Introduction to Cryptography
Lecture Notes 2020

Joan Daemen, Bart Mennink, Jan Schoone

Version: 13 February 2020

Version 13 February 2020

Contents

List of Algorithms

Part I

Introduction to symmetric cryptography

Chapter 1

One-time pad encryption

Suppose that Alice wants to send messages to Bob in such a way that no adversary that can eavesdrop on their communication can learn *anything* about the content of the messages. They have no problem with the adversary knowing the length of the messages nor their time of transmission. In other words, they want to protect the confidentiality of the *contents* of the messages. This can be done with *encryption.*

1.1 One-time pad

Assume that Alice and Bob have established a shared secret key K that is at least as long as the message they want to transmit. We will later discuss how Alice and Bob can establish a shared secret and will for now assume they have it.

Consider the case where Alice wants to send a 2MB file M to Bob. In its simplest form, this is nothing more than a string of $2 \cdot 1024 \cdot 1024 \cdot 8 = 16777216$ bits. Alice can *encipher* the message M into a *cryptogram* C by adding the key K bitwise to M:

$$C = M \oplus K \,. \tag{1.1}$$

Alice then sends C to Bob, who can recover the message M from C using the shared key K:

$$M = C \oplus K \,.$$

This encryption scheme is called the *one-time pad (OTP).* In 1949, Shannon proved that OTP offers perfect confidentiality against an eavesdropper Eve if K has a uniformly random distribution. Intuitively, M is the bitwise sum of the cryptogram C observed by Eve and K that is fully random as far as Eve is concerned, and so M is also fully random. In other words, for Eve, C gives no information on M.

In a bit more detail, suppose that Eve, with knowledge of C, guesses that the corresponding message is $M^\star$. By basic probability, Eve is right with probability

$$\Pr(M = M^\star) = \Pr(C \oplus K = M^\star) = \Pr(K = C \oplus M^\star) = 1/16777216 \,,$$

as K has a uniformly random distribution.

1.2 Using the same key twice

Alice and Bob should be careful, though, not to reuse the key K! Suppose Alice enciphers another 2MB file M' with the same key K and sends the resulting cryptogram C' to Bob:

$$C' = M' \oplus K\,.$$

Eavesdropper Eve, might have intercepted both C and C'. This gives her an additional advantage in guessing the corresponding messages. The reason for this is that if Eve simply *adds C and C'*, it learns the bitwise sum (or, equivalently, difference) between M and M':

$$C \oplus C' = (M \oplus K) \oplus (M' \oplus K) = M \oplus M'\,.$$

Given some knowledge about M, Eve may derive information about M'.

We note that Eve may, conceivably, have some knowledge about any of the two messages.

Example 1.2.1. Various file types often start with a header that is (almost) the same for different files of the same type. For example, for a typical html file may start with

```
<html><head><title>Fancy website</title></head>...,
```

and a typical pdf file may start with

```
%PDF-1.5
5 0 obj
<<
/Type /ObjStm
/N 100
/First 810
/Length 1193
/Filter /FlateDecode
>>
stream
...
```

♠

1.3 Remedy

The remedy to the situation is that Alice and Bob set up a longer key K. This key must be as long as the total length of all messages exchanged between Alice and Bob. If a message M must be encrypted and transmitted, Alice starts from the first key bit that has not been "consumed" yet. For example, if Alice wants to encrypt messages M_1 of length m_1, M_2 of length m_2, and M_3 of length m_3, the corresponding cryptograms are:

$$\begin{aligned}
C_1 &= M_1 \oplus [\text{bits } 1 \ldots m_1 \text{ of } K]\,,\\
C_2 &= M_2 \oplus [\text{bits } m_1 + 1 \ldots m_2 \text{ of } K]\,,\\
C_3 &= M_3 \oplus [\text{bits } m_1 + m_2 + 1 \ldots m_3 \text{ of } K]\,.
\end{aligned}$$

The remaining bits of K, namely bits $m_1+m_2+m_3+1$ onwards, can be used for later encryptions. In this way, OTP is a very simple encryption scheme that offers perfect confidentiality. It has been used successfully in, e.g., espionage and diplomatic correspondence .

However, for many applications it is not a satisfactory solution:

- The amount of key material to be kept must be as large as the amount of communication that one wants to protect with it. In modern communication this may become quite large, think of, e.g., media content. Moreover, Alice may not only want to communicate securely with Bob, but also with Carl, Donald, Ethan, Francis, Giovanni, Helen, and so on. So she needs these very long keys for each of these guys 'n gals, taking up even more storage.

- The only practical way for Alice and Bob to securely establish an arbitrarily large uniform random key is by physically meeting each other in person. Alice and Bob may have never physically met each other, or they may have decided that they want to secure their communication *after* meeting each other;

- In most applications, we wish to reduce the problem of protecting the confidentiality of large amounts of data by that of protecting the secrecy of a short cryptographic key. A very telling example of this is disk encryption. Say, Alice wants to prevent the data on her laptop falling in the wrong hands in case it would get stolen or found after she would have forgotten it somewhere. This can be achieved with encryption. However, if Alice wants to encrypt the data M on a 224GB hard disk using OTP into C, she would first need to buy a 224GB hard disk to store the key K and make sure the thief or finder cannot get his hands on the K hard disk. Interestingly, both the C and the K hard disk are needed to reconstruct the data M, also by Alice, and in this case the probability of data loss due to disk crash almost doubles (do you see why?).

Chapter 2

Stream ciphers

Consider the case that Alice and Bob managed to agree on a relatively short random secret key K, for instance one of 256 bits, or 32 bytes. Clearly, a 256-bit key K is too short to encrypt a 2MB file M using one-time-pad encryption. The solution lies in the concept of *stream ciphers.* A stream cipher SC is a type of *cryptographic primitive.* Cryptographic primitives take an input in a well-specified domain and generates an output in some other specified domain. A stream cipher takes as input a fixed-length key K, and returns an arbitrarily long keystream Z. We write:

$$\mathrm{SC}\colon \{0,1\}^{256} \to \{0,1\}^{*}\,,\ K \mapsto Z\,.$$

In mathematical terminology, this is just a *function.* The keystream Z is in principle infinite so in practical implementations one must in some way specify how many keystream is desired, e.g., with an input parameter ℓ. The stream cipher then just returns the first ℓ bits of the keystream Z. Note that the bits of the keystream Z is fully determined by the key K.

The underlying idea is that the keystream Z generated by a stream cipher "looks random" for someone that does not know the key K. In other words, Alice and Bob can use Z for OTP encryption and only have to keep K secret. Both can evaluate the stream cipher to generate keystream Z on the fly, Alice when enciphering and Bob when deciphering.

Kerckhoffs' principle. An important concept in cryptography is the separation between the *cryptographic primitive* and the key K. As said, the former defines a function and this is usually described in an *algorithm* and implemented as a program or a hardware circuit on a computing device. When using a stream cipher for encryption both Alice and Bob have such a physical device implementing the algorithm and containing their shared key K. The algorithms of modern stream ciphers are public information and can be read in standards, research papers or industrial documentation. See for example the algorithm of RC4 on wikipedia: https://en.wikipedia.org/wiki/RC4. This was not always the case and in particular RC4 was kept secret for a quite some time until it was reverse engineered from some implementation. In fact it is good practice to base the security of encryption on the secrecy of the key K only and not that of the algorithm. This has become known as Kerckhoffs' principle, named after the 19th century cryptographer Auguste Kerckhoffs that first formulated it, see https://en.wikipedia.org/wiki/Auguste_Kerckhoffs.

Alice and Bob can use a stream cipher to generate 2MB of keystream Z from their shared secret K and encrypt a 2MB message M by adding Z. Alice enciphers M to C as follows:

$$C = M \oplus \mathrm{SC}(K)\,, \tag{2.1}$$

where in $M \oplus \mathrm{SC}(K)$ the stream cipher output (keystream) is truncated to the length of M. She sends C to Bob, who deciphers it as follows:

$$M = C \oplus \mathrm{SC}(K)\,. \tag{2.2}$$

2.1 Linear feedback shift registers

To make the concept of a stream cipher more tangible, we give in the following sections some concrete examples of stream cipher algorithms. These are all based on so-called linear feedback shift registers (LFSR).

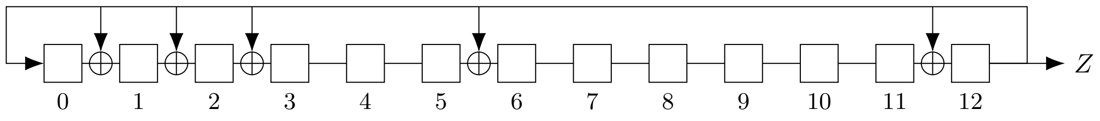


Figure 2.1: Example Galois LFSR.

We depict an example LFSR in Figure 2.1. It has a state of 13 bits. This state is initialized using a key, and for each iteration, the last bit will be part of the keystream Z as well as used to update the state. For example, if we initialize the state with `0101010101010`, the first iteration gives keystream bit `0` and updates the state to `0010101010101`. The second iteration generates keystream bit `1`, and updates the state to `1110011101011`.

So far, we have generated two keystream bits `01`. Proceeding the process yields the following states, where underlined bits are those affected by the $\oplus$:

$$\begin{gathered}
\mathtt{0}\underline{\mathtt{101}}\mathtt{01}\underline{\mathtt{0}}\mathtt{10101}\underline{\mathtt{0}}\,,\\
\mathtt{0}\underline{\mathtt{010}}\mathtt{10}\underline{\mathtt{1}}\mathtt{01010}\underline{\mathtt{1}}\,,\\
\mathtt{1}\underline{\mathtt{110}}\mathtt{01}\underline{\mathtt{1}}\mathtt{10101}\underline{\mathtt{1}}\,,\\
\mathtt{1}\underline{\mathtt{000}}\mathtt{00}\underline{\mathtt{0}}\mathtt{11010}\underline{\mathtt{0}}\,,\\
\mathtt{0}\underline{\mathtt{100}}\mathtt{00}\underline{\mathtt{0}}\mathtt{01101}\underline{\mathtt{0}}\,,\\
\mathtt{0}\underline{\mathtt{010}}\mathtt{00}\underline{\mathtt{0}}\mathtt{00110}\underline{\mathtt{1}}\,,\\
\mathtt{1}\underline{\mathtt{110}}\mathtt{00}\underline{\mathtt{1}}\mathtt{00011}\underline{\mathtt{1}}\,,\\
\mathtt{1}\underline{\mathtt{000}}\mathtt{00}\underline{\mathtt{1}}\mathtt{10001}\underline{\mathtt{0}}\,,\\
\mathtt{0}\underline{\mathtt{100}}\mathtt{00}\underline{\mathtt{0}}\mathtt{11000}\underline{\mathtt{1}}\,,\\
\mathtt{1}\underline{\mathtt{101}}\mathtt{00}\underline{\mathtt{1}}\mathtt{01100}\underline{\mathtt{1}}\,,\\
\mathtt{1}\underline{\mathtt{001}}\mathtt{10}\underline{\mathtt{1}}\mathtt{10110}\underline{\mathtt{1}}\,,\\
\mathtt{1}\underline{\mathtt{011}}\mathtt{11}\underline{\mathtt{1}}\mathtt{11011}\underline{\mathtt{1}}\,.
\end{gathered}$$

After these iterations, we have generated a keystream `01100110111`.

Formally, one can describe this LFSR using states $s^0, s^1, s^2, \ldots$ and keystream bits $z_0, z_1, z_2, \ldots$. Here, $z_i = s^i_n$, i.e., the last bit of the previous state. Each iteration evolves the state as follows

$$s_{i+1} \leftarrow \begin{cases} s_i & \text{if there is no feedback tap at } s_{i+1}\,; \\ s_i + s_n & \text{if there is a feedback tap at } s_{i+1}\,. \end{cases}$$

In the LFSR of Figure 2.1, the feedback taps are at positions $1, 2, 3, 6$, and 12.

2.1.1 *LFSRs by polynomials

If we represent the current state of an LFSR s^i by $(s^i_0, \ldots, s^i_{n-1})$ then we can calculate s^{i+1} like we discussed above. Using polynomials over $\mathbb{F}_2$, we can find a bijective correspondence between states s^i and polynomials $S(X)$ determined by

$$S^i(X) = s^i_0 + s^i_1 X + \ldots + s^i_n X^n \, .$$

The iteration of the LFSR can then be calculated in terms of these polynomials. We have

$$S^{i+1}(X) = X \cdot S^i(X) \bmod q(X) \, ,$$

where $q(X)$ is the *feedback* polynomial of the LFSR. We have $\deg q(X) = n+1$ and a coefficient 1 at X^j if and only if there exists a feedback tap in front of the jth box.

For example, the feedback polynomial of the Galois LFSR of Figure 2.1 is $q(X) = X^{13} + X^{12} + X^6 + X^3 + X^2 + X + 1$.

Indeed, let the initial state be given as before: `0101010101010`. Then this corresponds to $S^0(X) = X + X^3 + X^5 + X^7 + X^9 + X^{11}$. Then $S^1(X) = X \cdot S^0(X) = X^2 + X^4 + X^6 + X^8 + X^{10} + X^{12}$, and $S^2(X) = X \cdot S^1(X) = X^3 + X^5 + X^7 + X^9 + X^{11} + X^{13}$. Taking this last polynomial modulo $q(X)$, we get:

$$S^2(X) = X^3{+}X^5{+}X^7{+}X^9{+}X^{11}{+}X^{12}{+}X^6{+}X^3{+}X^2{+}X{+}1 = 1{+}X{+}X^2{+}X^5{+}X^6{+}X^7{+}X^9{+}X^{11}{+}X^{12}.$$

It is a straightforward verification that this coincides to the manually updated states from before.

Using this method, it is easy to compute, say 5, iterations at once. You just calculate $S^{i+5}(X) = X^5 \cdot S^i(X) \bmod q(X)$. If $q(X)$ is irreducible, these states will cycle. If $q(X)$ is primitive, then this cycle has the maximal length, as X is a generator of the field $\mathbb{F}_2[X]/q(X)$. (This is the field $\mathbb{F}_{2^n}$.)

2.1.2 LFSRs by matrices

Consider the LFSR from Figure 2.1 again. Also, consider the following matrix

$$\begin{pmatrix}
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\
1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\
\cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\
\cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\
\cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1
\end{pmatrix}$$

where a $\cdot$ denotes a 0. What they have in common, is that when given some state s^0 and iterating the LFSR, then applying the matrix to s^0 returns some s^1 that coincides with the LFSR applied to s^0.

So, in other words $s^{t+1} = M \cdot s^t$ where s^j are states of the LFSR and M the matrix corresponding to the LFSR. We see that the word Linear is justified, the LFSR is just a linear map!

In general, one can say that the matrix corresponding to an LFSR is of the following form:

$$\begin{pmatrix}
0 & \cdots & 0 & 1 \\
 & & & a_1 \\
 & I_n & & \vdots \\
 & & & a_{n-1}
\end{pmatrix}$$

Each a_i is then a 1 if and only if before statebit i we do an addition of the last statebit.

