

Chapter 2

Basic cryptosystems

This chapter starts with a look at some of the most popular cryptosystems. The description in this chapter focusses on the fundamental properties and leaves out some details, in particular proofs why certain things work the way they do. The complete underpinnings for these methods are provided in later chapters.

We learn to ask the fundamental questions: does it work correctly? How easy is the system to use for its legitimate players? How hard is it to break for others? In other words: what can we say about its security?

The first system is the *Advanced Encryption Standard* (AES), chosen from 15 candidates in a competition launched in 1997 by the National Institute of Standards and Technology (NIST), a US government institution. This system is an example of a symmetric cryptosystem in which the two protagonists (sender and receiver) share the same key. AES is characterized by its simplicity, good structure, and efficiency.

We briefly discuss two fundamentally different types of cryptosystems that we will encounter: symmetric vs. asymmetric systems. In the first type, sender and receiver share the same secret key, while in the latter type, only the receiver needs a secret key to decrypt an encrypted message and all other information is publicly available. If you have not yet seen such systems, stop here for a moment! Does this not sound contradictory? How could it possibly work?

We describe the RSA system named after its inventors Rivest, Shamir & Adleman. The security of this *asymmetric* or *public key cryptosystem* is somewhat related to the difficulty of *factoring large integers* into their prime factors.

The third example is the Diffie & Hellman key exchange protocol. Here the goal is not to send a secret message, but slightly more modest: the two players just want to agree on a common secret key (which they may

then use in some other cryptographic setting). This example introduces the idea of doing cryptography in groups. The security of such systems relies on the difficulty of computing *discrete logarithms* in these groups.

We then discuss Shamir's scheme for sharing a secret among many players so that together they know the secret but any coalition of fewer than all players has no knowledge about it. This is based on *polynomial interpolation*.

The final example is Naor & Shamir's *visual cryptography*. We include it here because of its striking effect: you have two random pictures, one on paper and one on a transparency, and when you overlay them, you can see a secret message.

2.1. Goals of cryptography

Electronic transactions and activities play an ever increasing role in our lives. Many of these need to be protected against all kinds of interference, accidental or malicious. This is the general task of *information technology security*. *Cryptography* provides some basic building blocks for secure electronic systems. Its most fundamental task is secure information transmission: someone wants to send a message over an insecure channel such as the internet to a recipient, and a third party listening in should not be able to understand the message. Following a long-standing tradition, the computers involved are personalized as BOB sending a message to ALICE, and EVE eavesdropping on the line. This is achieved by BOB *encrypting* his message x with a key K and sending the result $y = \text{enc}_K(x)$ to ALICE, and then ALICE *decrypting* y with her own key S to recover $x = \text{dec}_S(y)$. In some systems they share the same key: $K = S$. Both BOB and ALICE should be able to do their work efficiently, but EVE, knowing only y (and also K in some systems; then $K \neq S$), should not be able to decipher the message, that is, to recover x with reasonable effort.

In addition to this fundamental task, there are many other objectives in cryptography, such as securely signing a message or establishing one's identity. Can you imagine how one would do that over the internet, without meeting in person? These questions are discussed in later chapters.

Coming back to the fundamental task, we have to clarify what the "efficiency" of BOB's and ALICE's actions might mean, and the inability of EVE to recover the message. Some systems, such as AES in Section 2.2, are completely fixed. Then efficiency means that it has to be implementable for the purpose at hand, maybe for secure transmission of a pay-TV video signal at a rate of megabits per second. EVE's inability to decipher the message without the secret key might mean that it is, as far as we know, beyond the power of any adversary for as long as its secrecy is important.

In other systems, such as RSA or Diffie-Hellman (Sections 2.5 and 2.6), we have a security parameter n which may, in principle, be chosen arbitrarily. Then the standard notion from Theoretical Computer Science is that “efficient” should mean that BOB and ALICE work in time polynomial in n , and that EVE is not able to discover x in polynomial time.

The concepts of complexity theory provide a precise framework in which to state the latter task. But its basic questions, such as whether $P \neq NP$, are unresolved at this time, and the design of a system secure in this sense is an open and extremely difficult question. However, a reasonable modification asks not for an absolute proof that EVE is unable to recover the original message from the transmission, but to relate it to other problems: if she could find the message, then she would also be able to solve a well-studied open problem.

The question can be formulated as the quest for a *one-way function* f : given x , it should be easy to compute $y = f(x)$, but given some y , which occurs as an image under f , it should be hard to find an x with $f(x) = y$. Furthermore, ALICE has some (small) secret S with whose help it is actually easy to find x from y ; then f is called a *trapdoor function*. An example of a one-way function is multiplication: it is easy to multiply two integers, say two large prime numbers p and q , and find $N = p \cdot q$. But computationally it is quite difficult to recover p and q from N , although they are completely determined by N . Thus $x = (p, q)$ with $p < q$ and $f(x) = p \cdot q$ is a one-way function. No trapdoor is known for this f , but a closely related trapdoor function is used in the RSA cryptosystem (Section 2.5).

There are many variations of what it means for EVE to break such a system. Clearly it should be infeasible to efficiently recover x or ALICE’s secret key S from y . But also much weaker achievements might be considered fatal, for example if she can find out some information about the message x : is it an English text message? Does some specific word occur, such as “MasterCard” or “bomb”? It would even be dangerous if she could not do this all the time, but only for some messages, just slightly better than guessing.

EVE might have some knowledge about the possible values of x . For example, she might know (or guess) that x is a string of 1024 bits representing the Extended ASCII encoding of a 128-letter English text. Only a tiny fraction of the 2^{1024} possible x ’s are of that form: we have a sparse *message space*.

In Chapter 9, we study *security notions* which consist of *resources* and *attack goals*. One possible resource is that EVE may be able to see $\text{enc}_K(x)$ for many x ’s of her choice: *chosen plaintext attack* (CPA). A strong and hence desirable concept is the *chosen ciphertext attack*, where she can see

x for several y 's of her choice with $\text{enc}_K(x) = y$. Among the various *attack goals*, recovering the secret key or the plaintext come to mind at first. A weak and hence desirable notion here is *indistinguishability*: EVE submits two plaintexts and receives an encryption y_0 of one of them, chosen with equal probability. As above, she may ask for as many pairs (x, y) with $y = \text{enc}_K(x)$ as she wants, where she may specify either x or y ; of course, with $y \neq y_0$. Then she has to distinguish which of the two plaintexts was chosen, with probability better than just guessing. Indistinguishability means resistance against such an attack.

A combination of allowed resources and desired attack goal defines a *security notion*. A cryptographic system is *secure* in this notion if the goal cannot be accomplished with those resources, for example, if encryptions cannot be distinguished as above even using chosen ciphertexts. These things are further explained in Chapter 9 and throughout the text. For a brief impression, the reader might take a peek at Figure 9.5. An important aspect of these notions is that no specific method of attack is assumed, rather just the tools allowed and the goal to be achieved.

2.2. Advanced Encryption Standard (AES)

In the early 1970's, a team at International Business Machines developed a cryptosystem which became known as the *Data Encryption Standard* (DES). The US *National Bureau of Standards* (NBS) published it in FIPS PUB 46 on 15 January 1977 as a standard for US government cryptography, for documents that are sensitive but not classified. (The *National Security Agency* (NSA) is responsible for higher levels of security.) As a consequence, any software or hardware system with cryptographic capabilities tendered to the US government had to be based on DES. Sales to government agencies can be highly lucrative, and any company interested in them had to use DES. Thus it quickly found widespread use.

Over the years, many attacks on DES were developed, most notably differential cryptanalysis and linear cryptanalysis (Chapter 6). In response to this and concerns about its small key space, DES was strengthened by tripling its number of “rounds”: triple-DES or 3-DES. In DES, the so-called *S*-boxes provide the only nonlinear functions. They are optimized with respect to resistance to differential cryptanalysis, but their structure is rather opaque.

From the start, experts harbored suspicions—never substantiated—that the NSA might have built a “trapdoor” into DES that enabled it to decipher encrypted messages. Already in 1981, Deavours warned that

The agency [NSA] is currently capable of breaking DES using probable plaintext. The major cryptanalytic hardware

involved is rumored to consist of 4 CRAY-1 computers. Analysis takes less than a day, on the average.

Finally, on 17 July 1998 the *Electronic Frontier Foundation* (EFF) presented its US\$ 250,000 DES breaker. DES was dead, for most practical purposes. But it was still the standard and thus in heavy use . . . The standard was finally withdrawn in 2005. The US NIST, successor agency of the NBS, opened on 12 September 1997 a competition for AES, to replace DES. The requirements were for a block cipher with blocks of 128 bits, and possible key lengths of 128, 192, and 256 bits. Not surprisingly, the specifications were rather more precise than in their 1973 competition which led to the adoption of DES. 15 candidates were submitted to NIST, and pared down to a short list of five systems by August 1999. These included *MARS* from IBM's Don Coppersmith, one of the chief designers of DES, *RC6*, developed by Ron Rivest and three collaborators from RSA Laboratories, *Serpent* by Anderson, Biham, and Knudsen, and *Twofish* by Bruce Schneier's Counterpane Company. On 2 October 2000, the NIST announced the winner: AES, a system developed by the Belgian cryptographers Joan Daemen and Vincent Rijmen and originally called *Rijndael*. NIST expects this system to be secure for at least thirty years.

NIST was generally lauded for an open and well-documented procedure. One of its requirements was to make plausible that there are no hidden trapdoors, thus alleviating some of the concerns that had surrounded the DES standardization in 1977.

The features that secured Rijndael's first place in a tough competition are security—resistance against all currently known attacks—and efficiency—on a wide variety of platforms, from 8-bit smartcards to 32- or 64-bit processors. Furthermore, it has a simple algebraic description with few unexplained choices (see the end of this section), and it is implausible that they could hide a trapdoor. No effective attack is known in 2015.

AES encrypts a message of 128 bits using a key of 128, 192, or 256 bits, distinguished by designations like AES-128. It is an *iterated cipher*, in which a sequence of four operations is applied a certain number of times. Namely, it consists of 10 *rounds* at key length 128 (12 rounds at 192 and 14 rounds at 256 bits) and each round performs these four operations, except that the last one leaves out MIXCOLUMNS. Furthermore, there is an additional initial round, executing only ADDROUNDKEY. Each operation turns a 128-bit word into another 128-bit word. To describe the operations, each 128-bit word (or *state* in AES) is treated as a 4×4 matrix (or

array, or block) of 8-bit bytes:

$$(2.1) \quad \begin{array}{cccc} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{array}.$$

The four operations have the following features:

- SUBBYTES substitutes each single byte by another value,
- SHIFTRROWS permutes the bytes in each row,
- MIXCOLUMNS performs a linear transformation on each column of the matrix,
- ADDROUNDKEY adds the round key to the whole matrix.

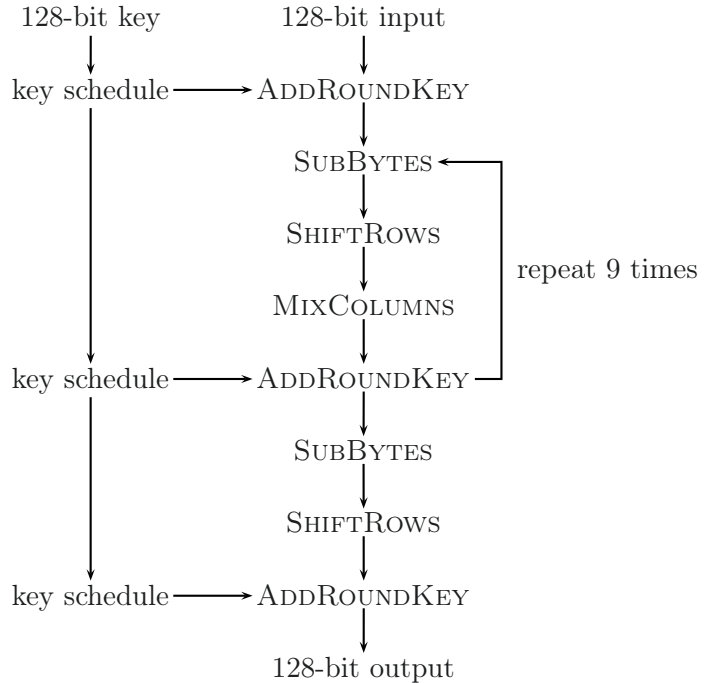


Figure 2.1: AES with a 128-bit key.

