–596, Berlin: Springer-Verlag, 2017.

[51] FIPS 180, *Secure Hash Standard,* Federal Information Processing Standard (FIPS), Publication 180, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., May 11, 1993 (revised as FIPS 180-1, 1995; FIPS 180-2, 2003; FIPS 180-3, 2008; FIPS 180-4, 2015).

[52] FIPS 202, *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions,* Federal Information Processing Standard (FIPS), Publication 1202, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., August 4, 2015.

[53] Bertoni, G., Daemen, J., Peeters, M., and Van Assche G., *The Keccak SHA-3 Submission*, January 14, 2011 `http://keccak.noekeon.org`.

[54] ISO/IEC 10118, *Information technology – Security techniques – Hash-functions, Part 1: General*, 2016. *Part 2: Hash-functions using an n-bit block cipher algorithm*, 2010. *Part 3: Dedicated hash-functions*, 2003 (amended in 2006). *Part 4: Hash-functions using modular arithmetic*, 1998 (corrected 2014).

[55] Knudsen, L.R., "Chosen-text attack on CBC-MAC." *Electronics Letters*, Vol. 33, No. 1, 1997, pp. 48–49.

[56] Petrank E., and Rackoff, C., "CBC MAC for real-time data sources." *Journal of Cryptology*, Vol. 13, 2000, pp. 315–338.

[57] Jutla, C.S., "Encryption Modes with Almost Free Message Integrity." In *Advances in Cryptology, Proc. Eurocrypt'03, Lecture Notes in Computer Science 2045,* Pfitzmann, B. (ed.), pp. 529–544, Berlin: Springer-Verlag, 2003.

[58] Gligor, V.D., Donescu, P., "Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes." In *Fast Software Encryption, Lecture Notes in Computer Science 2355,* Matsui, M. (ed.), pp. 92–108, Berlin: Springer-Verlag, 2002.

[59] Rogaway, P., Bellare, M., and Black, J., "OCB: A block-cipher mode of operation for efficient authenticated encryption." *ACM Trans. Inf. Syst. Secur.*, Vol. 6, No. 3, 2003, pp. 365–403.

[60] NIST, *SP 800-38B Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication,* Special Publication 800-38B, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., May 2005 (updated 2016).

[61] Preneel, B., and van Oorschot, P.C., "MDx-MAC and building fast MACs from hash functions." In *Proc. Crypto'95, Lecture Notes in Computer Science 963,* Coppersmith, D. (ed.), pp. 1–14, Berlin: Springer-Verlag, 1995.

[62] Bellare, M., Canetti, R., and Krawczyk, H. "Keying hash functions for message authentication." In *Advances in Cryptology, Proc. Crypto'96, Lecture Notes in Computer Science 1109,* pp. 1–15, Koblitz, N. (ed.), pp. 1–15, Berlin: Springer-Verlag, 1996. Full version `http://www.cs.ucsd.edu/users/mihir/papers/hmac.html`.

[63] ISO/IEC 9797, *Information technology – Security techniques – Message Authentication Codes (MACs). Part 1: Mechanisms using a block cipher*, 2011. *Part 2: Mechanisms using a dedicated hash-function*, 2011.

[64] Carter, J.L., and Wegman, M.N., "Universal classes of hash functions." *Journal of Computer and System Sciences*, Vol. 18, 1979, pp. 143–154.

[65] Wegman, M.N., and Carter, J.L., "New hash functions and their use in authentication and set equality." *Journal of Computer and System Sciences*, Vol. 22, No. 3, 1981, pp. 265–279.

[66] Black, J., Halevi, S., Krawczyk, H., Krovetz, T., and Rogaway, P., "UMAC: Fast and secure message authentication." In *Advances in Cryptology, Proc. Crypto'99, Lecture Notes in Computer Science 1666,* Wiener, M. (ed.), pp. 216–233, pp. 216–233, Berlin: Springer-Verlag, 1999.

[67] NIST *SP 800-38D Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC,* Special Publication 800-38D, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., November 2007.

[68] Bernstein, D.J., "The Poly1305-AES message-authentication code." In *Fast Software Encryption, Lecture Notes in Computer Science 3557,* Gilbert, H. and Handschuh, H. (eds.), pp. 32–49, Berlin: Springer-Verlag, 2005.

[69] Kabatianskii, G.A., Johansson, T., and Smeets, B., "On the cardinality of systematic A-codes via error correcting codes." *IEEE Transactions on Information Theory*, Vol. IT–42, No. 2, 1996, pp. 566–578.

[70] NIST, *SP 800-38C Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality,* Special Publication 800-38C, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., May 2004 (updated 2007).

[71] Bellare, M., and Rogaway, P., "The exact security of digital signatures. How to sign with RSA and Rabin." In *Advances in Cryptology, Proc. Eurocrypt'96, Lecture Notes in Computer Science 1070,* Maurer, U. (ed.), pp. 399–414, Berlin: Springer-Verlag, 1996.