In the next section, among others, we will show that this linearity is a weakness of the LFSRs.

2.2 Attacks on stream ciphers

We exemplify two attacks on the LFSRs that retrieve the entire secret state of the LFSR. These two attacks display pitfalls we wish to avoid when building stream ciphers.

2.2.1 Exhaustive key search

One attack that could be performed on an arbitrary LFSR is the *exhaustive key search*. In an exhaustive key search attack, Eve has obtained a certain ciphertext, and tries to decrypt it will all possible keys. Here, it is typically assumed that Eve knows the cryptographic function that has been used (Kerckhoffs principle), but not the initial state.

Example 2.2.1. Alice want to send a "meaningful" message P to Bob, for instance a piece of English text. To do so, she uses an LFSR and an initial state s that both she and Bob have agreed on. She generates a keystream Z that is as long as the original message and adds it to her message. The result $C = P + Z$ is sent to Bob, but stolen by an adversary.

Eve can perform an exhaustive key search as follows. For each possible initial state (there are 2^n of those), she generates a keystream Z' and computes $Z' + C$. If the result is a "meaningful" text, she has probably chosen the correct initial state and $Z' + C$ is the message P that Alice sent to Bob. ♠

In above example, 2^n possible initial states exist and they all have equal probability to be correct. On the average, the right key will be hit after trying about half of the key space. This means that it requires Eve to try 2^{n-1} initial states in order to guess the key correctly.

So as opposed to OTP, encryption with a stream cipher does not offer perfect secrecy: in fact, an adversary can find the plaintext message M from C alone via exhaustive key search. However, this comes at a high computational cost. If the key K is long enough, the cost may be prohibitively high.

Exhaustive key search is not limited to LFSRs or even to stream ciphers. Instead, any cryptographic primitive that uses a secret key can be attacked with exhaustive key search.

2.2.2 Exploiting the linearity of the LFSR

This subsection is devoted to the weakness that linearity implies. We first show an easy attack that is possible when knowing n subsequent bits of the keystream of an LFSR to determine all states from the initial state onwards.

Example 2.2.2. Let our LFSR be given as in Figure 2.2.

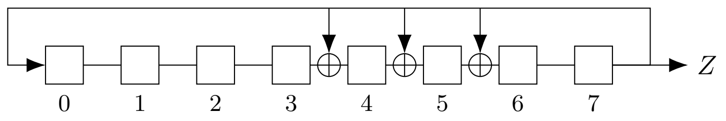


Figure 2.2: LFSR on 8 bits

Given 8 subsequent keystream bits `01011010`, we can replicate the initial state and any state leading up to and from the current state.

We start by writing the initial state $s^0 = (s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7)$. Since $z_0 = 0$, we find that $s_7 = 0$. Then we iterate the LFSR once, to get $s^1 = (0, s_0, s_1, s_2, s_3, s_4, s_5, s_6)$. Then since $z_1 = 1$, we find that $s_6 = 1$.

We repeat this process in Table 2.1.

s^0	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	$s_7 = z_0 = 0$
s^1	0	s_0	s_1	s_2	s_3	s_4	s_5	s_6	$s_6 = z_1 = 1$
s^2	1	0	s_0	s_1	$s_2 + 1$	$s_3 + 1$	$s_4 + 1$	s_5	$s_5 = z_2 = 0$
s^3	0	1	0	s_0	s_1	$s_2 + 1$	$s_3 + 1$	$s_4 + 1$	$s_4 + 1 = z_3 = 1$
s^4	1	0	1	0	$s_0 + 1$	$s_1 + 1$	s_2	$s_3 + 1$	$s_3 + 1 = z_4 = 1$
s^5	1	1	0	1	1	s_0	s_1	s_2	$s_2 = z_5 = 0$
s^6	0	1	1	0	1	1	s_0	s_1	$s_1 = z_6 = 1$
s^7	1	0	1	1	1	0	0	s_0	$s_0 = z_7 = 0$

Table 2.1: A display of the replicating of the initial state for an LFSR.

We now immediately obtain that $s^0 = (0, 1, 0, 0, 0, 0, 1, 0)$. The current state is one iteration more: $s^8 = (0, 1, 0, 1, 1, 1, 0, 0)$. ♠

A way around this particular attack is to not use all bits, but only certain ones. For example, using only each 10th output bit for our keystream. By Kerckhoffs, this addition should be known as well, so an attacker knows that only the 10th output bits get used.

The previous attack still works, and still as easy as before. To explain this practically, we use the matrix that corresponds to the LFSR.

Example 2.2.3. Consider the LFSR given in Figure 2.3.

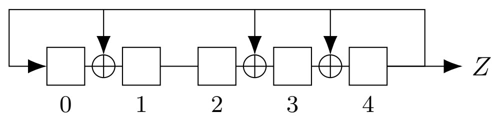


Figure 2.3: An LFSR on 5 bits

The matrix that corresponds with it is the matrix

$$M = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & 1 \\ 1 & \cdot & \cdot & \cdot & 1 \\ \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & 1 \\ \cdot & \cdot & \cdot & 1 & 1 \end{pmatrix}.$$

Remember that we have $s^{t+1} = M \cdot s^t$.

As the attacker, we get the following keystream 01110 that corresponds to the outputs of the LFSR after iterations $0, 5, 11, 17, 29$.

In other words $(z_0, z_5, z_{11}, z_{17}, z_{29}) = (0, 1, 1, 1, 0)$.

Since at an iteration, the last bit becomes an output bit, we know that z_i is the last row of $M^i \cdot s^0$ with s^0 being the initial state.

Denoting m_i for the last row of M^i we immediately get the following system of equations:

$$\begin{cases} m_0 \cdot s^0 = 0; \\ m_5 \cdot s^0 = 1; \\ m_{11} \cdot s^0 = 1; \\ m_{17} \cdot s^0 = 1; \\ m_{29} \cdot s^0 = 0. \end{cases}$$

If we write N for the matrix consisting of these m_i we have

$$N = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & 1 \\ 1 & \cdot & 1 & \cdot & 1 \\ \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & 1 & 1 & 1 \\ 1 & 1 & \cdot & \cdot & \cdot \end{pmatrix}.$$

Solving the system $N \cdot s^0 = (0,1,1,1,0)$ then gives the desired initial state options, namely $(1,1,0,1,0)$. ♠

2.3 Filtered LFSRs

The attacks of Section 2.2.2 pose a general problem. A cipher consisting of only linear functions can be analyzed and broken with linear algebra. This implies that a secure stream cipher must always contain a non-linear component.

An obvious place to include a non-linear component is by computing the keystream bit from the statebits by means of a non-linear function.

In other words, instead of using $z_t = s^t_{n-1}$ (i.e., just taking the last LFSR state bit as keystream bit), we put $z_t = f(s^t_0, \ldots, s^t_{n-1})$ for some non-linear function f.

Consider the example function of Figure 2.4. The keystream bit is one of the bits marked with a $\cdot$, so either of bits s_1 to s_{16}. The choice of the bit that will be used for the keystream is determined by the *multiplexer*. This multiplexer takes the values $(s_{18}, s_{21}, s_{24}, s_{27})$ and turns them into an integer:

$$1 + s_{18} + 2s_{21} + 4s_{24} + 8s_{27} \in [1, 16]\,.$$

This integer, finally, determines the index of the bit that will be the keystream bit.

The bits s_{18}, s_{21}, s_{24} and s_{27} are called *address* bits. So we determine $1 + s_{18} + 2s_{21} + 4s_{24} + 8s_{27} \in [1, 16]$.

In Figure 2.4, apparently, the multiplexer decides that the keystream bit will be s_5. We thus necessarily have $(s_{18} = s_{21} = s_{27} = 0$ and $s_{24} = 1$.

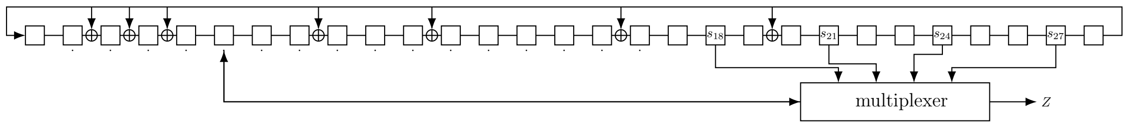


Figure 2.4: An example of a filtered LFSR

We need to explain why this multiplexer is indeed a non-linear function. For the sake of simplicity, first consider a 2-to-1 multiplexer: it takes one bit (in this case s_9) to select the keystream bit from the state. In detail, it returns s_1 if $s_9 = 0$, and s_2 if $s_9 = 1$.

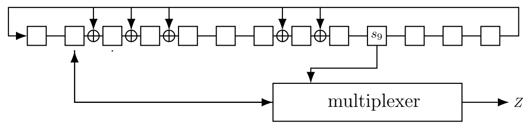


Figure 2.5: A 2-to-1 multiplexer on an LFSR

In formula, this means that the keystream bit z satisfies

$$z = f(s_0, s_1, \ldots, s_{11}, s_{12}) = (s_9 + 1)s_1 + s_9 s_2 \,.$$

This function is, indeed, non-linear. For the original example of Figure 2.4, the output function would be

$$\begin{aligned} z = f(s_0, s_1, \ldots, s_{27}, s_{28}) = {} & s_{16}s_{18}s_{21}s_{24}s_{27} + s_{14}(s_{18}+1)s_{21}s_{24}s_{27} + s_{13}s_{18}(s_{21}+1)s_{24}s_{27} \\ & + s_{12}s_{18}s_{21}(s_{24}+1)s_{27} + s_{11}(s_{18}+1)s_{21}(s_{24}+1)s_{27} \\ & + s_{10}s_{18}(s_{21}+1)(s_{24}+1)s_{27} + s_9(s_{18}+1)(s_{21}+1)(s_{24}+1)s_{27} \\ & + s_8s_{18}s_{21}s_{24}(s_{27}+1) + s_7(s_{18}+1)s_{21}s_{24}(s_{27}+1) \\ & + s_6s_{18}(s_{21}+1)s_{24}(s_{27}+1) + s_5(s_{18}+1)(s_{21}+1)s_{24}(s_{27}+1) \\ & + s_4s_{18}s_{21}(s_{24}+1)(s_{27}+1) + s_3(s_{18}+1)s_{21}(s_{24}+1)(s_{27}+1) \\ & + s_2s_{18}(s_{21}+1)(s_{24}+1)(s_{27}+1) + s_1(s_{18}+1)(s_{21}+1)(s_{24}+1)(s_{27}+1) \,. \end{aligned}$$

As we will later see, LFSRs filtered by a multiplexer can still be broken by attacks.

2.3.1 Guess-and-determine attack

The guess-and-determine attack will be included at a later date.

2.4 Combiner LFSRs

We can also achieve non-linearity is by using multiple LFSRs and combining their output in a non-linear fashion. We call these Combiner LFSRs.

Example 2.4.1. As an example, we can take two Filtered LFSRs, run both for, say 5 cycles, and obtain two keystreams of length 5. Then we can view those keystreams as integers and add them modulo 2^5 and obtain a new integer. Then view that integer as a bitsequence again and we have our keystream.

Take for example an LFSR of 61-bit length and one of length 67 bits.

Assume that the LFSR of length 61 bits gives an output `01010`, while the other gives as output `11100`.

Then the Combiner LFSR generates keystream `00110` since $10 + 28 = 38 \equiv 6 \pmod{32}$. ♠

In the above we represented a bit-string as an integer, and vice-versa. Since we will do this more often, we will define those functions properly. In these functions we adopt a little-endian convention: the first bits of a string have the highest weight in the integer.

Definition 2.4.2. We define a function that interprets a string as an non-negative integer. The function StoI: $\{0,1\}^* \to \mathbb{Z}_{\geq 0}$ is defined by

$$\mathrm{StoI}(\mathtt{b}_0\mathtt{b}_1 \ldots \mathtt{b}_{\mathtt{n}-1}) = \sum_{\mathtt{i}=0}^{\mathtt{n}-1} \mathtt{b}_\mathtt{i} 2^{\mathtt{n}-1-\mathtt{i}} \,.$$

Definition 2.4.3. We define a function that encodes an integer to an n-bit string, where n is a parameter of the function. The integer must be in the range $[0, 2^b - 1] \subset \mathbb{Z}$. The function $\mathrm{ItoS}_b \colon [0, 2^b - 1] \to \{0,1\}^b$ is defined as follows:
$\mathrm{ItoS}_n(x)$ is the unique n-bit string that yields $x = \mathrm{StoI}(b)$.

The function $\mathrm{ItoS}_b(x)$ is undefined if x is an integer out of the domain. Implementations shall ensure this does not happen or return an error.

2.4.1 Divide-and-conquer attack

The Combiner LFSR can be attacked in its own right. An attacker can guess the initial state of the 61-bit LFSR (2^{61} options) and using the keystream bits of the Combiner LFSR, reconstruct the output bits from the other LFSR.

After retrieving some output bits of the 67-bit LFSR in this way, the attacker can check if they satisfy the recurrence relation. If so, the guess for the initial state of the 61-bit LFSR is probably correct, as well as after enough iterations, one can carry on generating more and more keystream bits.

So the security from a combiner LFSR is upper bounded by the length of the shortest of the two LFSRs.

An exhaustive search for the initial states would amount to 2^{128} guesses!

2.5 Modern stream ciphers

Even if Eve cannot recover the key K and even if SC is a well-designed stream cipher, it can still only be used once for each key K, as the algorithm transforms identical keys to identical keystreams!

This problem can be resolved by developing stream ciphers that get an additional input: a *diversifier* D. Its goal is to generate multiple keystream sequences from the same key. Namely, running the stream cipher with different values (K, D) gives different keystreams Z. Informally, for a secure stream cipher these keystreams appear as mutually independent sequences of random bits.

This brings us to the following modern definition of a stream cipher. Let k be the key size and d the diversifier size. A stream cipher is now defined as follows:

$$\mathrm{SC}\colon \{0,1\}^k \times \{0,1\}^d \to \{0,1\}^* \,,\ (K, D) \mapsto Z \,. \tag{2.3}$$

For fixed key, we simply write $\mathrm{SC}_K(\cdot) = \mathrm{SC}(K, \cdot)$. Key K is the key agreed upon by Alice and Bob beforehand, and in order to transmit a message M securely, Alice takes a unique diversifier D (i.e., one that was not used before), computes

$$C = M \oplus \mathrm{SC}_K(D)\,,$$

with keystream $\mathrm{SC}_K(D)$ truncated to the length of M, and sends (D, C). Bob can decrypt the straightforward way, given that he knows the key and diversifier:

$$M = C \oplus \mathrm{SC}_K(D)\,,$$

Informally, a stream cipher SC is secure if for an adversary that does not know the key, keystreams due to one or more values of D are hard to distinguish from mutually independent sequences of random bits. At this point we are vague about what "distinguish from random" mean and how can we qualify the statement that it should be "hard". We define those notions more formally in Chapter 4.

2.6 The basis for security of cryptographic primitives

OTP encryption is provably secure, but for the generation of a keystream from a short key and diversifier there is no such proof.