Figure 2.1 illustrates the global view. The four operations in the middle constitute one round. For the initial round, the round key is explicitly

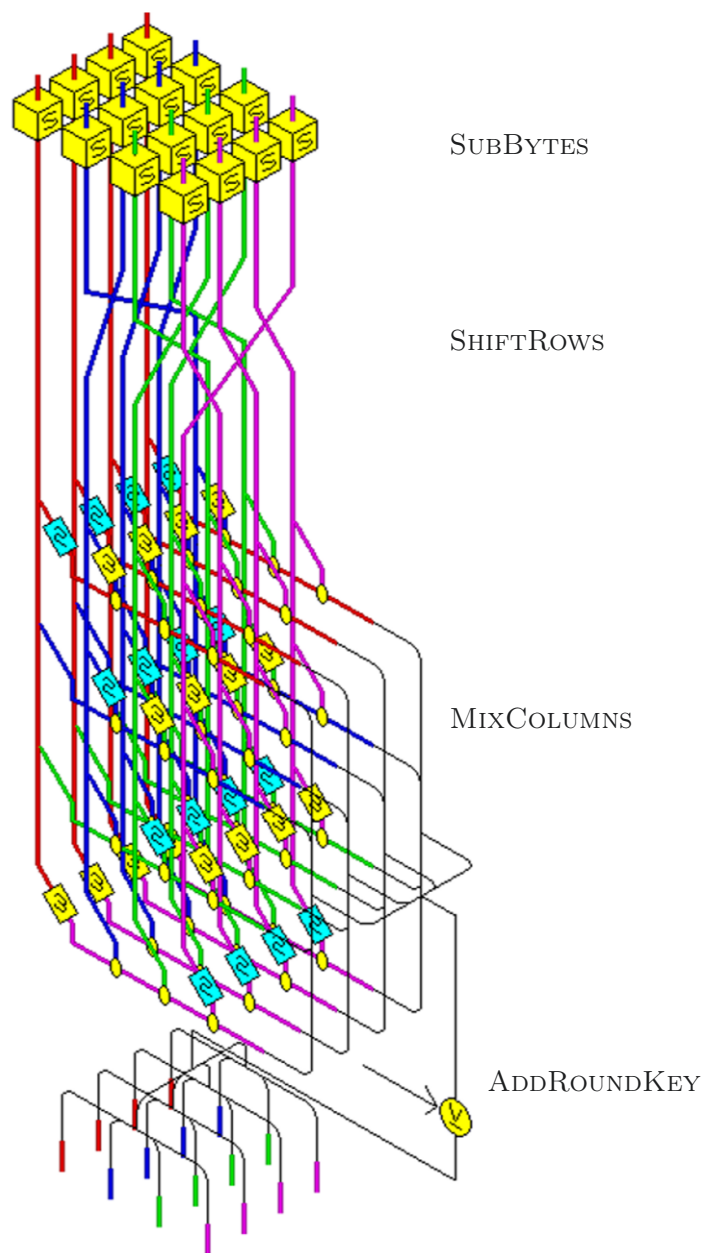


Figure 2.2: One round of AES.

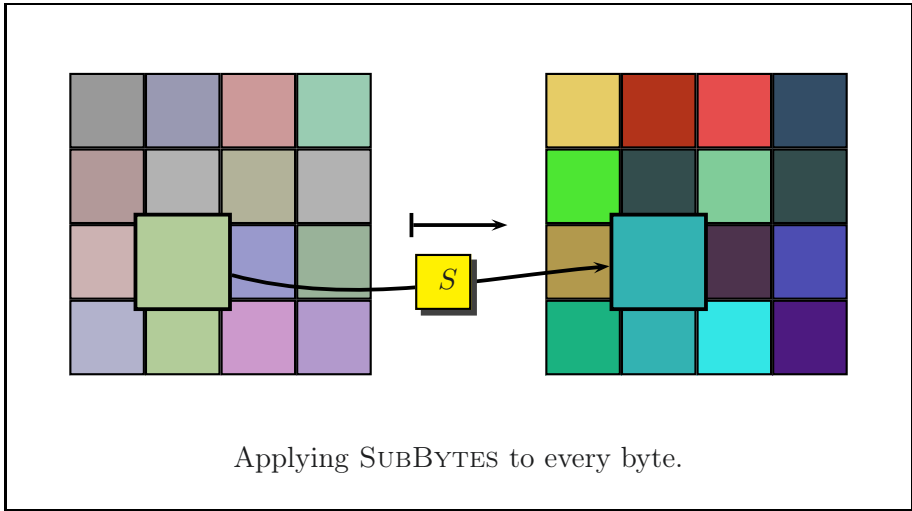


Figure 2.3: The SUBBYTES operation.

provided as the secret key to the procedure. From this, the round keys for the later rounds are calculated by the *key schedule*; see Section 2.3.

We now describe in more detail the four operations, assuming that the reader is familiar with the material in Sections 15.1 through 15.4. We see many cryptosystems in this book, including RSA and group-based cryptography, say with elliptic curves, which by their nature require some algebra. But AES is the winner in a competition for bit-oriented (or Boolean) cryptography. The elegant algebraic description that follows is witness to the *unreasonable effectiveness of algebra* in cryptography.

SUBBYTES. The basic processing unit is an 8-bit byte

$$(2.2) \quad a = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0) \in \{0, 1\}^8.$$

The fundamental operations on these bytes are addition and multiplication. The sum

$$c = a + b$$

of two bytes simply is the bitwise sum modulo 2 (exclusive-or, XOR):

$$c_i = a_i + b_i$$

for $0 \leq i \leq 7$. For example, if we take

$$(2.3) \quad a = (10011011), b = (11001101),$$

then

$$(2.4) \quad c = a + b = (01010110).$$

In hexadecimal notation, we have $a = 9B$, $b = CD$, and $c = 56$.

For multiplication, we first consider the byte a to represent the polynomial

$$a_7t^7 + a_6t^6 + \cdots + a_1t + a_0,$$

so that a as in (2.3) now represents

$$t^7 + t^4 + t^3 + t + 1 \in \mathbb{F}_2[t].$$

The product $a \cdot b$ of two bytes a and b is calculated by multiplying the two polynomials, giving a polynomial of degree not more than 14. For the two polynomials from (2.3), this is

$$p = t^{14} + t^{13} + t^{11} + t^{10} + t^8 + t^6 + t^5 + t^3 + t^2 + t + 1 \text{ in } \mathbb{Z}_2[t].$$

Since we work over $\mathbb{F}_2$, all coefficients are reduced modulo 2. More details are given in Section 15.1.

We have an obvious problem: the result has up to 15 bits, but we should come up with just one byte. Algebra provides an elegant solution: reduce modulo a polynomial of degree 8. Indeed, in AES we work in the finite field $\mathbb{F}_{256}$ defined by the irreducible polynomial

$$(2.5) \quad m = t^8 + t^4 + t^3 + t + 1 \in \mathbb{F}_2[t],$$

so that $a \bmod m \in \mathbb{F}_2[t]/(m) = \mathbb{F}_{2^8} = \mathbb{F}_{256}$. Now we divide p by m with remainder, obtaining

$$(2.6) \quad p = (t^6 + t^5 + t^3) \cdot m + (t^4 + t^2 + t + 1) \quad \text{in } \mathbb{F}_2[t],$$

$$9B \cdot CD = a \cdot b = (00010111) = 17 \quad \text{in } \mathbb{F}_{256}.$$

Thus we are back to degree at most 7, or 8 bits. Multiplication in $\mathbb{F}_{256}$ maps two bytes to one byte. But in SUBBYTES, we have only one byte as input. How can we use the arithmetic in $\mathbb{F}_{256}$? The answer is: inversion.

Since $\mathbb{F}_{256}$ is a field, every nonzero element $a \in \mathbb{F}_{256}^\times$ has an inverse $a^{-1} \in \mathbb{F}_{256}^\times$. This can be calculated by the Extended Euclidean Algorithm (Section 15.4). We extend this mapping to all of $\mathbb{F}_{256}$ by simply sending zero to itself:

$$(2.7) \quad \text{inv}(a) = \begin{cases} a^{-1} & \text{if } a \neq 00, \\ 00 & \text{if } a = 00, \end{cases}$$

where $00 = (00000000)$. This is called the *patched inverse*. In our example (2.3), the Extended Euclidean Algorithm produces

$$(2.8) \quad (t^7 + t^3) \cdot a + (t^6 + t^3 + t^2 + t + 1) \cdot m = 1 \text{ in } \mathbb{F}_2[t],$$

$\mathbb{F}_{256} = \mathbb{F}_{2^8} \ni a = a_7t^7 + a_6t^6 + a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0$,
with all $a_i \in \mathbb{F}_2 = \{0, 1\}$.

Representation: 8 bits for an element = 1 byte.

Addition: XOR, $(a + b)_i = a_i + b_i$.

Multiplication: as for polynomials modulo $t^8 + t^4 + t^3 + t + 1$.

Example $57 \cdot 83 = \text{C1}$:

$$\begin{aligned}
 & (t^6 + t^4 + t^2 + t + 1) \cdot (t^7 + t + 1) \\
 &= t^{13} + t^{11} + t^9 + t^8 + t^7 \\
 &\quad + t^7 + t^5 + t^3 + t^2 + t \\
 &\quad + t^6 + t^4 + t^2 + t + 1 \\
 &= t^{13} + t^{11} + t^9 + t^8 + t^6 + t^5 + t^4 + t^3 + 1 \text{ in } \mathbb{Z}_2[t] \\
 &= t^7 + t^6 + 1 \text{ in } \mathbb{Z}_2[t]/(t^8 + t^4 + t^3 + t + 1).
 \end{aligned}$$

Figure 2.4: The byte field $\mathbb{F}_{256}$.

as calculated in Example 15.20 (ii), so that indeed $\gcd(a, m) = 1$ in $\mathbb{F}_2[t]$, and

$$\text{inv}(a) = (10001000) = 88 \text{ in } \mathbb{F}_{256}.$$

In a surprising connecting with elliptic curves, we show in Section 6.4 that the patched inverse is nearly optimal in its resistance against a particular attack, namely, linear cryptanalysis.

AES also uses a similar, yet different, algebraic structure on bytes, namely the ring $R = \mathbb{F}_2[t]/(t^8+1)$. This is not a field, since $t^8+1 = (t+1)^8$ is not irreducible in $\mathbb{F}_2[t]$; see (15.4). Thus a byte a as in (2.2) now represents the element

$$a_7t^7 + a_6t^6 + a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0 \text{ in } R.$$

Addition is, again, just the bitwise addition (or XOR). Thus (2.4) is also valid in R . Multiplication of two such polynomials gives a polynomial of degree at most 14, whose remainder modulo $t^8 + 1$ has again degree at most 7. Reduction modulo $t^8 + 1$ is particularly easy, since it corresponds to just adding the lower and the upper halves of the polynomial, in the following sense. We split

$$c = c_1t^8 + c_0$$

into its upper and lower parts $c_1, c_0 \in \mathbb{F}_2[t]$ of degree at most 7. Then

$$c = c_1(t^8 + 1) + (c_1 + c_0) = c_1 + c_0 \text{ in } R.$$

To multiply the two bytes a and b of (2.3) in this new representation, we write their product as

$$(2.9) \quad p = (01101101) \cdot x^8 + (01101111) = 6D \cdot x^8 + 6F,$$

and then their product in the ring R is the sum of these two bytes:

$$9B \cdot CD = (10011011) \cdot (11001101) = (00000010) = 02 \text{ in } R.$$

In AES, actually only multiplication in R by the fixed polynomial

$$t_1 = (00011111) = 1F = t^4 + t^3 + t^2 + t + 1$$

is used, and only the polynomial

$$t_0 = (01100011) = 63 = t^6 + t^5 + t + 1$$

is added to others. Since t_1 is invertible modulo $t^8 + 1$, multiplication of bytes by t_1 corresponds to an invertible linear transformation over $\mathbb{F}_2$. For a byte b , the bits in

$$c = t_1 \cdot b + t_0$$

can also be described by the affine linear transformation

$$\begin{pmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}.$$

To sum up, SUBBYTES consists of applying to each byte a in the block individually the following steps:

$$(2.10) \quad \begin{aligned} a &\leftarrow \text{inv}(a) \quad \text{in } \mathbb{F}_{256}, \\ a &\leftarrow t_1 \cdot a \quad \text{in } R, \\ a &\leftarrow a + t_0. \end{aligned}$$

Its description involves some algebra, but SUBBYTES is most efficiently implemented by a 256-byte look-up table. It is the only nonlinear operation in AES and is sometimes called its S-box, in analogy with DES.

$$\begin{array}{c}
 \mathbb{F}_{256} \quad \longrightarrow \quad \mathbb{F}_{256} \quad \longrightarrow \quad \mathbb{F}_{256}, \\
 \\
 \boxed{S} : a \longmapsto \text{inv}(a) = \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} \longmapsto \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \\
 \\
 a \mapsto 05 \cdot a^{254} + 09 \cdot a^{253} + \text{F9} \cdot a^{251} + 25 \cdot a^{247} + \text{F4} \cdot a^{239} \\
 \quad + 01 \cdot a^{223} + \text{B5} \cdot a^{191} + 8\text{F} \cdot a^{127} + 63
 \end{array}$$

Figure 2.5: The SUBBYTES S-box.