[72] FIPS 186, *Digital Signature Standard,* Federal Information Processing Standard, NIST, US Dept. of Commerce, May 1994 (revised as FIPS 186-1, 1998; FIPS 186-2, 2000; change notice published in 2001; revised as FIPS 186-3, 2009; revised as FIPS 186-4, 2013).

[73] Adams, C., and Lloyd, S. *Understanding PKI. Concepts, Standards, and Deployment Considerations,* 2nd edition, Addison-Wesley, 2003.

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

[75] Hiltgen, A.P.L., "Construction of feebly-one-way families of permutations." In *Proc. Auscrypt'92, Lecture Notes in Computer Science 718,* Seberry, J. and Zheng, Y. (eds.), pp. 422–434, Berlin: Springer-Verlag, 1993.

[76] Goldwasser, S., and Bellare, M., *Lecture Notes on Cryptography,* `http://www.cs.ucsd.edu/users/mihir/papers/gb.html`.

[77] Goldreich, O., *Foundations of Cryptography: Volume 1, Basic Tools,* Cambridge University Press, 2001.

[78] Dziembowski, S., and Maurer, U., "Optimal randomizer efficiency in the bounded-storage model." *Journal of Cryptology*, Vol. 17, 2004, pp. 5–26.

[79] Bennett, C.H., Brassard, G., and Ekert, A.K. "Quantum cryptography." *Scientific American*, Vol. 267, No. 4, October 1992, pp. 50–57.

[80] Wiener, M.J., "Efficient DES Key Search." *Presented at the Rump Session of Crypto'93.* Reprinted in *Practical Cryptography for Data Internetworks,* W. Stallings (ed.), pp. 31–79, Washington DC: IEEE Computer Society, 1996.

[81] EFF, *Cracking DES. Secrets of Encryption Research, Wiretap Politics & Chip Design,* O'Reilly, 1998.

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

[83] Biham, E., and Shamir, A., *Differential Cryptanalysis of the Data Encryption Standard,* Berlin: Springer-Verlag, 1993.

[84] Matsui, M., "The first experimental cryptanalysis of the Data Encryption Standard." In *Advances in Cryptology, Proc. Crypto'94, Lecture Notes in Computer Science 839,* Desmedt, Y. (ed.), pp. 1–11, Berlin: Springer-Verlag, 1994.

[85] Joux, A., and Stern, J., "Lattice reduction: a toolbox for the cryptanalyst." *Journal of Cryptology*, Vol. 11, 1998, pp. 161–185.

[86] Shor, P.W., "Algorithms for quantum computation: discrete logarithms and factoring." In Proc. 35th Annual Symposium on Foundations of Computer Science, S. Goldwasser (ed.), pp. 124–134, Washington DC: IEEE Computer Society, 1994.

[87] Vandersypen, L.M.K., M. Steffen, G. Breyta, C.S. Yannoni, M.H. Sherwood, and I.L. Chuang, "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance." *Nature*, Vol. 414, 2001, pp. 883–887.

[88] Grover, L.K., "A fast quantum mechanical algorithm for database search." In *Proc. 28th Annual ACM Symposium on Theory of Computing*, pp. 212–219, New York, NY: ACM Press, 1996.

[89] Kocher, P., "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems." In *Advances in Cryptology, Proc. Crypto'96, Lecture Notes in Computer Science 1109,* pp. 104–113, Koblitz, N. (ed.), pp. 104–113, Berlin: Springer-Verlag, 1996.

[90] Kocher, P., Jaffe, J., and Jun, B., "Differential power analysis." In *Advances in Cryptology, Proc. Crypto'99, Lecture Notes in Computer Science 1666,* Wiener, M. (ed.), pp. 388–397, pp. 388–397, Berlin: Springer-Verlag, 1999.

[91] Gandolfi, K., Mourtel, C., and Olivier, F., "Electromagnetic analysis: Concrete results." In *Proc. Cryptographic Hardware and Embedded Systems – CHES 2001, Lecture Notes in Computer Science 2162,* C.K. Koç, D. Naccache, and C. Paar (eds.), pp. 251–261, Berlin: Springer-Verlag, 2001.

[92] Quisquater, J.-J., and Samyde, D., "ElectroMagnetic Analysis (EMA): Measures and countermeasures for smart cards." In *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Lecture Notes in Computer Science 2140,* I. Attali and T. Jensen (eds.), pp. 200–210, Berlin: Springer-Verlag, 2001.

[93] Borst, J., Preneel, B., and Rijmen, V., "Cryptography on smart cards." *Journal of Computer Networks*, Vol. 36, 2001, pp. 423–435.

[94] Boneh, D., DeMillo, R.A., and Lipton, R.J., "On the importance of eliminating errors in cryptographic computations." *Journal of Cryptology*, Vol. 14, 2001, pp. 101–119.

[95] Yen, S.-M., and Joye, M., "Checking before output may not be enough against fault-based cryptanalysis." *IEEE Transactions on Computers,* Vol. 49, No. 9, 2000, pp. 967-970.