So then, how can we be assured of the fact that some concrete cryptographic primitive, i.e., a certain stream cipher, is secure? The short answer is: we cannot. We can never know for sure whether a concrete cryptographic primitive is secure. On the other hand, we can have confidence in the security of a concrete cryptographic primitive. Such confidence is always based on the fact that the cryptographic primitive was published, that many researchers that are experts in cryptanalysis have tried to break it, but none succeeded.

Note that this section is not specific for stream ciphers but applies to all cryptographic primitives.

2.6.1 The open cryptographic research activity

But why would researchers spend their time in trying to break some cryptographic primitive? They do so because finding an effective or innovative attack against a cryptographic primitive can be published at some prestigious venue, and this publication in turn can help the researcher in building an academic career. This is what we call the *open cryptographic research activity.* Some researchers design a cryptographic primitive and publish it. Other researchers try to attack this primitive and publish their attack if they succeed. This is what we call cryptanalysis. If there is an effective attack, the designers may publish an update of their design, with some modifications so that the published attack does not work. Then the primitive can be attacked again, updated again, etc. At some point, the design tends to stabilize and we have a primitive that looks okay.

If a cryptographic primitive has been public for a long while, and if it has attractive features such as high efficiency, and if there have been no attacks, then we can start getting confidence in it. Another factor for confidence is when a cryptographic primitive becomes a standard. That usually means that many experts have looked at the security status of the primitive and deemed it good enough to become a standard.

2.6.2 Security claim

But what does it mean for a cryptographic primitive to be secure? This is something that is clearly defined for a cryptographic primitive if its publication is accompanied by a *security claim.* A security claim is an unambiguous statement that defines the minimum success probability an attack must have to be considered as *breaking* the primitive. In the security claim, *breaking* is well-defined and usually it means to distinguish the output of the primitive from a sequence of random bits. The minimum success probability is typically expressed in terms of the attacker effort in terms of computation (offline complexity) and data obtained from the keyed primitive (online complexity).

A security claim serves two purposes:

- For potential cryptanalists, it serves as a challenge. It is an invitation to come up with attacks that have a higher success probability as the one in the claim. If someone finds such an attack, the designer will have to admit his design is broken. He can then choose to strengthen his design or weaken the security claim.
- For potential users, it serves as a security specification. Assuming no attacks have been found that refute the claim, the user can assume there are no attacks with higher success probability than specified in the claim. Of course the confidence in the claim is based on the open cryptographic research activity: the older the claim and the more experts have tried to break the primitive, the higher the confidence.

Often designers publish a cryptographic primitive without a security claim. In that case, the default is that exhaustive key search (see Section 2.2.1) should be the most efficient way to distinguish the primitive's output from a sequence of uniformly random bits. This serves as the default security claim.

2.6.3 Expressing security strength

Often, a security claim, either challenged or assumed to be satisfied, is expressed as a mere number called the *security strength*.

If a claim for a cryptographic primitive implies that the expected effort to break it with success probability close to 1 is 2^s *units*, e.g., executions of the primitive, then we say the primitive has a claimed security strength of s bits. Clearly, due to the existence of exhaustive key search, the security strength of a cryptographic primitive taking a k-bit key is at most $k - 1$ bits.

2.6.4 Provable security

As said, there are not provably secure cryptographic primitives. However, for primitives that are built in a modular way, one can prove they are secure on the condition that (one or more) underlying primitives are secure. Translated to the concept of security claims: one can prove that these primitives satisfy a certain security claim if their underlying primitives satisfy some security claim. Of course, the security claim of the underlying primitive cannot be proven but this means that cryptanalysts can limit their attention to the underlying primitives. This is a very seductive thought for many so-called *computer scientists* and it explains the popularity of block ciphers (see later).

Even more seductive are provable reductions of security claims to so-called famous mathematical problems such as factoring or the traveling salesman problem. Here, one proves that breaking the cryptographic primitive implies breaking some mathematical problem that is considered to be hard. This approach is often considered in public key cryptography.

Chapter 3

Block ciphers

We saw that the security of stream encryption breaks down when we use the same keystream for encrypting multiple messages. Modern stream ciphers address this by supplying a diversifier D as input and as long as the user takes a different value for D for each encryption with the same key, different keystream will be used. We say the diversifier D should be a *nonce*: each value should be used at most once. For security, it is sufficient for the value D to be a nonce over all encryptions with the same key K. However, in certain attack scenario's (multi-target attack, see later), it is better for security to have D be a so-called *global nonce*: each encryption with the stream cipher anytime anywhere uses a different D.

In practice, ensuring D is a (global) nonce is easy to achieve as in most systems messages or data streams already have a unique identifier (e.g. IP packages, date and time of emails, ...) and so do communicating parties (email address, IP address, URL, ...) and if these do not exist, they can easily be constructed.

However, in the past incompetent or lazy protocol designers have screwed up and there are many cases where the nonce requirement has been violated. Such violation is often addressed by the term *nonce misuse.* In the academic world it is considered an intellectual challenge to design cryptographic primitives that are robust against misuse. This means: the reduction of security under misuse should be as small as possible.

Another property of stream encryption that is perceived as a disadvantage by badly educated people is the following. Eve can flip a bit of the plaintext that Bob will see after decryption (say bit i of P) simply by flipping the corresponding bit of the ciphertext (bit i of C). This is because each of the (deciphered) plaintext only depends on a single bit of the ciphertext. If the message contains some kind of financial transaction, Eve could modify the amount and commit fraud. So Eve may intercept a cryptogram (D, C) coming from Alice, flip a certain bit of C, and transmit the updated cryptogram to Bob. Bob would not be able to notice that Eve has altered the cryptogram! The problem with this argument is that it is based on wrong expectations. One expects encryption to protect the integrity of the transmitted message. Encryption was never meant to protect integrity and it rarely does. For protecting integrity, one needs to compute a cryptographic checksum over the message or ciphertext before sending and include it in the cryptogram. Such a cryptographic checksum is often called a message authentication code (MAC) or a tag.

Anyway, robustness against misuse and protection of message integrity by encryption can be realized up to some point by another type of encryption that we will introduce in the next section.

3.1 Wide block ciphers

In stream encryption each ciphertext bit depends only on a single plaintext bit and vice versa. At the other end of the spectrum, one may consider an encryption scheme that has *each bit of cryptogram C to depend on each bit of message M* and vice versa. A cryptographic primitive that realizes this for arbitrary-length messages M is called a *wide block cipher* (flexible block cipher would have been a better name).

A wide block cipher WB takes as input a fixed-length key K and an arbitrarily long message M and transforms it to a ciphertext C of the same length as M:

$$C = \mathrm{WB}_K(M)\,. \tag{3.1}$$

The function must be injective for each key: if K is fixed, one should be able to decipher the ciphertext back to the plaintext. Basically, WB is a variable-length permutation. Informally, a wide block cipher is secure if with respect to an adversary that does not know the key K, WB_K maps different plaintexts to different ciphertexts that are seemingly unrelated and random, and vice versa.

Our wide block cipher WB has no diversifier. There is no need as the cryptograms corresponding to different messages look completely unrelated and hence no information is leaked. The only remaining leakage is that two messages with the same content will result in equal cryptograms if the same key K is used for the encryption.

Moreover, wide block encryption can be used to provide integrity protection of individual messages. The idea is that Alice adds some verifiable redundancy to the input of the wide block cipher before encryption and Bob can verify the presence of this redundancy after decryption. If Eve modified the cryptogram C to C', it is unlikely C' will decipher to a plaintext with valid redundancy. Of course this does not exclude Eve substituting a cryptogram C by an earlier cryptogram that Alice sent, a so-called replay attack.

A simple way to implement this is as follows. Alice constructs the input to the wide block cipher as the concatenation of the message M and a τ-bit string of zeros:

$$C = \mathrm{WB}_K(M \| \underbrace{0 \ldots 0}_{\tau})\,,$$

This means that the cryptogram C is of length $|M| + \tau$. Upon receipt of the cryptogram C, Bob computes

$$\mathrm{WB}^{-1}[K](C)\,.$$

If the resulting value *does not end* with τ zeros, this was not a message sent by Alice and Bob can discard it as fake news. For Eve to forge a message she must come up with a cryptogram C' that decrypts to a value that ends with τ zeros. She succeeds in this with probability around $1/2^\tau$. This success probability can be made arbitrarily small by increasing τ.

3.2 Block ciphers

Clearly, a secure wide block cipher would be a very powerful primitive. Unfortunately, there exists no wide block cipher with high security assurance. In reality, we are stuck with primitives that are a limited variant of wide block ciphers, namely, block ciphers with fixed block length. These are simply called *block ciphers*.

A block cipher B is a cryptographic primitive that takes as input a k-bit key K and a b-bit plaintext P and transforms it to a b-bit ciphertext C.

$$\mathrm{B}\colon \{0,1\}^k \times \{0,1\}^b \to \{0,1\}^b\,,\ (K,P) \mapsto C\,. \tag{3.2}$$

For a fixed key K, the mapping from P to C is invertible. In other words, it is a permutation. We denote this permutation by B_K, and its inverse by $P = \mathrm{B}_K^{-1}$.

Informally, a block cipher is secure if with respect to an adversary that does not know the key K, B_K maps different plaintexts to different ciphertexts that are seemingly unrelated and random, and vice versa.

Well-known examples of block ciphers are DES (with $k = 56$ and $b = 64$) and AES-128 (with $b = 128$ and $k = 128$), AES-192 (with $b = 128$ and $k = 192$) and AES-256 (with $b = 128$ and $k = 256$). A fixed-length block cipher is excellent for encrypting messages of b bits. For longer and shorter messages however it requires the application of error-prone message manipulation and padding and the advantages of wide block encryption almost fully evaporate. Block ciphers are no complete failure though as fortunately they can also be used to build stream ciphers and MAC functions that are somewhat efficient. These ways of using block ciphers are called *modes of use* and we treat them in Chapter 5.

3.3 Substitution-permutation networks

In a good block cipher each output bit depends on all input bits (diffusion) in a complicated way (confusion) and vice versa. The oldest block ciphers were substitution-permutation networks (SPN). These block ciphers achieve the goal by alternating two types of transformations: substitution that mixes bits of the state that are near each other and transposition that moves these bits away from each other. We illustrate such an SPN in Figure 3.1.

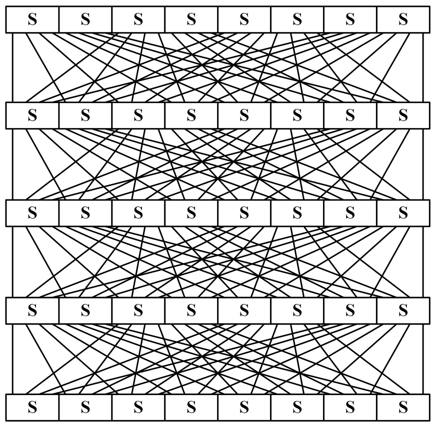


Figure 3.1: The SPN structure.

We give some more information about the two types of layers in the following subsections.

3.3.1 Substitution layers

A substitution layer partitions the bits of the state in a sequence of bytes, or nibbles (4-bit sub-blocks), or substrings of another length. For simplicity of description, we will just speak of bytes in the following but all explanations are also valid for other sub-block sizes. It applies to each of these bytes a substitution: it replaces the value of each byte by another value, according to carefully constructed lookup tables. These lookup tables are called S-boxes. It may be the case that different S-boxes are used for different byte positions. Modern ciphers apply the same S-box for all positions. For resistance to cryptanalysis, these S-boxes must satisfy non-linearity and diffusion criteria. Non-linearity criteria deal with the complexity of algebraic description of the S-box and the difficulty to predict the output of the S-box if only part of the input is known. Diffusion criteria deal with the dependence of output bits on input bits. Preferably, each output bit depends on a high number of input bits. Before the wide trail strategy, people believed that the cryptographic security of a block cipher resides in its S-boxes. They were seen as mini block ciphers and this view is still adopted by part of the research community. Note that to allow decryption, the S-boxes must be invertible in a classical SPN.

3.3.2 Permutation layers

A permutation layer moves bits from each byte to different bytes. To not confuse it with an invertible transformation, we will denote it by the term *transposition*. In the substitution layer the bits only interact within each byte and the presence of the transposition layer makes that bits will interact with bits of other bytes in the next round. The alternation of substitution and transposition should ensure that each bit of the output depends on a complicated way of all the input bits, if enough rounds are taken.

3.3.3 Where does the key come in?

In this description of the SPN the key does not interfere. In some older (pre-DES) designs, the S-boxes depended on the cipher key. Another way is to add a third layer that (bitwise) adds a round key. This is the option chosen in more recent SPN block ciphers such as Present or Gift. These round keys are derived from the cipher key by means of a so-called *key schedule*.

3.4 Data Encryption Standard

The National Institute for Standardization and Technology (NIST) is part of the Department of Commerce in the Government of the United States of America and is responsible for security standards in the US Government administration. In the '70s NIST noticed that there was a need to encrypt data that was greater than before. In particular, the US government, banks and industry all expressed a need for an encryption standard. After consulting with the National Security Agency (NSA) requests for proposals for ciphers were issued. Eventually IBM submitted a cipher that was consequently "improved" in collaboration with NSA and that was adopted as a standard by NIST. This is the Date Encryption Standard (DES).

3.4.1 Internals of DES

As most block ciphers, DES has two main parts: a data path and a key schedule. See Figure 3.2. The data path consists of the repeated application of a round function that takes a round key. The key schedule generates these round keys from the cipher key, also typically in an iterated manner. We illustrate this in Figure 3.3.

Data path in DES The data path of DES has a Feistel structure. We illustrate this in Figure 3.4. In the Feistel structure, the state is split in two, a left half L_i and a right half R_i. Here i denotes the round number. A round consists of two steps, that are involutions:

- The application of a keyed function F to the right half and the bitwise addition of the result to the left part The function F takes as arguments the round key RK_i and the right half : $L_i \leftarrow L_i + F(\mathrm{RK}_i, R_i)$.

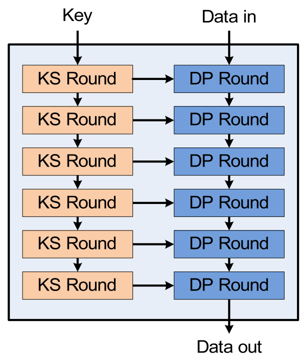


Figure 3.2: General structure of block ciphers.

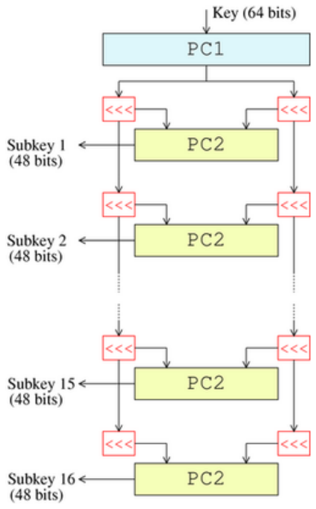


Figure 3.3: DES key schedule.