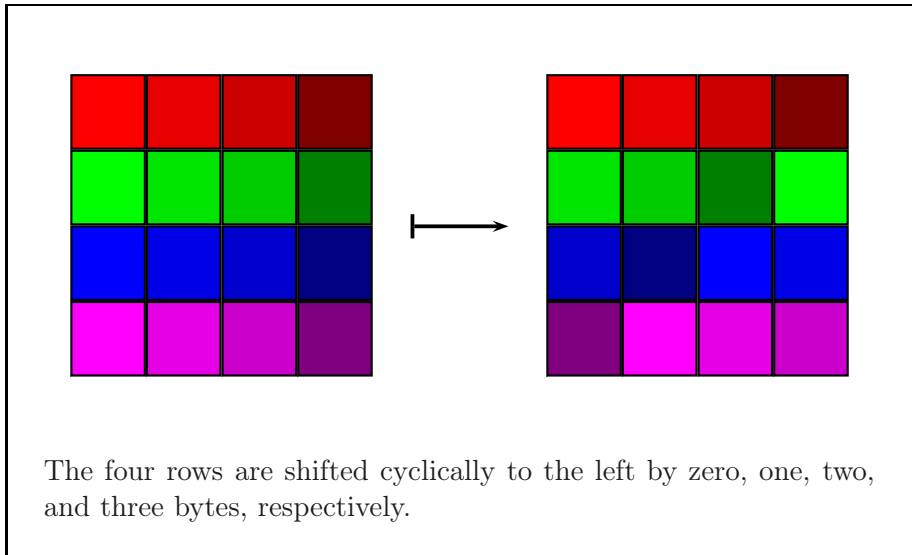


Figure 2.6: The SHIFTRows operation.

SHIFTRows. The operation SHIFTRows shifts each of the four rows cyclically to the left by 0, 1, 2, and 3 places, respectively. Thus SHIFTRows applied to the block (2.1) yields the array

$$(2.11) \quad \begin{array}{cccc} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{array} .$$

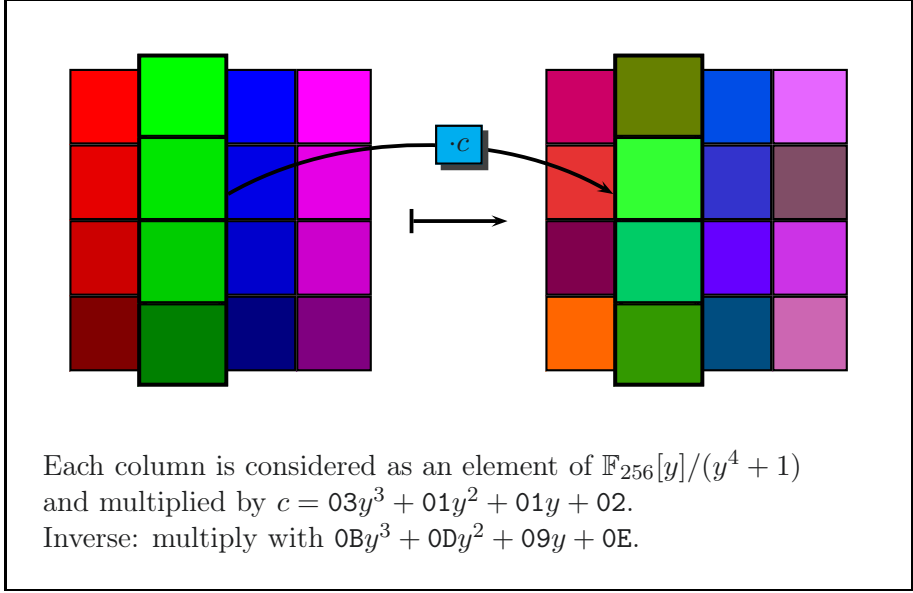


Figure 2.7: The MixColumns operation.

This is illustrated in [Figure 2.6](#).

MixColumns. Here we consider an array $a = (a_3, a_2, a_1, a_0)$ of four bytes a_3, a_2, a_1 , and a_0 as a polynomial

$$a_3s^3 + a_2s^2 + a_1s + a_0 \in \mathbb{F}_{256}[s]$$

of degree at most 3. Addition of such polynomials again corresponds to a bit-wise XOR. Multiplication gives a polynomial of degree at most 6 which is then decreased to degree at most 3 by reducing the result modulo $s^4 + 1 \in \mathbb{F}_{256}[s]$. Thus in effect we are working in the ring

$$S = \mathbb{F}_{256}[s]/(s^4 + 1)$$

with 256^4 elements. As $t^8 + 1$ above, $s^4 + 1 = (s + 1)^4$ is not irreducible in $\mathbb{F}_{256}[s]$, hence S is not a field. Reduction modulo $s^4 + 1$ is again particularly easy. If $b_0, b_1 \in \mathbb{F}_{256}[s]$ have degree at most 3, then

$$b_1s^4 + b_0 = b_1 + b_0 \text{ in } S.$$

In fact, this multiplication is only applied when one factor is the fixed polynomial

(2.12)

$$\begin{aligned} c &= (00000011) \cdot s^3 + (00000001) \cdot s^2 + (00000001) \cdot s + (00000010) \\ &= 03 \cdot s^3 + 01 \cdot s^2 + 01 \cdot s + 02 \end{aligned}$$

in $\mathbb{F}_{256}[s]$. The product of c with $a = (a_3, a_2, a_1, a_0)$ can also be described as the 4-byte word $b = (b_3, b_2, b_1, b_0)$ given by the matrix-vector product

$$\begin{pmatrix} b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} = \begin{pmatrix} 02 & 01 & 01 & 03 \\ 03 & 02 & 01 & 01 \\ 01 & 03 & 02 & 01 \\ 01 & 01 & 03 & 01 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}.$$

The operations on individual bytes are those in $\mathbb{F}_{256} = \mathbb{F}_2[t]/(m)$, as above. We take the example

$$\begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} \text{A0} \\ 80 \\ 01 \\ 02 \end{pmatrix}.$$

Then

$$\begin{aligned} b_3 &= 02 \cdot \text{A0} + 01 \cdot 80 + 01 \cdot 01 + 03 \cdot 02 \\ &= t \cdot (t^7 + t^5) + 1 \cdot t^7 + 1 \cdot 1 + (t + 1) \cdot t \\ &= t^8 + t^7 + t^6 + t^2 + t + 1. \end{aligned}$$

Since $t^8 = t^4 + t^3 + t + 1$ in $\mathbb{F}_{256}$, we have

$$b_3 = t^7 + t^6 + t^4 + t^3 + t^2 = (11011100) = \text{FC in } \mathbb{F}_{256}.$$

It is interesting to note the three roles that the byte 11011100 plays here: first as an element of $\mathbb{F}_{256}$, represented by a polynomial in $\mathbb{F}_2[t]$ of degree 7, then as an 8-bit string, and finally a 2-letter hexadecimal word. It is also the binary representation of the decimal integer 220. Even more interesting is the fact that we consider the byte as elements of different domains, such as in the inversion in $\mathbb{F}_{256}$ or in the second step in SUBBYTES, and then a multiplication on the same data may yield completely different results depending on the underlying domain. This versatility is another aspect of the unreasonable effectiveness of algebra in cryptography.

ADDROUNDKEY. The 128-bit block and a round key of the same size are added bitwise.

This concludes our general description of the four AES operations. In a software implementation, it is usually advantageous to replace calculations by table look-up as far as possible. Using a table of 4 kB, a round of AES can be executed with 16 table look-ups and 16 XORs of 32 bits.

AES evolved from earlier ciphers like *SHARK* (Rijmen *et al.* 1996) and *Square* (Daemen *et al.* 1997). Its design philosophy aimed at resistance against linear and differential cryptanalysis (Chapter 6) and high

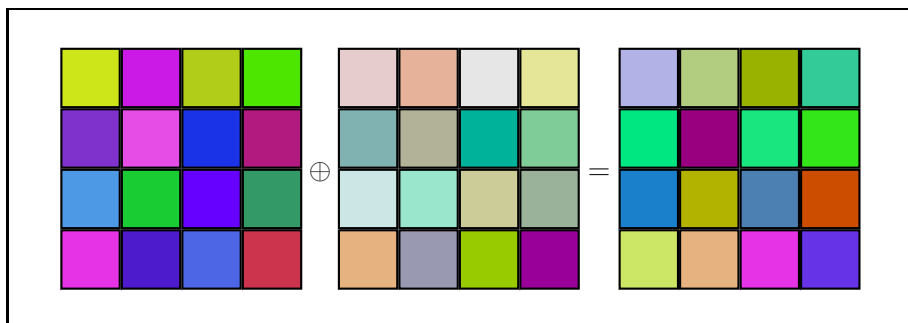


Figure 2.8: The ADDROUNDKEY operation.

throughput. The choices that this entailed are explained in Daemen & Rijmen (2002b). As examples, SUBBYTES using inversion was suggested in Nyberg (1994), and the modulus m is the first of 30 irreducible polynomials in Table C of Lidl & Niederreiter (1983). MIXCOLUMNS is based on matrices in which every square submatrix is nonsingular, a notion from the theory of error-correcting codes (MacWilliams & Sloane 1977, Chapter 11, Theorem 8). These have good diffusion properties. Namely, if F is a field, $M \in F^{n \times n}$ is MDS and $x, x^* \in F^n$ distinct, then the two vectors (x, Mx) and (x^*, Mx^*) in F^{2n} have Hamming distance at least $n + 1$, that is, the two vectors differ in at least $n + 1$ positions. The authors say convincingly: “We believe that the cipher structure does not offer enough degrees of freedom to hide a trapdoor.”

The omission of MIXCOLUMNS in the last round — or generally a final permutation step — is quite common, because it does not decrease security (ciphertext bits are just permuted in a publicly known way), but enables decryption with a similar structure; see Exercise 6.1.

As required in the AES competition, the algorithm is fast on a large variety of platforms. Software implementations can reach over 12 GB/sec.

Experts and the relevant standardization institutions consider AES secure. The strongest attack publicly known in 2015 (Bogdanov *et al.* 2011) has a cost of $2^{126.1}$ operations, compared to 2^{128} for exhaustive key search for AES-128. It does not constitute a serious threat to the security of AES. The most effective attacks are not on the system itself, but on specific implementations. Even the NSA seems stymied: “*Electronic codebooks, such as the Advanced Encryption Standard, are widely used and difficult to attack cryptanalytically. NSA has only a handful of in-house techniques.*”

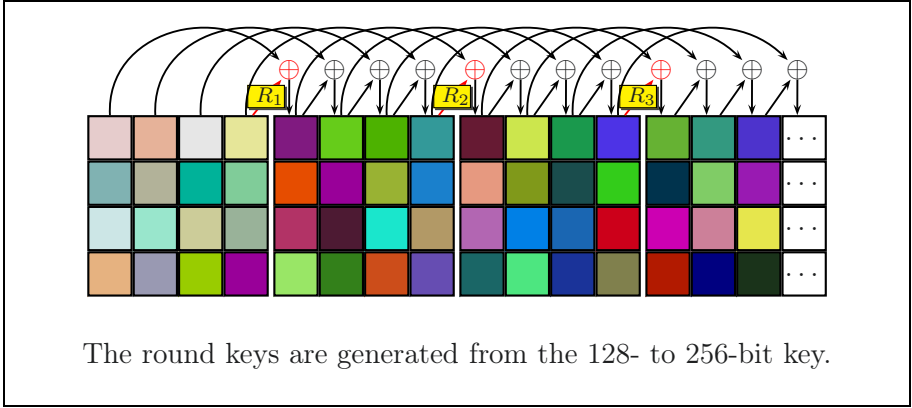


Figure 2.9: The AES key schedule.

2.3. The AES key schedule

AES allows keys of 128, 192, or 256 bits, which corresponds to ℓ_k many 32-bit words for $\ell_k = 4, 6$, or 8. The number ℓ_r of rounds after the initial one is given in [Table 2.10](#).

key length		ℓ_r rounds
in bits	in ℓ_k words	
128	4	10
192	6	12
256	8	14

Table 2.10: Key lengths and number of rounds in AES.

Each word has the format of a single column in an array like (2.11), but with 6 or 8 columns at the larger key lengths. In each round we need a *round key* array of ℓ_k words, and one more for the initial round. Thus we require a total of $\ell_k(\ell_r + 1)$ round key words.

We first explain this for 128-bit keys K , so that $\ell_k = 4$. The secret key K makes up the first four 4-byte words E_0, E_1, E_2, E_3 of the *extended key* $E_0, \dots, E_{4(\ell_r+1)-1}$, consisting of $4(\ell_r + 1)$ such words. We produce the others one by one, using the previous ones. Then our round keys consist of one block of four words after the other from the extended key.