- The swapping of left and right half

DES has 16 such rounds and in the last round the swap is omitted. Hence it consists of 16 F-steps alternated with 15 swap steps. As each of these 31 steps is an involution, the inverse of DES also consists of 16 F-steps alternated with 15 swaps. The only difference between DES and its inverse is the order of the round keys. DES starts with RK_1 and ends with RK_{16} and the inverse of DES starts with RK_{16} and ends with RK_1

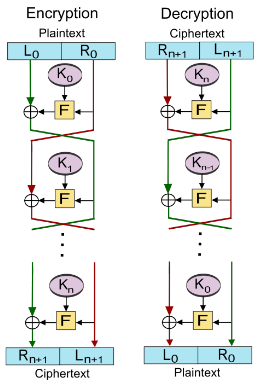


Figure 3.4: The Feistel structure.

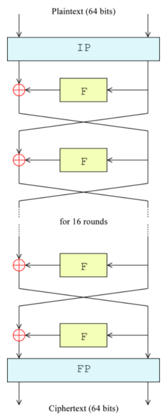


Figure 3.5: The Feistel structure in DES.

The data path of DES has two additional permutations, the Initial Permutation (IP) and the Final Permutation (FP), see Figure 3.5. These do not add to the cryptographic security and are probably a consequence of sloppiness on behalf of someone at IBM or NSA.

The function F in the Feistel structure for DES reminds of an SPN round function and is depicted in Figure 3.6. It has four layers:

Expansion The 32 input bits are expanded to a 48-bit string by duplicating the bits in positions $1, 4, 5, 8, 9, \ldots$.

Round key addition A 48-bit round key is bitwise added to the 48-bit string.

Substitution The 48-bit string is split in 8 blocks of 6 bits and to each of the bits an S-box is applied. These S-boxes have a 4-bit output and are all different.

Permutation the resulting 32 bits undergo a bit transposition.

There is not much documentation why the F looks the way it does. It looks like something engineers would end up with starting from a function F consisting of 32-bit round key addition, a layer of 4-bit S-boxes and a 32-bit permutation and finding out this does not offer sufficient resistance against some attacks they tried out. But this is of course speculation. Note that there is no requirement for F to be invertible.

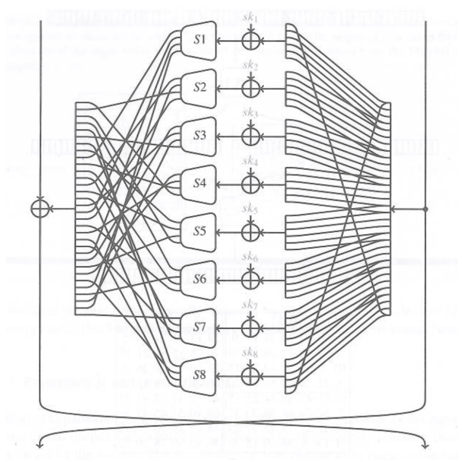


Figure 3.6: The F-function in DES.

Key schedule in DES Figure 3.3 suggests that the length of the key in DES is 64 bits. However, 8 of these bits are not used and the effective length of the cipher key is only 56 bits. In particular, in permuted choice 1 (PC1) 8 of the key bits are thrown away and the remaining 56 bits are split into two strings of 28 bits: the *left half* and the *right half*. Then these two halves enter a procedure that performs the following processing for each round:

- The two 28-bit halves are cyclically shifted over one or two positions (denoted by $\lll$). The number of positions depends on the round number.
- From each half, permuted choice 2 (PC2) selects 24 bits. These 48 bits form the round key that is applied in the F function of the current round.

As PC1 and PC2 are just selecting bits from their input, every round key bit can be traced back to a particular bit of the cipher key.

3.4.2 Some structural weaknesses of DES

Weak keys If the cipher key consists 56 zeroes, all round key bits are all-zero 48-bit strings. Similarly, an all-1 cipher key results in all round keys being all-1 strings.

As the only difference between encryption and decryption is the order of the round keys, for those cipher keys encryption and decryption are the same. In other words, for those cipher keys DES is an involution. A ciphertext can be decrypted by just encrypting it again. This is a property not present in a random permutation and hence a weakness that may be exploited. The all-1 keys and all-0 keys are therefore called *weak key*.

There also exist cipher key pairs K_1, K_2 that satisfy $(\mathrm{DES}_{K_2} \circ \mathrm{DES}_{K_1})(P) = P$ for all plaintexts P. Key pairs like these are called *semi-weak* and DES has 6 of those.

Complementation property The operation of complementing is changing all bits that are 0 to 1 and vice versa. Alternatively, it is the bitwise addition of an all-1 string. We denote complementation of a string X by $\overline{X}$.

We will now investigate what happens if, for a given plaintext P and cipher key K, we complement all their bits.

Let L_0 and R_0 the left and right half of the plaintext P. The the left and right half of $\overline{P}$ are $\overline{L_0}$ and $\overline{R_0}$

The right half $\overline{R_0}$ now enters the F-function of the first round, you can follow in Figure 3.6.

Let the output of the expansion layer for plaintext P be denoted as X. As each bit of X is just a bit of R_0, complementing R_0 just complements all bits in X. So for plaintext $\overline{P}$ we have $\overline{X}$ at the output of the expansion. Then round key RK_1 is added. For the cipher execution with $\overline{P}$ and $\overline{K}$, this round key is just the complement of the round key RK_1. This is because each round key bit can be traced back to a cipher key bit. Let the input of the S-box layer for plaintext P and cipher key K be denoted as Y. Then this input for $\overline{P}$ and $\overline{K}$ is $\overline{X} + \overline{\mathrm{RK}_1} = X + \mathrm{RK}_1$. It follows that for both cipher executions, that for P and K and that for $\overline{P}$ and $\overline{K}$, the input to the S-box layer is the same. Hence, the output of the F-function is the same for both cipher executions. The result is that after the F-step, the intermediate result for $\overline{P}$ and $\overline{K}$ is just the complement from that for P and K. This is naturally still the case after the swap step. This reasoning can be followed for all rounds, and in the end we see that the ciphertext we obtain for $\overline{P}$ enciphered with $\overline{K}$ is simply the ciphertext we obtain for P encrypted with K, complemented. More formally:

$$\mathrm{DES}_K(P) = C \iff \mathrm{DES}_{\overline{K}}(\overline{P}) = \overline{C}.$$

The complementation property can be exploited to speed up exhaustive key search by a factor 2:

Online phase The adversary obtains the ciphertexts of a plaintext P and its complement $\overline{P}$: $C = \mathrm{DES}_K(P)$ and $C^* = \mathrm{DES}_K(\overline{P})$.

Offline phase The adversary makes guesses K' for the key and deciphers P with it: $C' = \mathrm{DES}_{K'}(P)$ until $C' = C$ or $C' = C^*$. If $C' = C$, then it is very likely that $K = K'$. If $C' = C^*$, then it is very likely that $K = \overline{K'}$.

Clearly, for this speedup to work, the adversary must be able to get the ciphertext corresponding to two plaintexts that are each others complement.

3.5 Statistical attacks on block ciphers

In this section we introduce statistical attacks on block ciphers that shape modern block ciphers in that the resistance against these types of attack are main design criterion. Such attacks have the following ingredients:

- The attacker can obtain a (typically large) number of plaintext-ciphertext couples $C^{(i)} = \mathrm{B}_K(P^{(i)})$. In some statistical attacks the concrete values of $P^{(i)}$ do not matter, in others the adversary must carefully choose the plaintexts $P^{(i)}$.
- Let $a^{(i)}$ be an intermediate computation result in the encryption of $P^{(i)}$ to $C^{(i)}$. We can see the mapping from P to a as a reduced-round and truncated version of the block cipher.

- It is important that $a^{(i)}$ can be computed from $C^{(i)}$ (this could also be $P^{(i)}$) and k_{a}, with k_{a} a relatively short substring of a round key.
- The mapping from $P^{(i)}$ to $a^{(i)}$ shall have for (almost) all cipher keys K some distinguishing property that can be observed if a large number of couples $(P^{(i)}, a^{(i)})$ are obtained. An example of such a property would be that the bitwise sum of all bits of a equals the first bit of P with probability $\frac{1+2^{-10}}{2}$. In a random permutation, this probability would be $\frac{1}{2}$. Since $\frac{1+2^{-10}}{2} > \frac{1}{2}$, this is a distinguishing property.
- Computing the values of $a^{(i)}$ from $C^{(i)}$ with the correct value of k_{a} corresponds with peeling of the last round(s). The obtained couples will exhibit the distinguishing property.
- If an incorrect hypothesis is made for k_{a}, the obtained values of $a^{(i)}$ are incorrect. Instead of peeling off the last round(s), one extends the block cipher with some sort of round(s). The resulting mapping will not exhibit the distinguishing property, or with much less strength. For this to be the case, $a^{(i)}$ shall depend in a relatively complicated way on $C^{(i)}$ and k_{a}.

- Clearly, in the offline phase the adversary must just try all hypotheses for k_{a} and will choose the one for which the couples $(P^{(i)}, a^{(i)})$ exhibit the distinguishing property.

We depict the concept of statistical attacks in Figure 3.7.

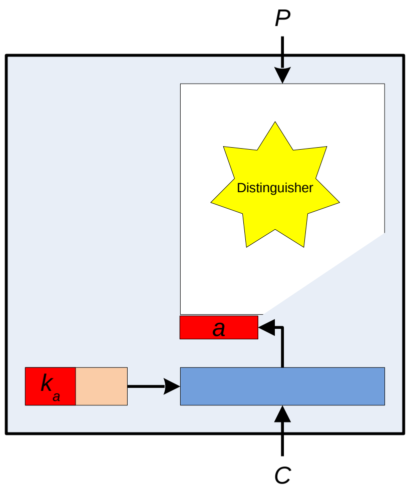


Figure 3.7: Diagram of statistical attack.

3.5.1 Differential cryptanalysis

Differential cryptanalysis is a statistical attack where the distinguisher is a difference propagation with relatively high probability.

Given a random pair of plaintexts P and P^* with difference Δ_{p}, the corresponding intermediate values a and a^* have difference Δ_{a} with some probability . Here $\mathrm{DP}(\Delta_{\mathrm{p}}, \Delta_{\mathrm{a}})$ is relatively high and can be considered independent from the cipher key K. This is called a high-probability differential.

A high-probability differential can be exploited as follows in an attack:

Online phase The adversary collects many pairs of plaintext-ciphertext couples: $C^{(i)} = \mathrm{B}_K(P^{(i)})$ and $C^{*(i)} = \mathrm{B}_K(P^{(i)} + \Delta_\mathrm{p})$.

Offline phase For all possible values of k_a, the adversary computes the corresponding values of $a^{(i)}$ and $a^{*(i)}$ and checks wether $a^{(i)} + a^{*(i)}$ has the value Δ_a for a fraction $\mathrm{DP}(\Delta_\mathrm{p}, \Delta_\mathrm{a})$ of the couples. This will be the case for the correct guess of k_a.

For this attack to work, the total number of plaintext-ciphertext couples, i.e., the data complexity of the attack must be above $1/\mathrm{DP}(\Delta_\mathrm{p}, \Delta_\mathrm{a})$. It can be shown that the attack can be conducted if the data complexity is only a very small factor above $1/\mathrm{DP}(\Delta_\mathrm{p}, \Delta_\mathrm{a})$.

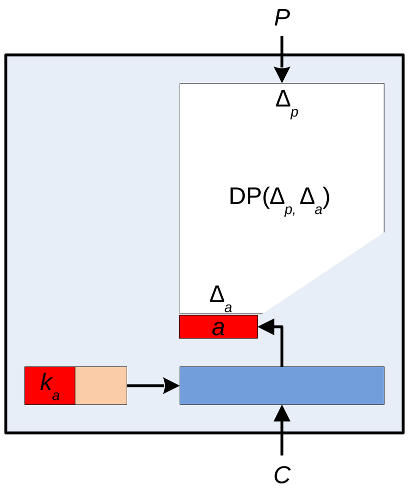


Figure 3.8: Differential cryptanalysis is a statistical attack.

3.5.2 Differential cryptanalysis on DES

Differential cryptanalysis was first published by Biham and Shamir in 1990 as an attack on DES. Their attack is based on a differential with $\mathrm{DP}(\Delta_\mathrm{p}, \Delta_\mathrm{a}) = 2^{-47}$, where a is the input to the 14-th round. The authors found this differential by tracing the propagation of a difference through the rounds. They consider a difference Δ_p at the input of the first round and determine the differences Δ_1 at the output of the first round that have the highest probabilities to occur given Δ_p. As the F-function is relatively simple, these probabilities are high and there are only a few such differences. Then they apply this to the second round for each of the selected Δ_1 resulting in a set of Δ_2 values, and so on for all rounds. This tree search algorithm finally results in a sequence of differences starting in Δ_p and ending in Δ_a. The probability $\mathrm{DP}(\Delta_\mathrm{p}, \Delta_\mathrm{a})$ can be approximated by the product of the probabilities of each of the steps: $\mathrm{DP}(\Delta_\mathrm{p}, \Delta_1)$, $\mathrm{DP}(\Delta_1, \Delta_2)$, $\ldots$, $\mathrm{DP}(\Delta_{13}, \Delta_\mathrm{a})$. Such a sequence of differences is called a *differential trail*.

They demonstrate that their attack can be executed with data complexity 2^{47} pairs of plaintext/ciphertext by using several sophisticated techniques. Academically speaking, this is a break of DES, since DES, with its 56-bit cipher key, is expected to offer security strength of 55 bits and this attack has (data) complexity 2^{47}, implying only 47 bits of security is offered. This reasoning sweeps the distinction between data and computational complexity under the carpet.

The attack of Biham and Shamir is impractical in almost all thinkable use cases as the adversary must collect 1000 Terabytes of chosen plaintext/ciphertext pairs and hence forms no threat in the real world. However, differential cryptanalysis provides a criterion on how to design a block ciphers. After the publication of Biham and Shamir there is consensus that a block cipher shall not exhibit differentials with high probability.

The attack of Biham and Shamir gives evidence that the designers of DES (IBM and NSA) already had partial knowledge of differential cryptanalysis and as they could have done an even worse job in their choice of S-boxes and the way they are embedded in the cipher structure.

3.5.3 Linear cryptanalysis

Linear cryptanalysis is a statistical attack where the distinguisher is a relatively high correlation between the plaintext and an intermediate value.

Definition 3.5.1. Given a sequence of binary variables $a = a_0 a_1 \dots a_{n-1}$ (typically the plaintext or some intermediate encryption result) and a *mask* $u = u_0 u_1 \dots u_{n-1}$ that is a sequence of bits of the same length as a determines a linear Boolean function of a as follows:

$$\sum_{i=0}^{n-1} u_i a_i \,.$$

So this is the bitwise sum of the bits of a in positions where there is a 1 in u occurs are present in our function.

We will mark masks with a subscript to identify the bitstring they are applied to, e.g., a mask applied to a bitstring x will be denoted as u_{x}.

Given a plaintext mask u_{P} and an intermediate value mask u_{a}, we can determine how much correlation occurs between the two corresponding linear Boolean functions.