For most indices $i \geq 4$, E_i is simply the sum in $\mathbb{Z}_2^{32}$ (bitwise XOR) of E_{i-1} and E_{i-4} :

$$E_i = E_{i-1} + E_{i-4}.$$

If i is a multiple of 4, then first a transformation is applied to E_{i-1} . Namely, the four bytes (a_3, a_2, a_1, a_0) of E_{i-1} are right-shifted cyclically

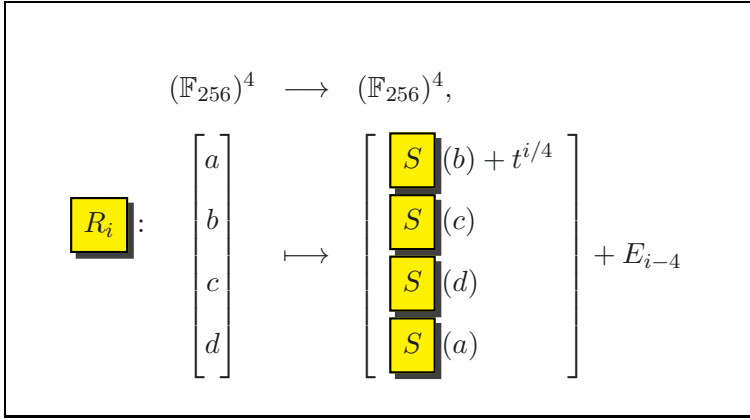


Figure 2.11: The nonlinear part of the key schedule for 128-bit keys and a multiple i of 4.

to give (a_0, a_3, a_2, a_1) . If we think of the word as an element a of $S = \mathbb{F}_{256}[s]/(s^4 + 1)$, this is simply multiplication by s^3 in S . Then SUBBYTES is applied to each byte individually, and a constant round word $c_{i/4}$ is added. These c_j are defined as

$$(2.13) \quad c_j = (0, 0, 0, t^{j-1}) = t^{j-1} \text{ in } S.$$

Thus c_j is constant in that it does not depend on the plaintext or the key. It is also a “constant” in S in that a general element of S is of the form $a_3s^3 + a_2s^2 + a_1s + a_0$, but for c_j we have $a_3 = a_2 = a_1 = 0$.

We recall that ℓ_k can take the values 4, 6, and 8. In the last case, a further transformation is applied. Namely, if i is 4 modulo 8, then E_{i-1} is replaced by SUBBYTES(E_{i-1}). Putting this together, the key expansion runs as follows for all three key lengths.

ALGORITHM 2.14. AES key expansion.

Input: A key $K_0, \dots, K_{\ell_k-1}$ consisting of ℓ_k many 4-byte words K_i .

Output: An extended key $E_0, \dots, E_{\ell_k(\ell_r+1)-1}$ consisting of $\ell_k(\ell_r+1)$ many 4-byte words E_i .

1. For i from 0 to $\ell_k - 1$ do
2. $E_i \leftarrow K_i$,
3. For i from ℓ_k to $\ell_k(\ell_r + 1)$ do steps 4 to 10
4. $L \leftarrow E_{i-1}$,
5. If ℓ_k divides i then
6. $c \leftarrow (0, 0, 0, t^{i/\ell_k-1})$
7. $L \leftarrow \text{SUBBYTES}(s^3 \cdot L) + c$,

$$\begin{array}{ll}
E_0 = K_0 = [00\ 00\ 00\ 00], & E_1 = K_1 = [00\ 00\ 00\ 00], \\
E_2 = K_2 = [00\ 00\ 00\ 00], & E_3 = K_3 = [00\ 00\ 00\ 00], \\
E_4 = K_4 = [00\ 00\ 00\ 00], & E_5 = K_5 = [00\ 00\ 00\ 00], \\
E_6 = [62\ 63\ 63\ 63], & E_7 = [62\ 63\ 63\ 63], \\
E_8 = [62\ 63\ 63\ 63], & E_9 = [62\ 63\ 63\ 63], \\
E_{10} = [62\ 63\ 63\ 63], & E_{11} = [62\ 63\ 63\ 63], \\
E_{12} = [9B\ 98\ 98\ C9], & E_{13} = [F9\ FB\ FB\ AA], \\
E_{14} = [9B\ 98\ 98\ C9], & E_{15} = [F9\ FB\ FB\ AA], \\
E_{16} = [9B\ 98\ 98\ C9], & E_{17} = [F9\ FB\ FB\ AA], \\
E_{18} = [90\ 97\ 34\ 50], & E_{19} = [69\ 6C\ CF\ FA], \\
E_{20} = [F2\ F4\ 57\ 33], & E_{21} = [0B\ 0F\ AC\ 99], \\
E_{22} = [A6\ A7\ 44\ 62], & E_{23} = [9F\ A2\ 37\ 0C], \quad \dots
\end{array}$$

Figure 2.12: Key expansion in AES-192 for the key consisting of all zeroes.

8. If $\ell_k = 8$ and i is 4 modulo 8 then
9. $L \leftarrow \text{SUBBYTES}(L),$
10. $E_i \leftarrow E_{i-\ell_k} + L.$

EXAMPLE 2.15. For a 192-bit key, messages consist of four words of four bytes each, and the cipher key of $\ell_k = 6$ such words $K_0, \dots, K_5$. From Table 2.10, we have $\ell_r = 12$ rounds, and thus need an extended key of $6 \cdot (12 + 1) = 78$ words. The first 24 are illustrated in Figure 2.12 for the key consisting of all zeroes. $\diamond$

We have now described the operations for one round and the key schedule. It remains to specify the *I/O* convention and how the whole system is put together.

Input and output are arrays of sixteen 8-bit bytes as in (2.1), 128 bits in total. The conversion to and from a 4×4 block is columnwise, so that the array $a_{00}, a_{10}, a_{20}, a_{30}, a_{01}, \dots$ corresponds to the block (2.1). In the same manner, the cipher key of $4\ell_k$ bytes is fed into the first words $E_0, \dots, E_{\ell_k-1}$ of the extended key.

Finally, the whole system operates as follows, as in Figure 2.1:

1. initial round: ADDROUNDKEY

2. for i from 1 to $\ell_r - 1$ do

- SUBBYTES
- SHIFTRows
- MIXCOLUMNS
- ADDROUNDKEY

3. final round:

- SUBBYTES
- SHIFTRows
- ADDROUNDKEY

The round keys are taken consecutively from the extended key.

2.4. Asymmetric vs. symmetric cryptography

In a *symmetric* (or *secret key*) *cryptosystem*, the same key is used for encryption and for decryption. All cryptographic methods were of this type until the 1970s, and so is AES. But then Diffie & Hellman made their revolutionary proposal for *asymmetric* (or *public key*) *cryptosystems*: each player uses a *public key* and a *secret key*. The public key of, say, BOB is used by everybody to encrypt messages to BOB. With his secret key, BOB can easily decrypt these messages, but without it, nobody should be able to do this.

	Cryptosystems	
	symmetric	asymmetric
examples	one-time pad, Caesar, DES, AES	RSA, Diffie-Hellman, ElGamal
speed	+	—
authentication	+	—
key exchange	—	+

Table 2.13: Symmetric vs. asymmetric cryptosystems.

At the current state of the art, both types of systems have their pros and cons. [Table 2.13](#) lists some systems, to be explained later. The basic advantage of some symmetric systems is their speed, for example, over 12 GB/sec for AES. Their disadvantage is that a previous exchange of the common key is necessary; see Chapter I for drastic illustrations of the ensuing problems, sometimes costing the lives of thousands. A further advantage of symmetric cryptosystems is that the recipient can feel secure

about the sender's identity, while in the asymmetric situation additional measures have to be taken against impostors; identification schemes and authentication are discussed in Chapter 10. But for modern cryptography, "symmetric vs. asymmetric" is not a competition, rather both sides win! We use asymmetric systems to share the common keys in a symmetric system, and then the latter for high-throughput communication.

Wonderful! Three examples illustrate this: we have already seen the symmetric system AES, next comes the asymmetric Rivest, Shamir and Adleman (RSA) system, and finally the asymmetric Diffie-Hellman key exchange.

2.5. The RSA cryptosystem

As the first realization of the abstract public key cryptosystem model suggested by Diffie & Hellman (1976), Rivest, Shamir & Adleman (1977) invented the *Rivest-Shamir-Adleman cryptosystem* (RSA).

We follow the long-standing tradition of calling the two players ALICE and BOB. Our scenario is that ALICE wants to send a message to BOB that he should be able to read, but nobody else. To this end, BOB generates a *secret key* sk and a *public key* pk . Anybody can read pk ; imagine it is posted on the internet or in some large database. But BOB guards sk carefully as his secret. ALICE uses pk to encrypt her message for BOB. BOB uses sk to decrypt it. In a symmetric cryptosystem like AES, the encryption and decryption keys are (essentially) the same, but here pk and sk are different, and in fact sk cannot be computed easily from pk (hopefully).

The messages to be sent may be text, digitized pictures, movies, or correct, data or program files, etc. But we assume here and always in the future that the messages have been converted into some standard form, say into a (possibly very long) string of bits 0 and 1. How to perform this conversion best depends on the type of data. For text, a common way is to use ASCII or extended ASCII encoding of letters into 7-bit or 8-bit strings, respectively. In practice, RSA is mainly used to transmit a secret key, and the (short) message is derived from the key. Digital signatures can also be produced via RSA (Section 8.1).

Suppose that ALICE wants to send a (long) string of bits. There is a *security parameter* n to be explained in a minute. ALICE splits her string into blocks of $n - 1$ bits each, and transmits each block separately. So we now explain how to transmit a single block $(x_0, \dots, x_{n-2})$ of $n - 1$ bits in the RSA system. We interpret the block as the binary representation of the natural number $x = \sum_{0 \leq i \leq n-2} x_i 2^i$. This number shall be transmitted.

The idea is the following. BOB chooses two prime numbers p and q at random with $n/2$ bits each, and so that their product $N = p \cdot q$ has n bits. He also chooses some random integer e with $1 \leq e < N$ and $\gcd(e, (p-1)(q-1)) = 1$. BOB's public key is $\mathbf{pk} = (N, e)$. ALICE looks it up and sends the encryption $y = x^e$ in $\mathbb{Z}_N$ to BOB, that is, the remainder of x^e on division by N . The magic now is that BOB can recover x from ALICE's message with the help of his private information derived from (p, q) . Here is the system described in full. The required algebraic terminology is explained in the *computer algebra toolbox* of Chapter 15. Throughout this text, the notation $a \xleftarrow{\square} A$ denotes a uniformly random choice of a from the finite set A , so that for any $b \in A$ the random variable a assumes the value b with probability $1/\#A$.

CRYPTOSYSTEM 2.16. RSA.

Key Generation keygen .

Input: Security parameter n .

Output: secret key $\mathbf{sk}$ and public key $\mathbf{pk}$.

1. Choose two distinct primes p and q at random with $2^{(n-1)/2} < p, q < 2^{n/2}$.
2. $N \leftarrow p \cdot q, L \leftarrow (p-1)(q-1)$. [N is an n -bit number, and $L = \varphi(N)$ is the value of Euler's φ function.]
3. Choose $e \xleftarrow{\square} \{2, \dots, L-2\}$ at random, coprime to L .
4. Calculate the inverse d of e in $\mathbb{Z}_L$.
5. Publish the public key $\mathbf{pk} = (N, e)$ and keep $\mathbf{sk} = (N, d)$ as the secret key.

Encryption enc .

Input: $x \in \mathbb{Z}_N, \mathbf{pk} = (N, e)$.

Output: $\text{enc}_{\mathbf{pk}}(x) \in \mathbb{Z}_N$.

6. $y \leftarrow x^e$ in $\mathbb{Z}_N$.
7. Return $\text{enc}_{\mathbf{pk}}(x) = y$.

Decryption dec .

Input: $y \in \mathbb{Z}_N, \mathbf{sk} = (N, d)$.

Output: $\text{dec}_{\mathbf{sk}}(y) \in \mathbb{Z}_N$.

8. $z \leftarrow y^d$ in $\mathbb{Z}_N$.
9. Return $\text{dec}_{\mathbf{sk}}(y) = z$.

After the key generation, ALICE may forget p , q , and $\varphi(N)$, and erase them on her computer. N is included in the secret key only for symmetry. Here is a simple example.

EXAMPLE 2.17. We take $n = 6$. Literally, we would be looking for primes between 6 and 7, but at such small values we have to be a bit more liberal and choose $p = 5$ and $q = 11$. Thus $N = 55$ is a 6-bit number, and $L = 40$. We choose $e = 13$. Using the Extended Euclidean Algorithm 15.4, we find in a single step that $-3 \cdot 13 + 40 = 1$, so that $d = e^{-1} = -3 = 37$ in $\mathbb{Z}_{40}$. Thus BOB publishes his public key $\text{pk} = (55, 13)$ and keeps his secret key $\text{sk} = (55, 37)$. This finishes the key generation.