The formula for correlation between two Boolean functions is given by

$$\mathrm{Corr}(f, g) = 2 \cdot \Pr(f(x) = g(x)) - 1 \in [-1, 1].$$

Note that correlation has a sign but the most important in linear cryptanalysis is its amplitude. When we instantiate f and g with linear Boolean functions defined by masks, we abuse this notation somewhat and we write $\mathrm{Corr}(u_{\mathrm{P}}, u_{\mathrm{a}})$.

Example 3.5.2. Let a be a bitstring of 5 bits and $u_{\mathrm{a}} = \texttt{10001}$, then the corresponding Boolean function is $a_0 + a_4$. Also let P be a bitstring of 8 bits and let $u_{\mathrm{p}} = \texttt{00101000}$, then the corresponding function is $p_2 + p_4$. The correlation of those two (for some fixed cipher key) is the probability that these two Boolean functions are equal times two, and then subtract 1. ♠

Linear cryptanalysis exploits the existence of couples of masks u_{P} and u_{a} such that $\mathrm{Corr}(u_{\mathrm{P}}, u_{\mathrm{a}})$ has a high amplitude for (almost) all cipher keys.

A high correlation can be exploited as follows in an attack:

Online phase The adversary collects many pairs of plaintext-ciphertexts: $C^{(i)} = \mathrm{B}_K(P^{(i)})$.

Offline phase For all possible values of k_{a}, the adversary computes the corresponding values of $a^{(i)}$ and checks wether the correlation observed in the obtained values matches the one predicted by the high correlation property. This will be the case for the correct guess of k_{a}.

For this attack to work, the total number of plaintext-ciphertext couples, i.e., the data complexity of the attack, must be above $1/\mathrm{Corr}^2(u_{\mathrm{a}}, u_{\mathrm{a}})$. It can be shown that the attack can be conducted if the data complexity is only a very small factor above $1/\mathrm{Corr}^2(u_{\mathrm{p}}, u_{\mathrm{a}})$.

Linear cryptanalysis is given schematically in Figure 3.9, where instead of $\mathrm{Corr}^2(u_{\mathrm{p}}, u_{\mathrm{a}})$ we have $C^2(u_{\mathrm{p}}, u_{\mathrm{a}})$.

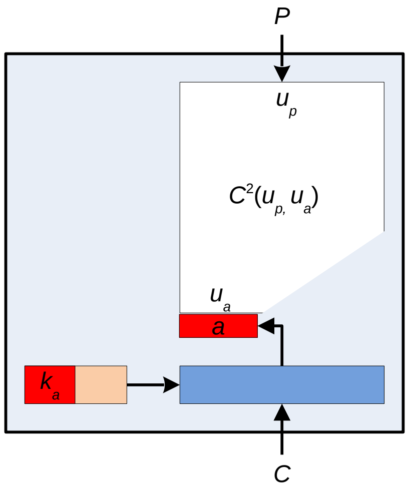


Figure 3.9: Linear cryptanalysis is a statistical attack.

3.5.4 Linear cryptanalysis on DES

In 1992, Matsui performed this statistical attack on DES, exploiting a correlation of $\mathrm{Corr}(u_\mathrm{p}, u_\mathrm{a}) = \pm 2^{-21}$. The author found this correlation by tracing the propagation of a correlation through the rounds. One considers a mask u_p at the input of the first round and determine the masks u_1 at the output of the first round that have the highest correlations with the sum of bits selected by u_p. As the F-function is relatively simple, these correlations are high and there are only a few such masks. Then this can be applied to the second round for each of the selected u_1 resulting in a set of u_2 values, and so on for all rounds. This tree search algorithm finally results in a sequence of masks starting in u_p and ending in u_a. The correlation $\mathrm{Corr}(u_\mathrm{p}, u_\mathrm{a})$ can be approximated by the product of the correlations of each of the steps: $\mathrm{Corr}(u_\mathrm{p}, u_1)$, $\mathrm{Corr}(u_1, u_2)$, $\ldots$, $\mathrm{Corr}(u_{13}, u_\mathrm{a})$. Such a sequence of differences is called a *linear trail*.

Matsui showed that his attack requires about 2^{43} known plaintexts/ciphertext pairs. This is an improvement of a factor 16 over differential cryptanalysis of DES. Moreover, it is known plaintext rather than chosen plaintext. Still, to pull it off in the real world would be hard.

Next to differential cryptanalysis, now linear cryptanalysis is something to consider when designing new block ciphers.

In current symmetric cryptography design, the resistance against differential and linear cryptanalysis is the basis. In particular, the criterion is the absence of high-probability differentials and high input-output correlations, for the block cipher reduced by a few rounds.

3.6 Triple-DES

3.6.1 The short key problem of DES

DES has a 56-bit cipher key, hence exhaustive key search takes on average 2^{55} key guesses and a DES execution for each of these guesses. Exploiting the complementation property, this reduces to $2^{54} \approx 1.8 \times 10^{16}$.

In 1997, it took 10.000 workstations that could do 500.000 trials per second to crack the DES challenge by RSA labs in 2,5 months. This was without exploiting the complementation property. The total complexity is very close to the expected one assuming 2^{55} key guesses.

In 2008 a machine was built with a lot of FPGA chips that could be programmed for exhaustive key search, called Copacobana Rivyera. This machine had a cost of 10.000$ and finds a DES key in less than a day.

According to Moore's law, a machine of that cost could now find a DES key in less than 10 minutes. Clearly, in 2019 56 bits of security is not enough.

Moore's law dictates that every two years, the number of transistors in an integrated circuit doubles, hence for the same cost, twice as many key guesses for DES can be tested.

So every two years, the amount of time it takes to find a DES key, is halved.

3.6.2 Tripling DES

A simple idea to fix this would be to apply DES twice on each plaintext block, using independent cipher keys K_1 and K_2. The resulting block cipher is called Double-DES and it has a key length of 112 bits. Due to a special type of divide-and-conquer attack, it offers much less security. Often, one says that it only offers 56 bits of security, but in practice double DES would be sufficient to offer protection against most adversaries. Only adversaries such as Google or NSA would have the resources to actually pull it off.

The attack on Double-DES is called a meet-in-the-middle attack and it works as follows:

Online phase The adversary obtains a few plaintext ciphertext pairs under the target key (K_1, K_2): $C = \text{DES}_{K_2}(\text{DES}_{K_1}(P))$ and $C' = \text{DES}_{K_2}(\text{DES}_{K_1}(P'))$.

Offline phase The adversary builds two tables. A first table, denoted $\mathcal{A}$, contains couples (K, A) with $A = \text{DES}_K(P)$ for all 2^{56} values of K and a second table, denoted $\mathcal{B}$ contains all couples (K', B) with $B = \text{DES}^{-1}_{K'}(C)$ for all 2^{56} values of K'. It then finds collisions between those tables: couples $(K, A) \in \mathcal{A}$ and $(K', B) \in \mathcal{B}$ that satisfy $A = B$. The corresponding values K and K' are candidates for the target key. The expected number of collisions is close to $2^{2\times 56}/2^{64} = 2^{48}$ and only one collision will correspond to the correct key. For each collision, the obtained couple (K, K') can be tested using the second plaintext ciphertext pair by checking whether $C' = \text{DES}_{K'}(\text{DES}_K(P'))$. This check is likely to eliminate all incorrect key candidates.

Usually, the computational complexity of this attack is approximated to around 2^{57} DES computations. Finding collisions between the two tables is actually expected to take more effort than that as the cost of memory access grows with the size of tables, and probably the most efficient way to find collision implies sorting both tables, the first according to A and the second according to B. The expected effort for sorting two tables with 2^{56} entries of each 15 bytes would take an effort that is orders of magnitude higher than doing 2^{57} DES computations. In other words, while doing 2^{56} DES computations is quite feasible nowadays, storing and sorting the 2 million Terabytes of tables required by this attack is still a challenge.

A simple improvement over Double DES, avoiding this meet-in-the-middle attack is Triple-DES. Triple-DES takes three 56-bit keys: K_1, K_2 and K_3 and it encrypts a plaintext P to a ciphertext C by first applying DES with K_1, then DES^{-1} with K_2 and then DES with K_3. We illustrate Triple-DES in Figure 3.10.

There are three flavors of Triple-DES, depending on how the keys are chosen.

The first option, and the only one that is endorsed by NIST, is to take three independent keys for K_1, K_2 and K_3. This gives rise to a key length of 168 bits. The meet-in-the-middle attack would in principle still work, but now one of the tables would have size 2^{112} 22-byte entries.

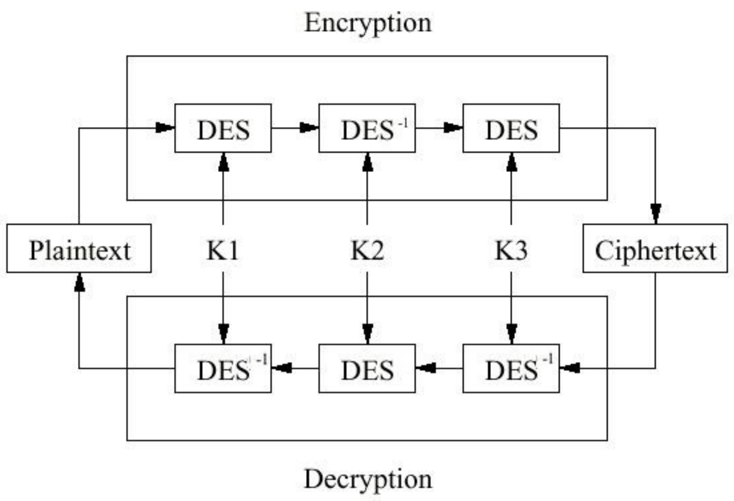


Figure 3.10: Triple-DES.

The second option takes two independent keys K_1 and K_2 and sets $K_3 = K_1$, giving rise to a key length of 112 bits. If an attacker could get hold of many plaintext-ciphertext pairs, the security decreases to 80 bits. Despite this, the two-key option is still used a lot worldwide, even by professional businesses as banks.

The third option is to takes $K_1 = K_2 = K_3$. In this case Triple-DES simplifies to just DES thanks to the fact that the middle operation in Triple-DES is DES^{-1}. As a reason for this seemingly strange choice one often gives backwards compatibility with DES, useful in making migration easier. If two nodes secure their communication with DES and key K, one can migrate first one node to Triple-DES and run it with key (K, K, K) and only later migrate the second node to Triple-DES and only then establish a 168-bit key. A more elegant solution would have been to define Triple-DES as just applying three DES in a row and use the weak key property for establishing backwards compatibility as $\text{DES}_0 \circ \text{DES}_0 \circ \text{DES}_K = \text{DES}_K$.

After having lived about 20 years with DES, even the people at NIST apparently got tired of it and they launched a contest for the selection of a successor of DES: the Advanced Encryption Standard.

3.7 The Advanced Encryption Standard (AES)

In 1998, NIST launched the AES (Advanced Encryption Standard) contest to find a successor/replacement for DES. The contest ended on 2 October 2000 when NIST officially announced that Rijndael, the submission by Joan Daemen and Vincent Rijmen, was selected as AES.

As opposed to DES, AES is still deemed secure. After extensive public scrutiny, no exploitable weaknesses have been found. The search for weaknesses in AES remains an ongoing process.

Rijndael is a family of 25 block ciphers supports all combinations of block lengths and key lengths that are a multiple of 32 bits with a minimum of 128 bits and a maximum of 256 bits. AES is a subset of three block ciphers: AES-128, AES-192 and AES-256. All three have a block length of 128 bits and they have cipher keys of 128, 192 and 256 bits respectively.

3.7.1 Underlying design principles

The main design principle underlying Rijndael are the following:

Simplicity Do not add complexity unless there is a demonstrated need for it. The aim is to have a design that is secure against known attacks, without introducing new vulnerabilities.

Modularity Compose the design of different building blocks, or steps, each with its own dedicated purpose. Select building blocks according to specific quantitative selection criteria.

Symmetry Assure most steps can be parallelized and act in a symmetrical way on the input. The large degree of parallelism allows to tradeoff area for speed in a flexible way in hardware implementations.

Hygiene Specify all steps in terms of operations in the same algebraic structure and assure they have relatively simple algebraic expressions. In particular, avoid mixing operations such as modular addition and bitwise addition.

An important factor in the design of Rijndael is the *wide trail strategy*. This strategy defines diffusion and nonlinearity criteria for the building blocks of the cipher to provide high resistance against differential and linear cryptanalysis in an efficient way. Wide trail ciphers differ from SPN ciphers by the presence of a dedicated *mixing layer*. In an SPN cipher a single substitution layer should introduce both non-linearity and mixing. This results in conflicting requirements for the S-boxes, often leading to a messy selection process full of compromise. In a wide trail cipher this conflict is absent as it has a dedicated layer for non-linearity and a dedicated layer for mixing. This results in clarity and in many cases in better overall performance.

3.7.2 Structure

When encrypting, the bytes of a plaintext block are mapped onto the elements of a state, the state undergoes a transformation and the elements of the state are mapped onto the bytes of a ciphertext block.

Rijndael is a key-iterated block cipher: the transformation from plaintext to ciphertext can be seen as the repeated application of an invertible round transformation, alternated with the addition of round keys. The number of rounds is denoted by N_r. An encryption consists of an initial key addition, denoted by `AddRoundKey`, followed by $N_r - 1$ applications of the transformation `Round`, and finally one application of `FinalRound`. The initial key addition and every round take as input the `State` and a round key. The round key for round i is denoted by `ExpandedKey`$[i]$, and `ExpandedKey`$[0]$ denotes the input of the initial key addition. The derivation of `ExpandedKey` from the `CipherKey` is denoted by `KeyExpansion`. A high-level description of Rijndael in pseudo-C notation is shown in Table 3.1.

```
Rijndael(State,CipherKey)
{
KeyExpansion(CipherKey,ExpandedKey);
AddRoundKey(State,ExpandedKey[0]);
for($i = 1$; $i < N_r$ ; $i \leftarrow i + 1$) Round(State,ExpandedKey[$i$]);
FinalRound(State,ExpandedKey[$N_r$]);
}
```

Table 3.1: High-level algorithm for encryption with Rijndael.

The number of rounds depends on the block length and the key length. Let $\mathrm{N_b}$ be the block length divided by 32 and $\mathrm{N_k}$ the length of the cipher key divided by 32. Then:

$$\mathrm{N_r} = \max(\mathrm{N_k}, \mathrm{N_b}) + 6. \tag{3.3}$$

The encryption and decryption algorithms of Rijndael are not the same, but do have the same structure.

3.7.3 The Round Transformation

The round transformation is denoted `Round`, and is a sequence of four invertible transformations in the following order: the non-linear layer `SubBytes`, the transposition layer `ShiftRows`, the mixing layer `MixColumns` and finally `AddRoundKey`. This is shown in Table 3.2.