Now ALICE wants to send a message to BOB, say $x = 7$. Thus she has to calculate $y = x^e = 7^{13}$ in $\mathbb{Z}_{55}$. The obvious way to do this is to compute the integer 7^{13} and take its remainder modulo 55. This would be quite cumbersome here, and utterly infeasible at practical values of the security parameter n , where x^e would have more bits than there are elementary particles in the universe. But there is an easy way out: the repeated squaring Algorithm 15.48 uses fewer than $2m$ operations in $\mathbb{Z}_N$ for an m -bit exponent. This is illustrated in Figure 2.14. Its first column

instruction	value	exp	bit	in $\mathbb{Z}_{55}$
$y_0 \leftarrow x$	x	1	1	7
$y_1 \leftarrow y_0^2$	x^2	10		49
$y_2 \leftarrow y_1 \cdot y_0$	x^3	11	1	13
$y_3 \leftarrow y_2^2$	x^6	110	0	4
$y_4 \leftarrow y_3^2$	x^{12}	1100		16
$y_5 \leftarrow y_4 \cdot y_0$	x^{13}	1101	1	2

Figure 2.14: Computing x^{13} .

shows the instruction, the second one the value as a power of x , the third column the binary representation of the exponent in column two, the fourth column the corresponding bit in the binary representation 1101 of 13, and the last column the value for $x = 7$ in $\mathbb{Z}_{55}$. By a squaring, a 0 is appended to the right of the exponent's representation, and a subsequent multiplication by $y_0 = x$ turns this into a 1. This multiplication is done if and only if the corresponding bit is 1. Then the representation of the exponent is an initial segment of the representation of 13; this is the case just above the horizontal lines. All intermediate results are taken modulo

$N = 55$ and never get larger than N .

Now ALICE has done her share of calculation and sends $y = \text{enc}_{55,13}(7) = 2$ to BOB. He decrypts in the same way, using the binary representation 100101 of 37, and computes the following sequence of results in $\mathbb{Z}_{55}$:

$$2, 4, 16, 36, 17, 14, 31, 7.$$

Thus $\text{dec}_{55,37}(2) = z = 2^{37} = 7$ in $\mathbb{Z}_{55}$ and indeed, this is the message that ALICE wanted to send to BOB. $\diamond$

Because of its importance, we assemble the notation of Figure 2.15 for RSA, which will be used repeatedly in this text. The length of the public key (N, e) is $2n$, and also for the secret key.

security parameter n ,
 distinct random primes p and q of at least $n/2$ bits,
 and so that $N = pq$ has n bits,
 $L = \varphi(N) = (p-1)(q-1)$,
 $e, d \in \mathbb{Z}_L$ with $ed = 1$ in $\mathbb{Z}_L$,
 plaintext x , ciphertext y , decryption z , all in $\mathbb{Z}_N$,
 $y = x^e$, $z = y^d$.

Figure 2.15: The RSA notation.

We have to address several questions.

Correctness: Is $z = x$?

Efficiency: How to calculate fast ...

- ... large primes at random?
- ... d from e ?
- ... powers modulo N ? This has to be done for each message, and speed is even more a concern than for the previous two points.

Security: Suppose that an eavesdropper—traditionally called EVE—listens to the communications between ALICE and BOB. Thus EVE knows y and, of course, (N, e) , and she would like to compute x . In fact, x is uniquely determined! But how long does it take to calculate this value? Is this difficult enough to provide security?

Some of these questions are addressed in Section 3.1. The tools for an efficient implementation of RSA will only be discussed there, but we state

a precise version of *efficient* now. It is the fundamental notion of polynomial time, which will be familiar to any student of computer science. In the key generation, we have to make random choices and thus require a *probabilistic algorithm*. This is the standard type of algorithm in this text. Its *expected runtime* for a fixed input is obtained by averaging over the algorithm's internal random choices. Thus a bound on the runtime has to hold for all inputs; there is no averaging over the inputs. For RSA encryption and decryption, no random choices are required. This special type of probabilistic algorithms is called *deterministic*. We recall the notions of “easy” and “hard” from Definition 15.29.

For the cryptographic protocols discussed in this text, efficiency always means that the operations performed by the legitimate players are easy. On the other hand, we want an adversary's problem to be hard. This leads to the security notions of Chapter 9. It turns out that RSA is not secure in the most demanding sense; see Example 9.22. For practical purposes, the current recommendation is to use a security parameter $n \geq 3000$; see Table 5.12.

Cryptographers at the British government agency CESG (Communications-Electronics Security Group) had invented, starting in 1970, many ingredients of public-key cryptography and the RSA and ElGamal cryptosystems before they were published in the open literature.

2.6. The Diffie-Hellman key exchange

For symmetric cryptosystems like AES, the shared secret key has to be chosen and communicated between all involved (and authorized) parties. In practice this agreement is a crucial difficulty in secret-key cryptography. A historical example is in Chapter I on the Zimmermann telegram. During the First World War, the Germans were cut off from their overseas embassies. They suspected that their old codes were broken, and sent a new code to Washington by U-boat. But it could not be forwarded to Mexico. An important secret message was sent safely to Washington in the new code, but then had to be transmitted in the old code to Mexico—and was duly broken. A safe public way of exchanging secret keys might have prevented this.

Can two players agree upon a shared secret key (for a symmetric cryptosystem) when they are forced to use an insecure channel? Stop here for a moment and convince yourself that this is impossible. The—somewhat surprising—correct answer is: yes, they can!

We cannot simply transmit the shared secret key, because the channel is insecure. Instead, each player sends a disguised version of the key from which the other player can assemble the shared key, but an eavesdropper

cannot. To achieve this, Diffie and Hellman proposed to use a prime p and work with the integers that are not divisible by p , and their multiplicative properties. These numbers form the group $\mathbb{Z}_p^\times$ of *units* modulo p . There are $p - 1$ of them, the product of two of them is again not divisible by p , and any of them has a multiplicative inverse. This inverse can be computed via Euclid's algorithm. Furthermore, this group $G = \mathbb{Z}_p^\times$ is *cyclic*, meaning that there is an element $g \in G$, called a *generator*, whose powers comprise all of $G = \{1 = g^0, g, g^2, g^3, \dots, g^{p-2}\}$. The details are discussed in the *computer algebra toolbox* of Chapter 15.

It turns out that the latter properties are also sufficient. Chapter 4 is dedicated to cryptography in groups, and we now describe the Diffie-Hellman key exchange in this general setting. So we have a finite cyclic group $G = \langle g \rangle$ and a generator $g \in G$. The group G , the generator g and all transmitted messages are public, so they are known not only to ALICE and BOB but also to any unauthorized eavesdropper EVE. The *order* of G , that is, the number of elements of G , is denoted as $d = \#G$. There is a global security parameter n , and d is an n -bit number. Then the elements of G can be represented by n -bit strings. A basic requirement is that the description of G can be provided with polynomially in n many bits, and the group operations can be executed on the representations of the elements of G in polynomially in n many bit operations. The latter two requirements are usually satisfied in an obvious way.

Some examples for G are:

1. the multiplicative group $G = \mathbb{Z}_p^\times$ of units modulo a prime p ,
2. the multiplicative group $G = \mathbb{F}_q^\times$ of a finite field $\mathbb{F}_q$,
3. cyclic subgroups of elliptic curves (Chapter 5) over finite fields.

We use the following example for illustration.

EXAMPLE 2.18. Let $G = \mathbb{Z}_{2579}^\times$ be the multiplicative group of units modulo the prime number 2579. Because $d = \varphi(2579) = 2578 = 2 \cdot 1289$ with 2 and 1289 prime, and $2^2 = 4$ and $2^{1289} = -1$ in $\mathbb{Z}_{2579}^\times$ are both different from 1, we conclude from Corollary 15.59 that $g = 2 \in G$ is a generator of G . The description of G consists of a few bits saying that “ G is of the form $\mathbb{Z}_p^\times$ ”, for example the ASCII encoding of the phrase in quotes plus the binary representation of p , 101000010011 in this case. G is identified with a subset of $\{0, \dots, p - 1\}$, and the binary representation of these integers provides the representation of the elements of G . The tools for efficient arithmetic in Chapter 15 show that the group operations can be performed at cost quadratic (hence polynomial) in n , with $n = 12$ in this example. $\diamond$

We now describe the *Diffie-Hellman key exchange*. First ALICE and BOB individually choose a secret key a and b , respectively, randomly in $\mathbb{Z}_d$. Then they publish their public keys $A = g^a$ and $B = g^b$, respectively, maybe by posting them in a large internet database. After this asymmetric set-up, they can now both compute, maybe for use in a symmetric cryptosystem, the shared secret key

$$g^{ab} = B^a = A^b,$$

since

$$(2.19) \quad B^a = (g^b)^a = g^{ab} = (g^a)^b = A^b.$$

Figure 2.16 gives a graphical representation of the system.

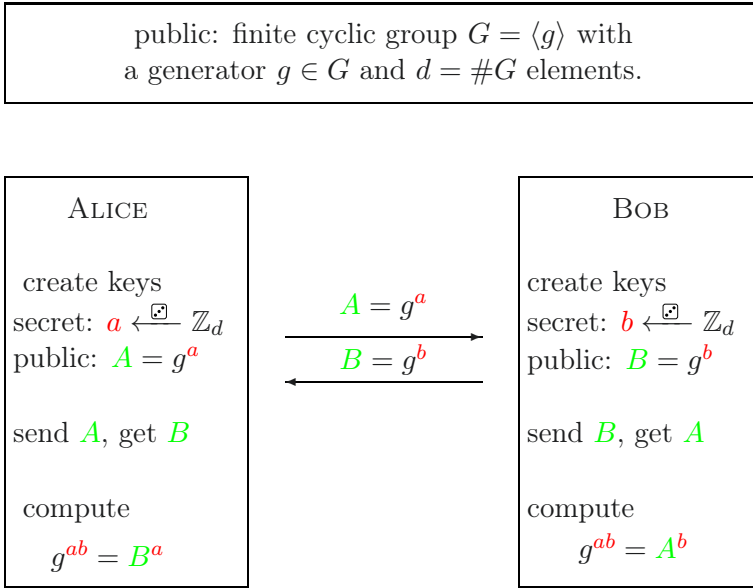


Figure 2.16: Diffie-Hellman key exchange.

Here is a formal description of the protocol.

PROTOCOL 2.20. Diffie-Hellman key exchange.

Set-up.

Input: security parameter n .

Output: G , g and d as below.

1. Determine a description of a finite cyclic group $G = \langle g \rangle$ with $d = \#G$ elements and a generator g of G , where d is an n -bit integer.

Key exchange.

2. ALICE chooses her secret key $a \xleftarrow{\mathbb{Q}} \mathbb{Z}_d$. She computes her public key $A \leftarrow g^a \in G$.
3. BOB chooses his secret key $b \xleftarrow{\mathbb{Q}} \mathbb{Z}_d$. He computes his public key $B \leftarrow g^b \in G$.
4. ALICE and BOB exchange their public keys A and B .
5. ALICE computes the shared secret key $k_A = B^a$.
6. BOB computes the shared secret key $k_B = A^b$.

We continue our little example.

EXAMPLE 2.18 CONTINUED. Recall that $G = \mathbb{Z}_{2579}^\times$ and $g = 2 \in G$.

2. ALICE chooses her secret key $a = 765$ and computes her public key $A = 2^{765} = 949$ in G .
3. BOB chooses his secret key $b = 853$ and computes his public key $B = 2^{853} = 435$ in G .
4. ALICE and BOB exchange their public keys $A = 949$ and $B = 435$.
5. ALICE computes the shared secret key $k_A = B^{765} = 2424$ in G .
6. BOB computes the shared secret key $k_B = A^{853} = 2424$ in G .

And lo and behold, the system works not only in general, but also in this particular case: ALICE and BOB share the key $k_A = k_B$. $\diamond$

In order to analyze this protocol, we should answer the three usual questions:

Correctness: Do ALICE and BOB get the same shared key?

Efficiency: Can ALICE and BOB do their computations efficiently?

Security : Is it hard for EVE to get information on the shared key?

The correctness is shown in (2.19), and ALICE and BOB indeed possess the same shared key $k_A = k_B$.

Concerning efficiency, the most costly operation is exponentiation in G . Figure 2.14 presents an example for the efficient algorithm of *repeated squaring* for computing $g^e \in G$, given $g \in G$ and $e \in \mathbb{N}$; see also Section 15.13. We may assume that $0 \leq e < d$ by Corollary 15.57 and then it uses $O(n)$ multiplications in G .

In order to discuss security, we slip into the rôle of EVE who works on the following task.

DIFFIE-HELLMAN PROBLEM 2.21 (DH_G). Given a group $G = \langle g \rangle$ of order d and A and B in G , compute $C \in G$ so that

$$A = g^a, B = g^b, C = g^{ab},$$

for some a and b in $\mathbb{Z}_d$.

DEFINITION 2.22. Let $G = \langle g \rangle$ be a group of order d , let $a, b, c \in \mathbb{Z}_d$ and $A = g^a$, $B = g^b$, and $C = g^c$ in G . If $ab = c$ in $\mathbb{Z}_d$, then (A, B, C) is a Diffie-Hellman triple.

Thus DH_G asks, for given $A, B \in G$, to find some $C \in G$ so that (A, B, C) is a Diffie-Hellman triple. There is also the following, possibly easier, version of DH_G .

DECISIONAL DIFFIE-HELLMAN PROBLEM 2.23 (DDH_G). Given a group $G = \langle g \rangle$ of order d and A, B, C in G , decide whether (A, B, C) is a Diffie-Hellman triple.

If DH_G can be solved efficiently, then so can DDH_G . But the reverse is conjectured to be false in general.

Instead of sending the public versions $A = g^a$ and $B = g^b$ of a and b , ALICE and BOB might of course send the secret values a and b . But then they have no secret hidden from EVE. More generally, if EVE can compute a from A , g , and G , then she only has to compute $B^a \in G$ to get the common key, just like ALICE does. We have already noted that raising to a power can be done efficiently. The “inverse” task amounts to the following.

DISCRETE LOGARITHM PROBLEM 2.24 (DL_G). Let G be a group, and $g, x \in G$ two group elements. Decide whether x is in the subgroup $\langle g \rangle \subseteq G$ generated by g , and if so, compute some $a \in \mathbb{Z}$ with $g^a = x$.

A refined version of this problem is described in Definition 4.2. The Discrete Logarithm Problem 2.24 is of great importance in cryptography and occupies a prominent place in this book. In Chapter 4, we discuss various ways to solve it, and also the “generic model” of computation which says that unless you have a special insight into your group G , the discrete log problem is hard. Chapter 5 is devoted to a type of groups—elliptic curves—for which no-one seems to have the required “special insight” so far, and which are currently popular in cryptography. As noted above, if EVE can solve the discrete logarithm problem efficiently, then she also can break the Diffie-Hellman Problem fast. The reverse $\text{DL}_G \leq_p \text{DH}_G$ is also true for “most” groups G under some plausible assumptions.

We have discussed security under the assumption that EVE attacks our protocol in a passive way so far. The bad guy only listens to the transmission and tries to compute results without disturbing the communication between ALICE and BOB. But can we assume EVE to be passive? This leads to a more general question on security: Can EVE get secret information without solving the Diffie-Hellman Problem 2.21?

One attack that works in this case is the *woman-in-the-middle attack*: EVE pretends to be ALICE when talking to BOB and to be BOB when exchanging data with ALICE. Both legitimate players think they have the right partner. And EVE can act in the expectedly evil way by generating her own part of the used shared key(s). This is illustrated in Figure 2.17.

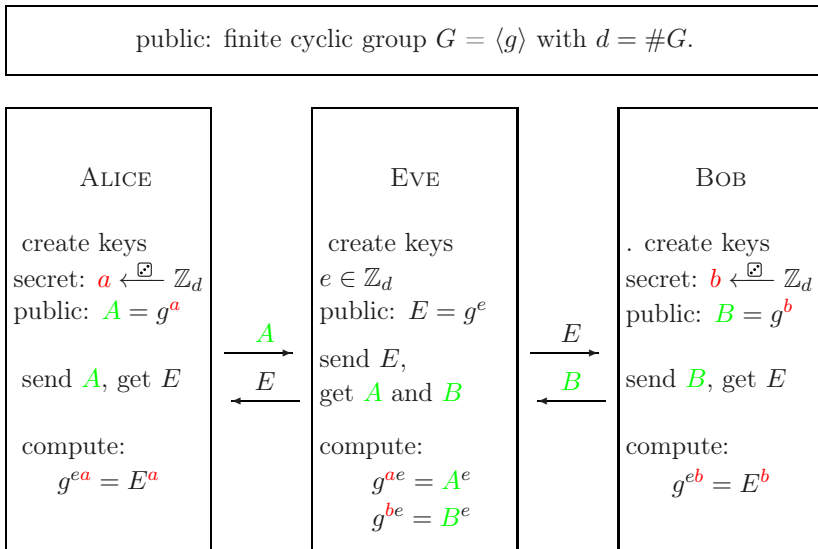


Figure 2.17: Woman-in-the-middle attack on the Diffie-Hellman key exchange.

BOB now believes that his shared key with ALICE is g^{be} , but the unsuspecting ALICE knows nothing about this; only EVE does. And similarly the other way around. In effect, EVE can read all information encrypted with the supposedly “shared” keys g^{eb} or g^{ea} , respectively. She can decrypt any message and re-encrypt it with the other key to hide the attack. BOB and ALICE may not recognize this active attack until EVE stops “translating” the encrypted messages.

This is clearly a killer for the system, and ALICE and BOB have to thwart this attack. Of course, they can simply compare whether the “shared” keys are identical. If they are not, they can be sure that someone has been messing around. But how can they be sure to compare to the

right key without identifying their partner and without disclosing their common key? How does BOB tell ALICE from EVE? They have to find a way to identify each other. In the “non-cryptographic world”, we know each other’s voices or faces, and we can use identity cards or passports when introducing ourselves to a stranger. ALICE and BOB have to use similar tools within the protocol. Such electronic tools do indeed exist, and we discuss them in Chapter 10.

2.7. Block ciphers

We now give a more formal description of encryption. Some theoretical framework is necessary to put examples into perspective. It remains bloodless until infused with specific instances; we have already seen AES and RSA, and more examples will follow.

DEFINITION 2.25. *A block encryption scheme (block cipher) S consists of four probabilistic algorithms $S = (\text{set-up}, \text{keygen}, \text{enc}, \text{dec})$. We have a security parameter $n \in \mathbb{N}$ and for each n a key space K_n , a plaintext space P_n , and a ciphertext space C_n , all of which are finite and publicly known. Often they consist of all (or most) bit strings of a certain length. The value n may be restricted to some set of integers. We let $K = \bigcup_{n \in \mathbb{N}} K_n$.*

set-up: on input n , it outputs descriptions of K_n, P_n, C_n ,
and of the encryption scheme used.

keygen: $\mathbb{N} \longrightarrow K^2$,
keygen(1^n) = (sk, pk) $\in K_n^2$,

and for each n

enc: $K_n \times P_n \longrightarrow C_n$,
enc_{pk}(x) = y ,
dec: $K_n \times C_n \longrightarrow P_n$,
dec_{sk}(y) = z .

The options for the descriptions are determined by the larger system that uses S . For example, set-up(3000) = (3000, “RSA”) might mean that 3000-bit RSA is used, and $P_n = \{0, 1\}^n$ and $C_n = \{0, 1\}^{n+1}$ consist of the binary representations of numbers modulo the RSA modulus. The input to keygen is the integer n , represented in unary as a string of n ones. The values sk, pk, x , and y are arbitrary elements of K_n , K_n , P_n , and C_n , respectively, and those in dec need not be related to those in enc. The “probabilistic encryption” enc is not a function in the usual sense,

but rather a “function ensemble”, that is, a probabilistic distribution on the functions $K_n \times P_n \rightarrow C_n$. The distribution is given by enc ’s internal randomization. We require three properties of S .

Correctness: The decryption of an encrypted plaintext equals the plaintext. That is, for all $n \in \mathbb{N}$ and $x \in P_n$, if $\text{keygen}(1^n)$ outputs (sk, pk) , then

$$\text{dec}_{\text{sk}}(\text{enc}_{\text{pk}}(x)) = x.$$

Efficiency: The three “function ensembles” keygen , enc , and dec can be computed by probabilistic polynomial-time algorithms, where the algorithm for dec is further restricted to be deterministic.

Security: This is the most nontrivial requirement, and various aspects of it will be discussed later; see Chapter 9.

This text sometimes uses appropriate modifications of these notions. For the security, the resources at an adversary’s disposal may be of different types. If neither pk nor sk is allowed to be known, we might as well replace both of them by the pair (sk, pk) , so that $\text{sk} = \text{pk}$ afterwards.

DEFINITION 2.26. *In a symmetric (or secret key) encryption scheme, we have $\text{sk} = \text{pk}$. In an asymmetric (or public key) encryption scheme, pk is known to an adversary.*

An early block cipher is the *Caesar cipher* (Section A.3) in which the encryption replaces each letter by another one, three positions later in the alphabet. This is a secret key encryption scheme of block size 1, with $P_1 = C_1 = K_1$ being the alphabet (which here replaces $\{0, 1\}$). If we allow a shift by an arbitrary number of positions, then this number is the secret key sk . In Caesar’s system, we have $\text{sk} = 3$ and $\text{enc}_3(\text{crypto}) = \text{fubswr}$ (using the English alphabet). Decryption consists of shifting back each letter by three positions, and $\text{dec}_3(\text{fubswr}) = \text{crypto}$.

In a fixed system like AES, there is no security parameter, and we only have the three key lengths of 128, 192, and 256 bits. Then $K_n = \{0, 1\}^n$ and $P_n = C_n = \{0, 1\}^{128}$ for $n \in \{128, 192, 256\}$, since all plaintexts and ciphertexts consist of 128 bits. All other K_n , P_n and C_n are empty. Say, we consider $n = 256$. Then Definition 2.25 simplifies as follows:

$$\begin{aligned} \text{keygen}: \{256\} &\rightarrow K_{256}, \\ \text{keygen}(256) &= \text{sk} \in K_{256}, \\ \text{enc}_{\text{sk}}: K_{256} \times P_{256} &\rightarrow C_{256}, \\ \text{dec}_{\text{sk}}: K_{256} \times C_{256} &\rightarrow P_{256}. \end{aligned}$$

The last two requirements stated above have to be modified. Efficiency now means fast enough on current computers, and security means to resist the known attacks; see Chapter 6.

2.8. Stream ciphers and modes of operation

In a *stream cipher*, one has an input stream whose length is not known in advance, and encodes the message as it flows by. As an example, we can single out the *one-time pad*, in which the ciphertext bits are obtained by binary addition (XOR) of the plaintext and the key bits:

$$y_i = x_i \oplus k_i,$$

where y_i , x_i , and k_i denote the i th bit of the ciphertext, plaintext, and key, respectively. This system is *perfectly (information-theoretically) secure* provided the key is “truly random” (Section 9.4). Such a scheme is difficult to implement for long plaintexts, since the key is as long as the plaintext. We mention a related system, the synchronous additive stream cipher, in Section B.1. These ciphers played an important role in the history of cryptography, associated with the names of Trithemius, Vigenère, and Vernam, and include the German *Siemens Geheimschreiber* and the British *Typex* from the Second World War. Such a stream cipher usually produces a stream of pseudorandom bits (Chapter 11) and uses it like the one-time pad above.

In this text we discuss mostly *block ciphers*, encrypting blocks of some length n . Examples so far are AES (with $n = 128$) and RSA (with variable n). However, we have to consider how to encrypt long messages with one of our block ciphers (keygen, enc, dec) of fixed message length n . There are several ways of doing this. First, we might simply chop the message into blocks of length n and encrypt each block separately. This is called the *Electronic Codebook* (ECB) and actually not a good idea. A passive adversary, intercepting many encryptions, would then know which data are identical to those in other messages.

In order to mitigate this problem, we can chain the encryptions together, so that the encryption of each block depends on the previous blocks. We split our input, as it streams by, into blocks $x_0, x_1, x_2, \dots$, each of n bits. For $i = 0, 1, 2, \dots$, we calculate some n -bit z_i which is transmitted. In the *cipher block chaining* mode (CBC), z_i is obtained by encrypting the sum (bitwise XOR) of x_i and z_{i-1} . (With an initial value v for z_{-1} ; often one takes $v = 0$, but this is less secure than a random v .) In the *cipher feedback* mode, z_i is the sum of x_{i-1} and $\text{enc}_K(x_i)$, again with an initial value for x_{-1} . The same key is used for all encryptions. One might also consider transmitting $z_i = \text{enc}_K(x_i) \oplus \text{enc}_K(z_{i-1})$, or use the

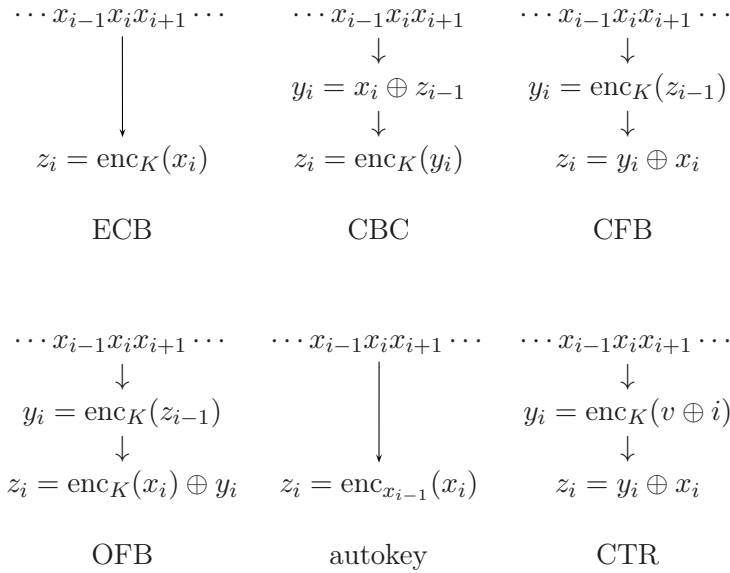


Figure 2.18: Modes of operation for block ciphers.

plaintext itself as key, assuming a key of length n : $z_i = \text{enc}_{x_{i-1}}(x_i)$. The former is the output feedback mode (OFB) and the latter is called *autokey* and was invented by Vigenère; see Section C.4. The *counter mode* (CTR) uses an *initial vector* (or *nonce*) v and a counter which, in the simplest case, is increased by one for each new block. The i th ciphertext then is $z_i = \text{enc}_K(v \oplus c) \oplus x_i$, where c is the binary representation of the counter, padded with leading zeroes to the appropriate length. CBC and OFB suffer from the problem of error propagation: an error in one encryption may contaminate subsequent blocks; see Exercise 2.10. Figure 2.18 summarizes these six modes of operation.