The final round of the cipher is slightly different: with respect to the round transformation, the `MixColumns` transformation has been removed. It is denoted `FinalRound` and also shown in Table 3.2. This is similar to DES, where the swap is omitted in the last round, and for the same reason: that decryption is similar to encryption.

```
Round(State,ExpandedKey[i])
{
SubBytes(State);
ShiftRows(State);
MixColumns(State);
AddRoundKey(State,ExpandedKey[i]);
}


FinalRound(State,ExpandedKey[Nr])
{
SubBytes(State);
ShiftRows(State);
AddRoundKey(State,ExpandedKey[Nr]);
}
```

Table 3.2: The Rijndael round transformation.

The state is a rectangular array of elements of $\mathbb{F}_{2^8}$ of four rows and $\mathrm{N_b}$ columns. A byte with bits $b_0b_1b_2b_3b_4b_5b_6b_7$ maps to the element in $\mathbb{F}_{2^8}$ given by

$$b(x) = b_0x^7 + b_1x^6 + b_2x^5 + b_3x^4 + b_4x^3 + b_5x^2 + b_6x + b_7 \ . \tag{3.4}$$

In this representation, multiplication shall be performed modulo the irreducible polynomial $m(x)$:

$$m(x) = x^8 + x^4 + x^3 + x + 1 \ . \tag{3.5}$$

In the following, constants in $\mathbb{F}_{2^8}$ are denoted by the hexadecimal notation of the corresponding byte value. For example, `57` corresponds with bit string 01010111 and hence with the polynomial $x^6 + x^4 + x^2 + x + 1$.

We will now describe the four step functions.

`SubBytes` This is the only non-linear transformation of the round. It applies an invertible S-box, called $\mathrm{S_{RD}}$, to the elements of the state. Figure 3.11 illustrates the effect of `SubBytes` on the state.

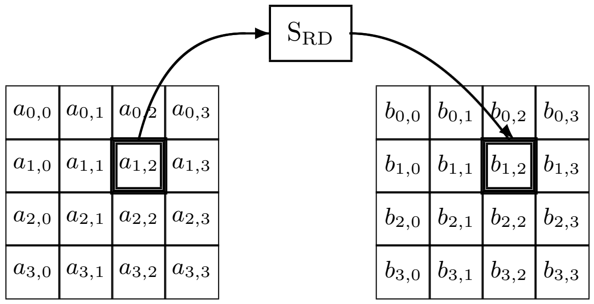

S_{RD}
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Figure 3.11: `SubBytes` acts on the individual bytes of the state.

The same S-box is used for all byte positions. This is a design choice motivated by concerns of simplicity and implementation cost. The S-box was constructed as the composition of two invertible mappings:

- Multiplicative inverse $y \leftarrow x^{-1}$ with $y \leftarrow 0$ if $x = 0$. This can be expressed as $y \leftarrow x^{2^8-2}$. This mapping has very good non-linearity properties that give, in combination with the other steps of the round function, good resistance against linear and differential cryptanalysis.
- Affine mapping: to complicate the algebraic expression without affecting the nonlinearity properties. The multiplicative inverse has a relatively simple expression that can be exploited in algebraic attacks. The introduction of this affine mapping should make these kinds of attacks ineffective.

In hardware and software the S-box can be implemented as a look-up table with 256 entries. In hardware the S-box can be implemented in a relatively compact circuit by exploiting the internal structure of the S-box.

The affine transformation L is defined by:

$$b = L(a)$$
$$\Updownarrow$$
$$\begin{pmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} = \begin{pmatrix} 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \\ 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \end{pmatrix} \times \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}. \quad (3.6)$$

The presence of the multiplicative inverse ensures that the S-box has good properties when it comes to offering resistance against the two most important types of statistical attacks:

- Differential cryptanalysis: there are no differentials $(\Delta_{\text{in}}, \Delta_{\text{out}})$ over the S-box with DP higher than 2^{-6}.
- Linear cryptanalysis: there are no input-output correlations $(u_{\text{in}}, u_{\text{out}})$ over the S-box with amplitude higher than 2^{-3}.

`ShiftRows` This is a transposition that cyclically shifts the rows of the state, each over a different offset. In particular, it moves the bytes of each column to 4 different columns. Row 0 is not shifted at all, row 1 is shifted over C_1 bytes, row 2 over C_2 bytes and row 3 over C_3 bytes. The shift offsets C_2 and C_3 depend on the block length. The different values are specified in Table 3.3. Figure 3.12 illustrates the effect of `ShiftRows` on the state.

Table 3.3: `ShiftRows`: shift offsets for different block lengths.

block length	C_0	C_1	C_2	C_3
128	0	1	2	3
160	0	1	2	3
192	0	1	2	3
224	0	1	2	4
256	0	1	3	4

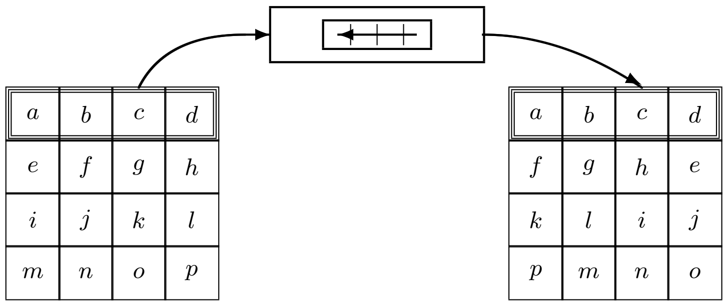


Figure 3.12: `ShiftRows` operates on the rows of the state.

`MixColumns` This is a mixing layer operating on the state column by column. The columns of the state are considered as polynomials over $\mathbb{F}_{2^8}$ and multiplied modulo x^4+1 with a fixed polynomial $c(x)$:

$$c(x) = \mathsf{03} \cdot x^3 + \mathsf{01} \cdot x^2 + \mathsf{01} \cdot x + \mathsf{02}. \tag{3.7}$$

This polynomial has been selected as one of the simplest polynomial that has a *branch number* equal to 5. The branch number is a measure that expresses the diffusion power of a mapping in the context of the wide trail strategy. As illustrated in Figure 3.13, the modular multiplication with a fixed polynomial can be written as a matrix multiplication.

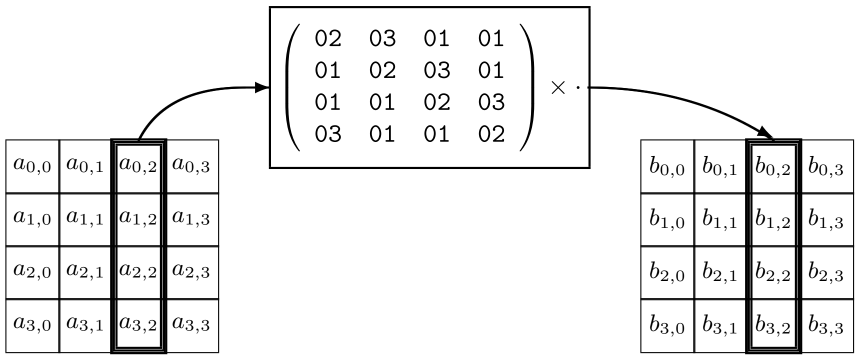

02 03 01 01
01 02 03 01
01 01 02 03
03 01 01 02
× ·
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Figure 3.13: `MixColumns` operates on the columns of the state.

The polynomial $c(x)$ is coprime to x^4+1 and therefore has an inverse modulo x^4+1. The inverse polynomial $d(x)$ is given by:

$$d(x) = \texttt{0B} \cdot x^3 + \texttt{0D} \cdot x^2 + \texttt{09} \cdot x + \texttt{0E}. \tag{3.8}$$

In hardware implementations, these linear maps can be efficiently hardwired. In software implementations, table-lookups can be used to efficiently exploit a wide range of processors. On 32-bit processors, the sequence of `SubBytes`, `ShiftRows` and `MixColumns` can be implemented by 16 table lookups.

`AddRoundKey` In this transformation, the state is modified by adding a round key to it. The addition in $\mathbb{F}_{2^8}$ corresponds with the bitwise XOR operation. The round key length is equal to the block length.

3.7.4 Key Schedule

The key schedule consists of two components: the key expansion and the round key selection. The key expansion specifies how `ExpandedKey` is derived from the cipher key.

The expanded key is a rectangular array with four rows of elements in $\mathbb{F}_{2^8}$. The key expansion function depends on the key length: there is a version for keys up to 224 bits and a version for keys longer than 224 bits. In both versions of the key expansion, the first $\mathrm{N_k}$ columns of the expanded key are filled with the cipher key. The following columns are computed recursively in terms of previously defined columns. The recursion uses the elements of the previous column, the bytes of the column $\mathrm{N_k}$ positions earlier, and round constants $\mathrm{RC}[j]$. The round constants are independent of $\mathrm{N_k}$ and defined by $\mathrm{RC}[j] = x^{j-1}$. The recursive key expansion allows on-the-fly computation of round keys on memory-constrained platforms.

3.7.5 The wide trail strategy

In the context of Rijndael, the wide trail strategy means that over four rounds, there are no differential trails with expected differential probability (EDP) above 2^{-150} and that there are no linear trails with correlation amplitude above 2^{-75}. This can be easily proven. In this section we will do this only for differential trails. The explanation for linear trails is similar, but not the same.

An r-round differential trail $Q = (\Delta_0 = \Delta_{\text{in}}, \Delta_1, \Delta_2, \ldots, \Delta_r = \Delta_{\text{out}})$ can be seen as the chaining of a number of differentials (Δ_{i-1}, Δ_i), each one over the round function. The EDP of a trail is the product of the differential probabilities of its round differentials:

$$\mathrm{EDP} = \prod_{0<i\leq r} \mathrm{DP}(\Delta_{i-1}, \Delta_i)\,.$$

To bound the DP of a round differential, we can partition its input difference Δ_{i-1} in bytes. As `SubBytes` is the first step of the round function, each of these bytes enters an S-box. The difference propagation in the different S-boxes is independent and hence the DP of the round differentials equals the product of the DP of the differentials over the individual S-boxes.

We now call the bytes in Δ_{i-i} that are zero *passive* and the ones that are non-zero *active*. A passive byte corresponds to a zero difference entering an S-box and we know that the difference will be zero at its output. An active byte corresponds to a non-zero difference and whatever the difference after the S-box, we know that the DP will be at most 2^{-6}. It follows that the DP of a round differential (Δ_{i-1}, Δ_i) is at most 2^{-6n}, with n the number of active bytes in Δ_{i-1}. It follows that the EDP of an r-round trail is at most 2^{-6n}, with n the total number of active bytes in Δ_0 up to and including Δ_{r-1}.

So clearly, a lower bound on the number of active bytes in trails implies an upper bound of their EDP.

Consider two-round trails $(\Delta_0, \Delta_1, \Delta_2)$. The number of active bytes in Δ_0 and Δ_1 imposes an upper bound on the EDP. So we can now how the number of active bytes evolves as we track a difference through the steps of a round:

- `SubBytes` does not modify the byte activity pattern: active bytes remain active and passive bytes passive.
- `ShiftRows` only moves bytes around without changing their value, so it preserves the number of active bytes.
- `MixColumns` may change the number of active bytes. However, if we define a column to be passive if all its bytes are passive and active otherwise, we see that it does not modify the column activity pattern.
- `AddRoundKey` does not modify the byte activity pattern.

So the number of active bytes in Δ_0 is the same as the number of active bytes at the input of `MixColumns` of the first round and the number of active bytes in Δ_1 is the same as the number of active bytes at the output of `MixColumns` of the first round.

Two-round trails and branch number of `MixColumns` Let us now consider an activity pattern at the input of `MixColumns` with a single active byte equal to `01` in the first position. Then the activity pattern at the output is the first column of the `MixColumns` matrix and hence has 4 active bytes. As the `MixColumns` matrix does not have zero elements, this is the case for any position of the single active byte at its input. It is even true for any value of the active byte at the input as multiplication in $\mathbb{F}_{2^8}$ of two non-zero elements gives a non-zero element.

If we try now an activity pattern at the input of `MixColumns` with a two active bytes, the number of active bytes at its output is not necessarily 4. It turns out that whatever the value and positions of those two active bytes, the number of active bytes after `MixColumns` is at least 3. Similarly, if we apply 3 active bytes at its input, its output has at least 2 active bytes. Finally, an input difference with 4 active bytes has at least 1 active byte at its output.

All in all, we see that for any non-zero difference pattern at the input of `MixColumns`, the total number of active bytes at input and output is at least 5. This is no coincidence: the `MixColumns` matrix has been especially constructed for this to be the case. For a given matrix M, the minimum number of active bytes in A and $M \times A$ is called its *branch number*. The branch number of the `MixColumns` matrix is 5. This is also the maximum possible branch number of a 4×4 matrix as an input with a single active byte can only give 4 active bytes at its output. A matrix with maximum possible branch number is called a *maximum distance separable (MDS)* matrix.

Four-round trails and optimal choice of `ShiftRows` The choice of `ShiftRows` ensures that any four-round trail has 25 active S-boxes. This can easiest be shown using figures.

The left side of Figure 3.14 depicts three rounds of AES, with `ShiftRows` of the first round removed. We can do this as `ShiftRows` and `SubBytes` commute: the latter operates on the bytes individually independent of their position and the former just switches positions. The addition of round keys does not play a role in this reasoning and we have omitted it from the figure for simplicity. The active bytes in the 4 S-box layers of this structure determine the EDP of a trail through it. If we switch `ShiftRows` and `SubBytes` in the last round, we obtain the right side of

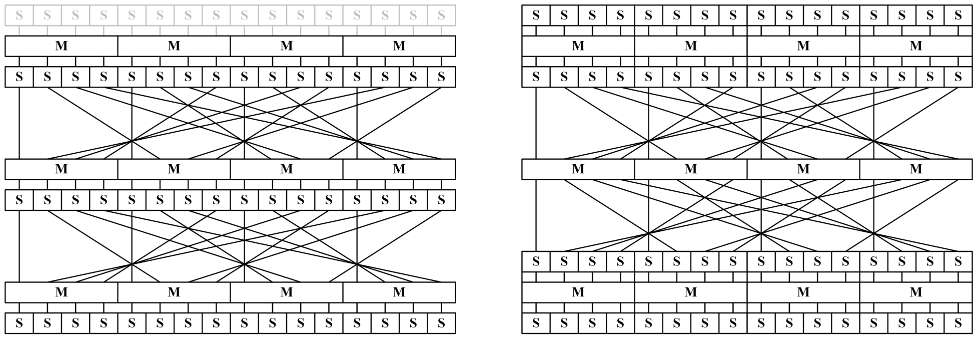


Figure 3.14: Representations of 2,5 rounds that determine the number of active bytes in a 4-round trail.