2.9. Secret sharing

The basic task for this section is the following: we want to distribute shares of a secret among n players so that together they can reconstruct the secret, but no proper subset of the players can. You may think of sharing access to your bowling club account with your friends, or of other people, possibly not your friends, sharing access to a nuclear first strike capability.

Shamir (1979) presents an elegant solution on just two pages. We identify possible secrets with elements of the finite field $\mathbb{F}_p = \mathbb{Z}_p$ for an appropriate prime p . Some bank cards for *Automatic Teller Machine* (ATM) access have four-digit decimal numbers as their secret *Personal Identifi-*

cation Number (PIN) codes. To distribute such a secret PIN to n players, we may take the smallest prime $p = 10\,007$ with five decimal digits. Then we choose $2n - 1$ random elements $f_1, \dots, f_{n-1}, u_1, \dots, u_n \xleftarrow{\$} \mathbb{F}_p$ uniformly and independently with all u_i nonzero, call our secret $f_0 \in \mathbb{F}_p$, set $f = f_{n-1}x^{n-1} + \dots + f_1x + f_0 \in \mathbb{F}_p[x]$, and give to player number i the value $f(u_i) \in \mathbb{F}_p$. If $u_i = u_j$ for some $i \neq j$, we have to make a new random choice; this is unlikely to happen if n is much smaller than $\sqrt{p}$, according to the birthday Theorem 3.41 and Exercise 3.18 (iv). Then all players together can determine the (unique) *interpolation polynomial* f of degree less than n , and thus f_0 . But if any smaller number of them, say $n - 1$, get together, then the possible interpolation polynomials consistent with this partial knowledge are such that each value in $\mathbb{F}_p$ of f_0 is equally likely: they have no information on f_0 . The tools for interpolation are described in Section 15.10.

We verify this security claim for a coalition of the $n - 1$ players $1, \dots, n - 1$. There is a unique interpolation polynomial $g \in \mathbb{F}_p[x]$ of degree at most $n - 2$ for their values, so that $g(u_i) = f(u_i)$ for all $i < n \leq p$. All possible interpolation polynomials for n players are of the form $s = g + h \prod_{1 \leq i < n} (x - u_i)$, with $h \in \mathbb{F}_p[x]$. The degree of the product is $n - 1$, and the degree of the sum is $\deg s = n - 1 + \deg h$, which is less than n —as required for f —if and only if $\deg h \leq 0$, that is, if and only if h is a constant. Now the constant term of the product is $u = (-1)^{n-1} \prod_{1 \leq i < n} u_i \neq 0$, by the choice of the u_i 's. Thus the constant term

$$s(0) = g(0) + h \cdot u$$

of s may be any value $v \in \mathbb{F}_p$ for the appropriate unique choice

$$h = (v - g(0))/u$$

of h . In other words, the $n - 1$ players have no information about $f_0 = f(0)$, since any value for it is consistent with their knowledge $f(u_1), \dots, f(u_{n-1})$.

EXAMPLE 2.27. Suppose that $n = 3$, $p = 7$, and the secret is $f_0 = 6$. We choose randomly $u_1 = 2$, $u_2 = 3$, $u_3 = 5$, $f_1 = 4$, $f_2 = 2$. Then the three players receive:

player i	1	2	3
point u_i	2	3	5
value $f(u_i)$	1	1	6

Together they can compute the unique interpolation polynomial $f = f_2x^2 + f_1x + f_0 = 2x^2 + 4x + 6$ and, in particular, the secret $f_0 = 6$.

But if only, say, players 2 and 3 collaborate, their unique interpolation polynomial is $g = 6x + 4$ of degree 1. All quadratic polynomials $s_h = g + h \cdot (x - 3)(x - 5)$ with $h \in \mathbb{F}_7$ interpolate these two values, and the constant term of s_h is $s_h(0) = h + 4$. When h runs through $\mathbb{F}_7$, every value in $\mathbb{F}_7$ appears exactly once as $h + 4$; for example, $s_6 = 6 + 4 = 3$ in $\mathbb{Z}_7$. The “correct” value $h = 2$ is as likely as any other. $\diamond$

This secret sharing scheme is *information-theoretically secure*: no matter how large the computational resources of players 2 and 3 are, they cannot find out anything about the secret f_0 .

We can extend this scheme to the situation where $k \leq n$ and each subset of k players is able to recover the secret, but no set of fewer than k players can do this. This is achieved by randomly and independently choosing $n + k - 1$ elements $u_0, \dots, u_{n-1}, f_1, \dots, f_{k-1} \in \mathbb{F}_p$ and giving $f(u_i)$ to player i , where $f = f_{k-1}x^{k-1} + \dots + f_1x + f_0 \in \mathbb{F}_p[x]$ and $f_0 \in \mathbb{F}_p$ is the secret as above. Again, it is required that $u_i \neq u_j \neq 0$ if $i \neq j$. Since f is uniquely determined by its values at k points, each subset of k out of the n players can calculate f and thus the secret f_0 , but fewer than k players together have no information on f_0 .

2.10. Visual cryptography

The system discussed here provides a visual representation of a secure symmetric cryptosystem such as the one-time pad; see Sections 2.8 and 9.4. In its simplest variant, this scheme of Naor & Shamir (1995) transmits an image by first creating a random image as secret key and then a second image depending on it and the message. By itself, this second image is again randomly generated, but the two images are highly correlated.

For illustration, suppose a company manager stays at a hotel for negotiations with another company. If she requires information from home, maybe a blueprint or picture, her company sends her the second image by fax. Anyone seeing this fax alone obtains no information. But she can superimpose her secret key slide, which she took with her, on the fax and see the message.

How is this achieved? The plaintext image is split into square pixels, each of which is either black or white. Each pixel is further divided equally into four square subpixels. Both in the random key and in the encrypted message, exactly two of the four subpixels are black, and two are white. There are six possible arrangements of two blacks in a 2×2 square as in [Figure 2.19](#). For the random key, one of the six is chosen uniformly at random, and independently for each of the many pixels. For the encryption, we choose the same arrangement as on the key if the plaintext pixel is white, and the complementary one if the plaintext pixel is black. If we

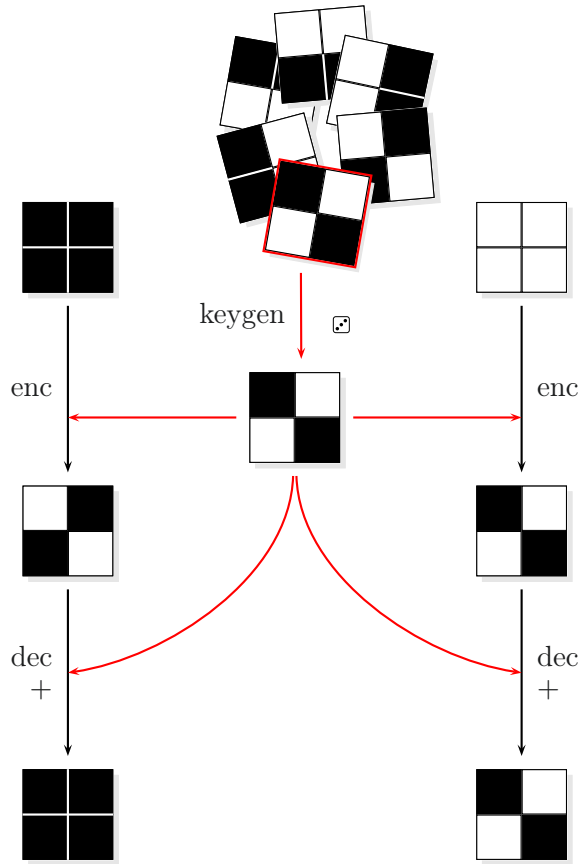


Figure 2.19: Visual cryptography—single pixel operation.

then superimpose the key and the encryption, we have exactly two or four subpixels black if the plaintext pixel is white or black, respectively.

More information about the *VisKy* system of visual encryption is available on the book's website, including examples and javascript software that allows you to encrypt your own images.

Notes 2.1. AES is not completely fixed, but rather there are three choices for the key length.

2.2. EFF's website is <http://www.eff.org/>.

For the AES selection process, see <http://csrc.nist.gov/archive/aes/>. AES was announced by NIST as US FIPS PUB 197 on 26 November 2001 which became effective on 26 May 2002. It may be used in the United States up to the top secret level. *Rijndael* is related to the Flemish word for *Rhine valley* and a play on the inventors' names. A visual presentation of AES is available at the book's website. Matrix entries are usually read row-before-column, as $a_{00}, a_{01}, \dots$, but AES uses a different convention for its 4×4 blocks.

Hamming (1980) coined "*the unreasonable effectiveness of mathematics*".

Osvik *et al.* (2010) report a throughput of 12.9 GB/sec for AES on an NVIDIA Graphics Processing Unit 8800 GTX running at 1.35 GHz. See Bingmann (2008) for an extensive but slightly more dated comparison of various platforms and standard software packages.

The expected cost of exhaustive search for a 128-bit key is 2^{127} operations. The quote at the end of the section is from an undated NSA document provided by Edward Snowden and published in the article Appelbaum *et al.* (2014). Attacks on specific AES implementations are known that use differential power analysis, fault attacks, cache misses, or cold boot.

2.5. The three inventors of RSA were postdocs at MIT at that time. They later rose to prominent positions in science, at MIT, the Weizmann Institute, and the University of Southern California, respectively. They founded in 1982 a company now called RSA Labs, sold it in 1996, and its present owner acquired it in 2006 for 2.1 billion US\$.

ALICE and BOB were both born on 4 April 1977, or at least we found the first usage of these names as players in a cryptosystem in the technical memo MIT/LCS/TM-82 of that date, in which the inventors of RSA present their system and write about users “A and B (also known as Alice and Bob)”.

There are several ways of prescribing the generation of the RSA modulus N precisely; see Loebenberger & Nüsken (2014) for a discussion.

According to the memos posted on the web page <http://www.cesg.gov.uk/about/nsecret.htm>, James Henry Ellis put forward in 1970 the idea of public key cryptography, which he called *non-secret encryption*. Clifford Christopher Cocks proposed in 1973 a practical implementation which was essentially equivalent to RSA (using $e = N$ in the RSA notation), and Malcolm Williamson suggested the Diffie-Hellman key exchange in $\mathbb{Z}_p^\times$ in 1976 (with a precursor in 1974). The CESG, based in Cheltenham, Gloucestershire, and today called the GCHQ is a descendant of the *Government Code and Cypher School* (GCCS) of ENIGMA-breaking fame; see Chapter J.

2.6. The key exchange idea was first published by Diffie & Hellman (1976). Their proposal used the only type of group then under consideration in cryptography, namely the group of units modulo a prime. The generalization to arbitrary cyclic groups is straightforward. In Figure 2.17, EVE might also use two different values of e , one for ALICE and one for BOB.

We do not discuss efficient pairings on an elliptic curve E in this text. In such groups E , DDH_E is easy, but before the advances in discrete logarithms mentioned in Section 4.8, DH_E looked hard.

It is not known whether $\text{DL}_G \leq_p \text{DH}_G$, but this was shown to be true under some plausible assumptions by Maurer & Wolf (1999). They take a cyclic group G with order $d = \#G$ and bit size $n \approx \log_2 d$, and assume that the set P of prime factors of d is given. According to Figure 4.6, even the case $P = \{d\}$ is of great interest. For $p \in P$ with p^2 dividing d , they assume that $p \in \text{poly}(n)$. For any prime p , they consider the Hasse interval $I_p = \{i: |i - p - 1| \leq 2\sqrt{p}\}$ of sizes of elliptic curves over $\mathbb{Z}_p$ (Theorem 5.10). Furthermore, $\nu(p)$ denotes the smallest integer s for which there exists an s -smooth integer in I_p , and $m = \max\{\nu(p): p \in P\}$. Maurer & Wolf exhibit a reduction from DL_G to DH_G using time $\sqrt{m} \cdot \text{poly}(n)$. Under the plausible assumption that $\nu(p) = (\log p)^{O(1)}$, the running time is polynomial in n . Proving this assumption seems out of reach for current methods in number theory. Even showing that $\nu(p) \leq p^\alpha$ for all $\alpha > 0$ is an open question; see Granville (2008), Section 4.4, for a description of the state of the art in smoothness bounds throughout his survey.

2.7. For an n -bit RSA modulus N , it would be more natural to take $\mathbb{Z}_N$ as plaintext and ciphertext spaces. The bit lengths n and $n+1$ only depend on n and are chosen because,

identifying numbers and their binary representations, we have $\{n\text{-bit integers}\} \subseteq \mathbb{Z}_N = \{0, \dots, N-1\} \subseteq \{(n+1)\text{-bit integers}\}$.

2.8. Here is a potential attack on ECB. The adversary sets up an internet shop and sells laptops at slightly lower prices than everybody else. He intercepts the encrypted versions of bank transfers from his clients to him, and might actually deliver the goods. Suppose that the beneficiary's account in a transfer is in the fourth of twenty blocks. He now has the encryptions of this fourth block and can insert it into later transfers that he intercepts. These monies would then end up in his account. (An attack might not work this way, but it illustrates the idea.)

A variant of the OFB mode is used in the ISO 10116 standard. A security reduction for the counter mode is in Rackoff (2012). Goldreich (2004), Chapter 5, discusses security aspects of various modes.

2.10. Visual cryptography was invented by Naor & Shamir (1995). Several variations and generalizations have been studied in the literature; see Cimateo (2011).

Exercises.

EXERCISE 2.1 (Birth number). *Let d be your date of birth in the format Year-MonthDay, so that 24 May 1990 gives $d = 19900524$. Let $N = 2^{16} + 1$, $a = d$ in $\mathbb{Z}_N$ with $0 \leq a < N$, $b = (d - a)/N = \lfloor d/N \rfloor$, and $c = a + b$ in $\mathbb{Z}_N$. Thus $c = 43116$ in the example. See Section 15.3 for division with remainder. We call c your birth number and will use it in several exercises. Now interpret the low-order eight bits a^* , b^* , c^* of the binary representations of a , b , c , respectively, as elements of $\mathbb{F}_{256}$, just as in AES. Compute*

- (i) $a^* + b^*$, and compare to c^* ,
- (ii) $a^* \cdot b^*$,
- (iii) $\text{inv}(a^*)$. State your year of birth and give a date in the same year for which $a^* = 0$.

EXERCISE 2.2 (One round of AES). *In this exercise we compute the first round of AES by hand. We start with an input matrix*

$$\begin{pmatrix} 01 & 11 & 21 & 31 \\ 02 & 12 & 22 & 32 \\ 03 & 13 & 23 & 33 \\ 04 & 14 & 24 & 34 \end{pmatrix}$$

and a key

$$\begin{pmatrix} AA & BB & CC & DD \\ AA & BB & CC & DD \\ AA & BB & CC & DD \\ AA & BB & CC & DD \end{pmatrix},$$

where all entries are in hexadecimal representation.

- (i) Compute `ADDROUNDKEY` for the first two bytes.
- (ii) Compute `SUBBYTES` for the two bytes that result in (i).
- (iii) After `SUBBYTES`, the matrix looks like

$$\begin{pmatrix} * & * & 55 & CE \\ C2 & D3 & 28 & DF \\ D3 & C2 & DF & 28 \\ E4 & 79 & 9B & 1E \end{pmatrix}.$$

Compute SHIFTRows of this matrix.

- (iv) Compute MIXCOLUMNS for the last column of the matrix that results in (iii).

EXERCISE 2.3 (Encryption and decryption with AES). Consider $t_1, t_0 \in R$ as in SUBBYTES.

- (i) Check that $\gcd(t_1, x^8 + 1) = 1$ in $\mathbb{F}_2[x]$.
 (ii) Compute the inverse of t_1 in $\mathbb{F}_2[x]/(x^8 + 1)$.
 (iii) Prove that $t_1 \cdot a + t_0 = t_1 \cdot (a + (00000101))$ for all bytes $a \in \mathbb{F}_{256}$.
 (iv) Compute the inverse P^{-1} of the following matrix P modulo 2:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Is there any relation to your result from (ii)?

- (v) Show that for all bytes $a \in \mathbb{F}_{256}$, we have $\text{inv}(a) = a^{254}$. Use Fermat's Little Theorem 15.56.
 (vi) Given an output of the function SUBBYTES, how can you compute the corresponding 1-byte input?
 (vii) Verify that $(0By^3 + 0Dy^2 + 09y + 0E) \cdot (03y^3 + 01y^2 + 01y + 02) = 1$ in $\mathbb{F}_{256}[y]/(y^4 + 1)$.
 (viii) SUBBYTES is given by the polynomial expression $a \mapsto 05 \cdot a^{254} + 09 \cdot a^{253} + F9 \cdot a^{251} + 25 \cdot a^{247} + F4 \cdot a^{239} + 01 \cdot a^{223} + B5 \cdot a^{191} + 8F \cdot a^{127} + 63$. What does this mean? Where do the various operations take place? Illustrate each by an example which is not in this text.
 (ix) Specify a decryption algorithm AES^{-1} that recovers the plaintext from an AES encryption and the secret key.

EXERCISE 2.4 (The two byte rings). The elements of the AES-rings $\mathbb{F}_{256}$ and $R = \mathbb{F}_2[x]/(x^8 + 1)$ can each be encoded using 8 bits per element. Show that the two rings are not isomorphic; see Section 15.2.

EXERCISE 2.5 (Commuting steps?). Do the following operations commute? That is, is the sequence in which you apply them irrelevant?

- (i) SUBBYTES and SHIFTRows.
 (ii) SHIFTRows and MIXCOLUMNS.
 (iii) MIXCOLUMNS and ADDROUNDKEY.
 (iv) ADDROUNDKEY and SUBBYTES.

EXERCISE 2.6 (Skipping a step). Consider the four operations (SUBBYTES, SHIFTRows, MIXCOLUMNS and ADDROUNDKEY) performed in each round of AES. An obvious proposal to speed up the encryption is to omit one of the steps. Comment on the security of a modified 128-bit AES, where each round leaves out the following step.

- (i) SUBBYTES,
- (ii) SHIFTRows,
- (iii) MIXCOLUMNS,
- (iv) ADDROUNDKEY.

EXERCISE 2.7 (AES operations). Take Baby-AES from Section 6.1 and do the following for all four AES operations $X \in \{\text{SUBBYTES}, \text{SHIFTRows}, \text{MIXCOLUMNS}, \text{ADDROUNDKEY}\}$.

- (i) For which blocks a is $X(a) = a$?
- (ii) Given a block b , describe how to compute a block a with $X(a) = b$.
- (iii) Do (i) and (ii) for the real AES of Section 2.2.

You may make reasonable assumptions about the key.

EXERCISE 2.8 (RSA keys). We want to build an RSA system using the prime numbers $p = 251$ and $q = 337$. (In practice, these numbers are much too small.) Furthermore we choose $e = 54323$ and $N = p \cdot q$ for our public key. Use the Extended Euclidean Algorithm (Section 15.4) to compute the matching secret key d with $e \cdot d = 1$ in $\mathbb{Z}_{\varphi(N)}$.

EXERCISE 2.9 (3RSA). The RSA system can be generalized to allow for products of more than two distinct prime numbers. Consider the RSA system for products $N = p_1 \cdot p_2 \cdot p_3$ of three distinct primes:

ALGORITHM. 3RSA key generation with security parameter n .

1. Choose three distinct prime numbers $p_1 < p_2 < p_3 < 2p_1$, so that

$$2^{n-1} < N = p_1 \cdot p_2 \cdot p_3 < 2^n.$$

2. Compute $\varphi(N) = (p_1 - 1) \cdot (p_2 - 1) \cdot (p_3 - 1)$.
3. Choose the public exponent $e \in \mathbb{Z}_{\varphi(N)}^{\times} \setminus \{2, 3, \dots, \varphi(N) - 2\}$ so that $\gcd(e, \varphi(N)) = 1$.
4. Compute the secret exponent d so that $d \cdot e = 1$ in $\mathbb{Z}_{\varphi(N)}$.
5. Now (N, e) is the public key and (N, d) is the secret key.

With these keys, the remainder of the system is identical to the one using products of two distinct primes.

- (i) Prove that the system works correctly.
- (ii) Which advantages and disadvantages of this system over standard RSA do you see? Efficiency? Security?
- (iii*) Generalize the RSA system to products $N = p_1 \cdot p_2 \cdots p_r$ of r primes. Are there noteworthy differences?

EXERCISE 2.10 (Modes of operation). Answer the following questions concerning error propagation for each of the six modes of operation described in Section 2.8. Let t be the time to encrypt one block of n bits.

- (i) How many text blocks are incorrect if one of the transmitted blocks is corrupted?
- (ii) How many text blocks are incorrect if one of the transmitted blocks is dropped and this is not noticed?
- (iii) How many text blocks are incorrect if one of the block encryptions outputs a wrong result?
- (iv) How long does it take to encrypt a message of $k \cdot n$ bits?
- (v) Draw conclusions from your observations.
- (vi) We define a further mode *Plain Block Chaining* that adds the plaintext x_{i-1} to the encryption of x_i as in the following picture.

$$\begin{array}{c}
 \cdots x_{i-1} x_i x_{i+1} \cdots \\
 \downarrow \\
 y_i = \text{enc}_K(x_i) \\
 \downarrow \\
 z_i = y_i \oplus x_{i-1}
 \end{array}$$

PBC

Answer the questions (i)–(v) also for this mode.

EXERCISE 2.11 (Caesar). We consider the 1-letter block cipher Caesar_3 and the plaintext *CryptoSchool*. Calculate the six encryptions according to the six chaining methods in Section 2.8. Use the English alphabet and $IV = \text{green}$ where needed.

EXERCISE 2.12 (Secret sharing). As an example for the secret sharing method of Section 2.9, we take $p = 10000019$, $n = 5$, and $u_1 = 1484998$, $u_2 = 8055552$, $u_3 = 412501$, $u_4 = 8994679$, $u_5 = 236054$.

- (i) Compute the secret from $f(u_1) = 2016419$, $f(u_2) = 951970$, $f(u_3) = 9707737$, $f(u_4) = 6395629$, $f(u_5) = 8552973$.
- (ii) Can you find a value for $f(u_5)$, so that the corresponding secret $f(0)$ is your birth number from Exercise 2.1, without changing the other values?

We investigate which data yields sensitive information and which data does not. We set $p = 1009$ so that iterations over all of $\mathbb{F}_p$ are reasonable, and $n = 7$. Let f_0 be your birth number (Exercise 2.1) in $\mathbb{F}_{1009}$, choose $u_0, \dots, u_6, f_1, \dots, f_6 \in \mathbb{F}_{1007}$ at random with the u_i nonzero and pairwise distinct, and $u_1, \dots, u_6$ not 1008.

- (iii) Suppose that $u_0 = 1008$ and that a coalition of the secret bearers 1 through 6 learns this by an indiscretion. Compute the distribution of possible secrets. That is, try all values for $f(u_0)$ and count how many times each possible secret occurs as the value $f(0)$.
- (iv) Now suppose that $f(u_0) = 1008$ and a coalition of secret bearers 1 through 6 learns this fact. Compute the distribution of possible secrets by trying all values for u_0 and counting the number of times that each possible secret occurs as the value $f(0)$.
- (v) Compare the results: is one of the indiscretions a problem for secret bearer number 0? Which one? Why? Can you describe “how much” information was disclosed?

Hanc Graecis conscriptam litteris mittit,
ne intercepta epistola nostra ab hostibus consilia cognoscantur.¹
CAESAR (c. 50 BC)

It must have been one of those ingenious secret codes which
mean one thing while they seem to mean another.
I must see this letter. If there were a hidden meaning in it,
I was confident that I could pluck it forth.
SIR ARTHUR CONAN DOYLE (1893)

La cifra semplice per ciò à lungo andare può più facilmente
essere intesa per congettura, senza contracifra,
però si è trovato di formare le cifre più varie
et con caratteri doppii e triplicati et con molte altre cose,
per iugannare li decifраторi.²
MATTEO ARGENTI (c. 1605)

The inclusion of references in other languages may help
to break down the linguistic provincialism which,
ostrichlike, takes refuge in the mistaken impression
that everything worthwhile appeared in,
or has been translated into, the English language.
CARL BENJAMIN BOYER (1968)

‘The letter?—Oh!—The letter! Keep looking at me between the
eyes, please. It was a string-talk letter, that we’d learned the way
of it from a blind beggar in the Punjab.’ I remembered that there
had once come to the office a blind man with a knotted twig and
a piece of string which he wound round the twig according to
some cipher of his own. He could, after the lapse of days or
weeks, repeat the sentence which he had reeled up. He had
reduced the alphabet to eleven primitive sounds, and tried to
teach me his method, but I could not understand.
RUDYARD KIPLING (1888)

¹He sent this letter written in Greek characters, lest by intercepting it the enemy might get to know of our designs.

²The simple cipher can, because of this, in the long run rather easily be broken by guessing, without the key, but one has found how to design more varied ciphers with double and triple characters and many other things, to fool the decipherers.

Lectures on Functor Homology

Franjou, V.; Touzé, A. (Eds.)

2015, VI, 149 p. 140 illus., 1 illus. in color., Hardcover

ISBN: 978-3-319-21304-0

A product of Birkhäuser Basel