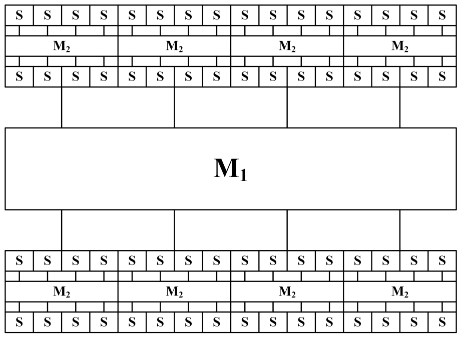


Figure 3.15: 2,5 rounds with super-`MixColumns` in the middle

Figure 3.14. We see in this figure two similar structures of `SubBytes` $\circ$ `MixColumns` $\circ$ `SubBytes` connected by a the mapping `ShiftRows` $\circ$ `MixColumns` $\circ$ `ShiftRows`. The former can be seen as the parallel application of four 32-bit mappings, operating on what we will call *columns* with some abuse of terminology. In each of these 32-bit mapping, the bytes first undergo the S-box, then the `MixColumns` matrix and then again the S-box. These mappings have become known as the AES *super-box*. The mapping between the two super-box layers is a kind of super-`MixColumns`. Upon inspection, one can see that super-`MixColumns` operates on 4 columns and has branch number 5. Namely, each instance of the `MixColumns` matrix in it takes a byte from all four input columns and outputs a byte to all four output columns. In other words, the number of active columns at its input and output is at least 5. We symbolize this in Figure 3.15.

Within a super-box, if at least one byte of its input or output is active, then thanks to the MDS property of the `MixColumns` matrix, it has at least 5 active bytes at its input and output together.

So a trail through this structure has at least 5 active super-boxes and each active super-box has at least 5 active bytes and hence any trail has at least 25 active bytes.

The link with error-correcting codes A (block) code C with *block length* n and *message length* m, with $m < n$, represents m-symbol messages with n-symbol codewords. The symbols belong to an alphabet and we denote the size of the alphabet by α. The α^m codewords form a subset of the set of all α^n n-symbol vectors. The Hamming distance between two codewords is the number of positions at which the corresponding symbols are different. The *distance* d of a code is the minimum of the Hamming distance over all pairs of its codewords. Often codes are characterized by their dimension parameters in the following notation: $[n, m, d]_\alpha$. One can now build a code using the `MixColumns` matrix M. We encode 4-byte message words A by 8-byte codewords given by $A \| M \times A$. Clearly, this is a code with distance 5, hence of type $[4, 8, 5]_{2^8}$. Hence the construction of MDS matrices can be seen as the building of an optimal error-correcting code.

3.7.6 Security

Since its submission to the AES competition in 1998, Rijndael has been cryptanalyzed by the cryptographic community and especially after it won the NIST competition, many academic papers have been published about the AES subset of Rijndael. Despite over 20 years of cryptanalysis, no exploitable weaknesses of AES have been shown.

A specifically interesting attack were the so-called biclique attacks that were published in 2011 and could be considered an academic break of AES. This is essentially meet-in-the-middle attack that uses an arsenal of sophisticated techniques to speed up exhaustive key search. Instead of having to do a full cipher execution per key guess, it allows to reduce this to a small fraction. The result is that the biclique attack could in principle be up to a factor 4 faster than straightforward exhaustive key search. There are however no practical implementation where this gain can be realized, so it remains a theoretical academic attack. At this moment it does not look like AES will be broken any day soon.

3.7.7 Implementations and cache attacks

One of the reasons why Rijndael won the AES competition is its versatility in implementation. On low-end processors, the S-box and multiplication by `02` can be implemented with a simple table-lookup and all the other operations by bitwise Boolean XOR instructions. This is also valid for dedicated hardware, where the internal structure of the S-box allows reasonable compact circuits. Moreover, the symmetry and parallelism allows a trade-off between circuit size and speed: from very compact with moderate throughput to large with ultra-high throughput.

The greatest asset of Rijndael in the AES competition when it comes to implementation was its so-called T-table implementation that allows to executing its round function in one table-lookup and 32-bit XOR instruction per byte. This was very well suited for the high end processors at the time, with their 32-bit registers.

However, a few years after the AES standard appeared, it was shown that T-table implementations leaked the key by their computation time. Modern processors store data in different types of memory that has different access speed. The closest to the central processing unit (CPU) is the memory with the highest speed. This is typically on the same chip as the CPU and therefore it has typically small size. Then there are other types of memory that are not on the CPU chip and take more time to access. CPU architecture optimizes processing time by storing the data that is used most often in the memory with highest speed: the so called *cache*. When executing T-table AES code, the T-table will preferably be stored in cache as that will give highest speed. However, on a modern computer there are typically many processes and each one will use the cache. Therefore it cannot be guaranteed that the T-tables will be in the cache all the time. As a matter of fact, at any given point in time, typically only part of the T-tables are in cache. When performing a table-lookup, the offset in the table is the value of a byte in the AES state and this value depends on a byte of a round key. Hence, the computation time of a table-lookup depends on the key. It has been shown that an adversary that can obtain timing information from a process running on the same CPU as the AES T-table code, can finally reconstruct the cipher key. This vulnerability was a big blow for AES, especially in the context of protocols such as SSL and TLS.

There were several initiatives to fix this problem. One was that of Peter Schwabe and Emilia Käsper to write efficient AES code that does not use table-lookups by using the technique of *bit-*

slicing. The other was the initiative of Intel to have dedicated AES instructions on their high-end CPUs that later spread to many other architectures.

A feature of both the bit-sliced software and the AES instructions is that they only become efficient if multiple AES encryptions can be done simultaneously. In the case of the AES instructions this is because it makes use of *pipelining*, a technique reminiscent of an assembly line in a car factory. This would have a major impact on the way block ciphers were used: where in the 20th century it was recommended to use block ciphers to do *block encryption* using so-called cipher block chaining (CBC) mode, in the 21st century they were simply used as a component to build a stream cipher, namely counter mode. These ways to use a block cipher will be treated in Chapter 5.

Chapter 4

Modeling security

In the last two chapters we have defined stream ciphers and block ciphers. In Chapter 2, we mentioned that a stream cipher is informally considered secure if its keystream strings look random. Block ciphers of Chapter 3 are likewise considered secure if they "map plaintexts to seemingly unrelated and random" ciphertexts and vice versa. This chapter will be concerned with how to formalize security of cryptographic primitives. We will take stream ciphers as a running example through Sections 4.1-4.5, and extend our formalization to block ciphers in Section 4.6.

4.1 Different types of adversary models

Consider a modern stream cipher of the form (2.3), that is instantiated with a secret key K. When we fix the key, the stream cipher becomes a function mapping a diversifier D to a keystream Z. We denote this function as $\mathrm{SC}_K : \{0,1\}^d \to \{0,1\}^*$.

There are different types of adversaries one might consider. Differences are in the amount of information that she obtains.

Recall Example 2.2.1. In this example, the adversary had some information about the data, namely that it was "meaningful" text. Having such information is a necessary condition in order to mount an attack in the first place.

Another typical case is that SC_K is used to encrypt a message M and Eve knows a small portion of the message M. For example, M is an official letter with some standard heading. In this case, we speak of a *known plaintext adversary.*

A known plaintext adversary can automate exhaustive key search with a program that systematically runs over all key values and only return those that have the standard heading in the deciphered ciphertext M'. A secure stream cipher generates seemingly independent random keystreams for different keys K, so applying (2.2) with a wrong key K' will result in a seemingly random message. If the standard heading is sufficiently long (how long must it be?), the program will only return one key guess: the correct one.

Eve's knowledge about the message may be more subtle. For example, she may only know that it is ASCII encoded English text, or just know it is the letter *when she sees it.* Here one typically speaks of a *ciphertext-only* adversary. In these cases finding the key K by exhaustive key search is still possible if the message is long enough, but it may be trickier to implement.

Eve's knowledge about the message may likewise be much more. For example, Eve may be able to choose the message herself and receive the corresponding ciphertext. Clearly, this will give her knowledge of a keystream Z corresponding to one diversifier, but one would like that it reveals nothing about keystreams from different diversifiers.

In general, when evaluating a cipher, one assumes the most powerful adversary model. If it is secure in that model, it is also secure in many weaker models.

4.2 When is an attack considered successful?

Clearly, if Eve manages to recover the key K, she can compute the keystream Z for any value of the diversifier D by means of running SC_K. However, this is not the only way for Eve to break it.

Suppose she can *predict* part of a new keystream: after having learned many (D, Z) tuples for different diversifiers, she happens to know that for a specific new diversifier $D^\star$, with high probability the keystream is equal to $Z^\star$. If Alice would use $D^\star$, then she can use this information to recover the plaintext from a ciphertext where $\mathrm{SC}_K(D^\star)$ was used as keystream. Clearly, this should also be considered as an attack.

As said, key recovery allows Eve to predict a keystream but the converse is not true. If Eve can predict part of a new keystream, she might not be able to recover the key. This means that a key recovery attack is (strictly) stronger than a keystream prediction attack.

We can weaken the attack even further. Suppose that Eve can spot certain *regularities* in a keystream. For example, after having learned many (D, Z) tuples, she can do basic statistical analysis and observe that the number of ones in all keystream sequences Z is much more than 50%. Should this be considered as a valid attack, even though it does not lead to Eve recovering the key or to recover plaintext from particular ciphertexts?

In general, we want SC_K to "look like" a function that responds randomly for each input. We will consider a theoretical function that behaves ideally in this respect: the *random oracle.*

4.3 Random oracle

A random oracle is an ideal cryptographic primitive that generates a random response to each query (and the same response for queries with the same argument). It is not practical but it can be described by an algorithm that clarifies the properties of its outputs. It accepts inputs of arbitrary size and generates infinite streams, and it is denoted "$\mathcal{RO}$".

We first give the algorithm for a random oracle that returns strings Z of fixed length and will deal with the infinite length in a later stage. $\mathcal{RO}$ maintains an archive AR that contains tuples (M, Z). We call an algorithm that keeps state between queries *stateful.* We abuse notation somewhat by saying $M \in \mathsf{AR}$ if there is a tuple (M, Z) in AR with first member M. For $M \in \mathsf{AR}$ we denote by $Z(M)$ the second member of the tuple (M, Z)

Initially, the archive AR is empty. If $\mathcal{RO}$ is queried with an input M, it integrates the query in the archive AR:

- If $M \notin \mathsf{AR}$, $\mathcal{RO}$ generates a uniformly random string Z, and stores (M, Z) in AR.

Then it returns $Z(M)$ to the query.

We wish our $\mathcal{RO}$ oracle to be usable as a stream cipher, and that implies it must be able to return an output Z of arbitrary length. Concretely, we adapt the query interface to $\mathcal{RO}$ by also including the desired length of Z, denoted by ℓ. The algorithm now becomes the following. If $\mathcal{RO}$ is queried with (M, ℓ), it integrates the query in the archive AR:

- If $M \in \mathsf{AR}$ and $Z(M)$ is shorter than ℓ bits, $\mathcal{RO}$ updates (M, Z) in the archive by appending

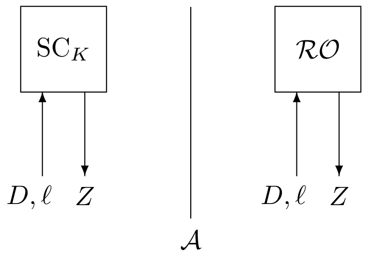


Figure 4.1: Distinguishing experiment for stream ciphers.

enough uniform random bits to Z so that its length becomes ℓ;

- If $M \notin \mathsf{AR}$, $\mathcal{RO}$ generates a uniformly random string Z and stores (M, Z) in AR.

Then it returns the first ℓ bits of $Z(M)$ to the query.

4.4 Distinguishing advantage

As explained in Section 2.6.2, we consider adversary Eve to be successful if she finds an attack that has a success probability that is higher than the one in the security claim that comes with the cryptographic primitive. This section is about how we can express such a claim. In a claim, one takes the strongest possible attacker model and if a cryptographic primitive is secure in that model, it is also secure in all weaker attacker models.

Random oracle distinguishability is modeled by the following thought experiment.

Behind the back of Eve, we secretly select either SC_K or $\mathcal{RO}$. Either function is chosen with 50% possibility: we generate a random bit $b \xleftarrow{\$} \{0, 1\}$, and select SC_K if $b = 1$ and $\mathcal{RO}$ if $b = 0$. We do not reveal our choice to Eve. However, Eve would like to know this.

To measure the success probability that Eve has in guessing b, we consider the experiment of Figure 4.1. In this figure, the adversary, Eve, is in either of two worlds: the "real world" where she speaks with SC_K and the "ideal world" where she speaks with $\mathcal{RO}$. As Eve does not know whether she is in the real or ideal world, we call the entity that is either SC_K or $\mathcal{RO}$ her "oracle". Eve can "query" her oracle. In a query she gives as input a diversifier D and requested stream length ℓ and her oracle responds with a keystream Z of ℓ bits. In the real world, this keystream satisfies $Z = \mathrm{SC}_K(D, \ell)$ and in the ideal world it satisfies $Z = \mathcal{RO}(D, \ell)$.

Eve can send a number of queries to her oracle, and eventually, she must guess whether she is in the real or in the ideal world. She does so by returning a bit $b' \in \{0, 1\}$: $b' = 1$ if she thinks she is in the real world and $b' = 0$ if she thinks she is in the ideal world.

Eve succeeds if $b' = b$, and we denote this by the event $\mathsf{success}$. Clearly, a naive adversary has a success probability of 1/2: she may simply make no queries and toss a coin to determine b'. Therefore, her goal would be to succeed in guessing b with probability significantly larger than 1/2. In other words, any adversary that succeeds with probability *significantly more than* 1/2 can distinguish. When considering adversary Eve in formal way, she (or rather, "it") is an algorithm. We denote such an adversary by $\mathcal{A}$. We now define the (probability) *distance* between the real world and the ideal world with respect to an adversary $\mathcal{A}$:

$$\Delta_{\mathcal{A}}(\mathrm{SC}_K\ ;\ \mathcal{RO}) = \mathbf{Pr}\left(\mathcal{A}^{\mathrm{SC}_K} = 1\right) - \mathbf{Pr}\left(\mathcal{A}^{\mathcal{RO}} = 1\right) . \tag{4.1}$$

In words, this distance is the difference between the probability that the adversary $\mathcal{A}$ returns $b' = 1$ in the real world minus the probability that it returns $b' = 1$ in the ideal world.

This distance is equal to two times the success probability of the adversary $\mathcal{A}$ minus 1:

$$\Delta_{\mathcal{A}}(\mathrm{SC}_K\ ;\ \mathcal{RO}) = 2\mathbf{Pr}\,(\mathsf{success}) - 1\,.$$

This can be seen by basic probability theory (for info only):

$$\begin{aligned}
\Delta_{\mathcal{A}}(\mathrm{SC}_K\ ;\ \mathcal{RO}) &= \mathbf{Pr}\left(\mathcal{A}^{\mathrm{SC}_K} = 1\right) - \mathbf{Pr}\left(\mathcal{A}^{\mathcal{RO}} = 1\right) \\
&= \mathbf{Pr}\,(b' = 1 \mid b = 1) - \mathbf{Pr}\,(b' = 1 \mid b = 0) \\
&= \mathbf{Pr}\,(b' = 1 \mid b = 1) - \big(1 - \mathbf{Pr}\,(b' = 0 \mid b = 0)\big) \\
&= \mathbf{Pr}\,(b' = 1 \mid b = 1) + \mathbf{Pr}\,(b' = 0 \mid b = 0) - 1 \\
&= \frac{\mathbf{Pr}\,(b' = 1 \wedge b = 1)}{\mathbf{Pr}\,(b = 1)} + \frac{\mathbf{Pr}\,(b' = 0 \wedge b = 0)}{\mathbf{Pr}\,(b = 0)} - 1 \\
&= 2\big(\mathbf{Pr}\,(b' = 1 \wedge b = 1) + \mathbf{Pr}\,(b' = 0 \wedge b = 0)\big) - 1 \\
&= 2\mathbf{Pr}\,(b' = b) - 1 \\
&= 2\mathbf{Pr}\,(\mathsf{success}) - 1\,.
\end{aligned}$$

We call $\Delta_{\mathcal{A}}(\mathrm{SC}_K\ ;\ \mathcal{RO})$ *the advantage of* $\mathcal{A}$ *in distinguishing* SC_K *from* $\mathcal{RO}$. In literature, it is also known as the advantage of $\mathcal{A}$ in breaking the pseudorandom function (PRF) security of SC_K, and it is denoted $\mathrm{Adv}^{\mathrm{prf}}_{\mathrm{SC}}(\mathcal{A})$.

Definition 4.4.1 (pseudorandom function security)**.** Let SC be a stream cipher with key size k. Let K be randomly drawn from the set of all keys. The advantage of an adversary $\mathcal{A}$ in distinguishing SC_K from a random oracle $\mathcal{RO}$ is defined as

$$\mathrm{Adv}^{\mathrm{prf}}_{\mathrm{SC}}(\mathcal{A}) = \Delta_{\mathcal{A}}(\mathrm{SC}_K\ ;\ \mathcal{RO})\,. \tag{4.2}$$

4.5 Understanding the distinguishing advantage

There is no way to prove an upper bound for $\mathrm{Adv}^{\mathrm{prf}}_{\mathrm{SC}}(\mathcal{A})$ for any concrete stream cipher, but it makes sense to claim such a bound. Breaking the cipher then simply corresponds with coming up with an algorithm $\mathcal{A}'$ with $\mathrm{Adv}^{\mathrm{prf}}_{\mathrm{SC}}(\mathcal{A}')$ higher than the one claimed. To understand what kind of claims we can make for a stream cipher SC, we first have to quantify the complexity of an adversary.

4.5.1 Adversarial complexity

There are two types of information that $\mathcal{A}$ might learn about its oracle.

First off, $\mathcal{A}$ can query its oracle, SC_K or $\mathcal{RO}$. These queries are called *online queries* (as in the real world these queries correspond to conversations with the keyed instance of SC_K), and they are stored in query history Q_d. Clearly, the advantage of $\mathcal{A}$ in distinguishing SC_K from $\mathcal{RO}$ increases with the number of queries that $\mathcal{A}$ makes. We could try to express the claimed advantage as a function of the number of queries. However, in many cases this is not a representative metric for the attack complexity. What is much more representative, is the total number of bits sent to and received from the oracle. In this case, it is the number of diversifier bits sent and keystream bits received in all queries. We define by $|Q_d|$ the number of diversifier and keystream bits in Q_d. It is called the *online* or *data* complexity.

The second source of information of $\mathcal{A}$ comes from evaluations of SC for arbitrarily chosen keys. For this, $\mathcal{A}$ need not be connected with its oracle: as the specification of SC is known, $\mathcal{A}$ can implement it and run it offline. For this reason, these evaluations are called *offline evaluations.* They are stored in a query history Q_c. As above, what matters for Q_c is the number of keystream bits, and we define by $|Q_c|$ the number of keystream bits in Q_c. It is called the *offline* or *computational* complexity.

In practice, measuring $|Q_c|$ is subtle. The reason for this is that it, intuitively, measures the maximum number of evaluations of SC that adversary $\mathcal{A}$ can compute offline in a certain amount of time. A typical normalization that is often considered, is that generating one keystream bit takes one unit of time, but it may be that the adversary can use its time more efficiently so as to compute more than $|Q_c|$ keystream bits in time $|Q_c|$.

On the other side, $|Q_c|$ not only covers function evaluations, but also other computations that are relevant to the scheme but cannot be expressed as a function of the number of evaluations of SC. For example, if the distinguisher has gathered a certain set of oracle-responses and a certain set of offline primitive evaluations, it may have to do linear algebra computations to make up its decision based on the gathered data.

These two inaccuracies in the time parameter are generally neglected. One typically assumes that time is scaled and normalized and that $|Q_c|$ time allows for the generation of $|Q_c|$ keystream bits of SC.

4.5.2 Claiming security

Having discussed adversarial complexities, we can now discuss what claims one may make for stream ciphers. Consider a given stream cipher SC with k-bit key. A typical claim that the designers may have posed for this stream cipher is the following:

Claim 4.5.1. Let SC be a stream cipher with key size k. There exists no adversary $\mathcal{A}$ with computational complexity $|Q_c|$ less than 2^{k-1} that succeeds in distinguishing SC_K from a random oracle with advantage more than $1/2$.

This claim captures the idea that there should be no attacks more efficient than exhaustive key search. However, it is rather vague about the advantage of adversaries with limited resources. The following claim is much more clear:

Claim 4.5.2. Let SC be a stream cipher with key size k. For any adversary $\mathcal{A}$ with computational complexity $|Q_c|$,

$$\mathrm{Adv}_{\mathrm{SC}}^{\mathrm{prf}}(\mathcal{A}) \leq \frac{|Q_c|}{2^k}\,.$$

4.5.3 Breaking claims

Claims like the above cannot be proven. However, it is possible to disprove them. This happens by means of attacks.

Again consider the stream cipher SC with k-bit key that serves are running example, and assume that the designers posed the claim of Assumption 4.5.2. Suppose a team of researchers find an attack on SC: they show that SC_K can be distinguished from a random oracle $\mathcal{RO}$ with probability 1 using computational complexity $2^{k/2}$. Formally, this means that this team of researchers have described an adversary $\mathcal{A}'$ with computational complexity $|Q_c| = 2^{k/2}$ such that

$$\mathrm{Adv}_{\mathrm{SC}}^{\mathrm{prf}}(\mathcal{A}') = 1\,.$$

This is better than the claimed bound of Assumption 4.5.1, and the adversary is thus considered to be a valid attack on the scheme.

4.6 Adaptation to block ciphers

Above description of distinguishing advantages is for functions that should behave like a random oracle. The situation for a block cipher is different. Recall from Section 3.2 that a block cipher takes as input a k-bit key K and a b-bit plaintext P and *bijectively* transforms it to a b-bit ciphertext C. For a fixed key, it is a permutation on b-bit strings. This makes the situation different from before, where the function was compared with a random oracle that has collisions and variable-length outputs.

We will use an adaptation of the pseudorandom function security of Definition 4.4.1 to block ciphers. In the new thought experiment, we compare a block cipher B_K with a random permutation $\mathcal{P}$. This function $\mathcal{P}$ is defined similarly as a random oracle, with the difference that it is for b-bit length inputs and outputs only, and that it never responds with the same value to two different values. Alternatively, one may consider $\mathcal{P}$ to be uniformly randomly drawn from the set of all b-bit permutations.

4.6.1 Distinguishing advantage

The distinguishing advantage remains mainly unchanged: adversary $\mathcal{A}$ is either in the "real world", talking to B_K, or in the "ideal world", talking to $\mathcal{P}$. It can make queries to its oracle, and in the end, it has to guess whether it is in the real world or the ideal world. The *distance* between the real world and the ideal world with respect to adversary $\mathcal{A}$ is now defined as

$$\Delta_{\mathcal{A}}(\mathrm{B}_K\ ;\ \mathcal{P}) = \mathbf{Pr}\left(\mathcal{A}^{\mathrm{B}_K} = 1\right) - \mathbf{Pr}\left(\mathcal{A}^{\mathcal{P}} = 1\right)\,. \tag{4.3}$$

We call $\Delta_{\mathcal{A}}(\mathrm{B}_K\ ;\ \mathcal{P})$ *the advantage of* $\mathcal{A}$ *in distinguishing* B_K *from* $\mathcal{P}$. In literature, it is also known as the advantage of $\mathcal{A}$ in breaking the pseudorandom permutation (PRP) security of B_K, and it is denoted $\mathrm{Adv}^{\mathrm{prp}}_{\mathrm{B}}(\mathcal{A})$.

Definition 4.6.1 (pseudorandom permutation security)**.** Let SC be a stream cipher with key size k. Let K be randomly drawn from the set of all keys. The advantage of an adversary $\mathcal{A}$ in distinguishing B_K from a random permutation $\mathcal{P}$ is defined as

$$\mathrm{Adv}^{\mathrm{prp}}_{\mathrm{B}}(\mathcal{A}) = \Delta_{\mathcal{A}}(\mathrm{B}_K\ ;\ \mathcal{P})\,. \tag{4.4}$$

For block ciphers, one often measures the complexity of an adversary in the number of input blocks to the oracle, rather than in the number of bits. As B_K and $\mathcal{P}$ are of fixed length b bits only, this is just a convention: there is a one-to-one correspondence.

4.6.2 Security of AES

Given the relevance of AES, it is used millions of times per day, worldwide, one might hope for a nice upper bound on

$$\mathrm{Adv}^{\mathrm{prp}}_{\mathrm{AES}}(\mathcal{A})$$

that holds for any possible adversary $\mathcal{A}$. Unfortunately, for block ciphers the situation is not much different than for stream ciphers. We cannot prove an upper bound on $\mathrm{Adv}^{\mathrm{prp}}_{\mathrm{AES}}(\mathcal{A})$. As discussed in Section 2.6 the best we can do, is to just claim an upper bound on this quantity.

For Rijndael (AES), a claim was made that corresponds to the following:

Claim 4.6.2. Consider AES with key size k. For any adversary $\mathcal{A}$ with computational complexity $|Q_c|$,

$$\mathrm{Adv}^{\mathrm{prp}}_{\mathrm{AES}}(\mathcal{A}) \leq \frac{|Q_c|}{2^k}\,.$$

Note that there exists an adversary whose advantage matches this bound, namely one that performs an exhaustive key search with $|Q_c|$ attempts.

The claim on AES turns out to be very useful for claiming security of cryptographic schemes that are built on top of AES. Many cryptographic encryption and authentication schemes used in practice can be considered to be built as modes of use *on top of* AES. Under the assumption that AES is secure (Claim 4.6.2), one can abstract this cryptographic primitive, and focus on the mode of use itself. This mode of use can then be formally proven secure using (mostly) probability theory and combinatorics. This approach will be demonstrated in the following two chapters.

Chapter 5

Block cipher modes of use for encryption

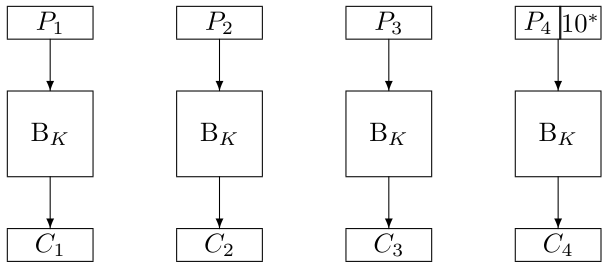
In this chapter, we consider ways to encipher messages with arbitrary length using a block cipher. These ways are called *modes of use*, or just *modes* for short. We subdivide these modes into two types depending on how the plaintext is processed. We treat the first type, called *block encryption*, in Section 5.1, and the second type, called *stream encryption*, in Section 5.2. We give a comparison of these modes in Section 5.3. In Section 5.4, we argue about security of modes.

5.1 Block encryption

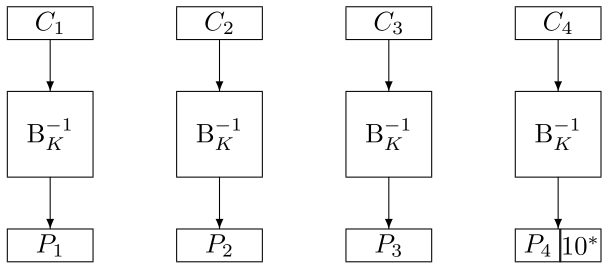
We speak of *block encryption* if the ciphertext is the result of applying a block cipher to the plaintext. The simplest block encryption mode is Electronic Codebook mode (ECB). We first explain a degenerated version of ECB that can only encrypt messages P that have as length a multiple of the block length b. This degenerated version already reveals the most important advantages and limitations of ECB.

Suppose the length of P is $\ell \cdot b$ bits. The first step is to split the plaintext P into ℓ blocks of b bits. We denote these blocks by $P_1, P_2, P_3, \ldots, P_\ell$. ECB encryption just consists of applying the block cipher to these blocks separately, leading to a sequence of ciphertext blocks $C_i = \mathrm{B}_K(P_i)$. The ciphertext C is simply the concatenation of the ciphertext blocks: $C = C_1 \| C_2 \| \cdots \| C_\ell$. ECB decryption is the inverse operation: split C into blocks C_i, apply the inverse block cipher to obtain the plaintext blocks $P_i = \mathrm{B}_K^{-1}(C_i)$ and concatenate the plaintext blocks to obtain $P = P_1 \| P_2 \| \cdots \| P_\ell$.

ECB has the advantage that both encryption and decryption of the blocks are parallelizable as they are encrypted and decrypted separately and independent of their position in the plaintext or ciphertext respectively. The consequence of this independence is that equal blocks $P_i = P_j$ in the plaintext will give rise to equal blocks $C_i = C_j$ in the ciphertext. This implies that the security offered by ECB is quite limited. It is reasonable to see *wide block encryption* using a random permutation, discussed in Section 3.1, as the ideal form of block encryption. Clearly, a block cipher in ECB mode can be distinguished from wide block encryption with a random permutation in a single two-block query as ECB will encrypt $P_1 \| P_1$ to $C_1 \| C_1$ while in wide block encryption the symmetry in P will vanish with overwhelming probability. So ECB cannot be called a secure mode by any modern definition of *secure*.



(a) Encryption.



(b) Decryption.

Figure 5.1: Electronic Codebook mode.

5.1.1 Padding

As said, we expect encryption modes to support arbitrary-length plaintexts, i.e., with length that is not necessarily a multiple of b. We can resolve this using *padding*: appending bits to the plaintext so that the result can be split into b-bit blocks.

A padding rule pad_b takes an arbitrary-length message and turns it into a string with length that is a multiple of b. ECB can now be extended to support arbitrary-length plaintexts P by including a padding step $P_1\|\ldots\|P_\ell \leftarrow \mathrm{pad}_b(P)$ prior to encryption. Decryption of C gives the padded message $P_1\|\ldots\|P_\ell$. For obtaining P, it is necessary to "undo" the padding.

To see why the ability to undo the padding is important, let us first consider the simplest possible padding rule:

$$\mathrm{pad}_b \colon P \mapsto P\|0^{-|P| \bmod b}\,.$$

This rule simply pads P with the minimum number of zeros to get a string with length a multiple of b bits. The operation is clearly non-injective: if we take any plaintext P of length $b-1$ bits, then

$$\mathrm{pad}_b(P) = \mathrm{pad}_b(P\|0)\,.$$

This means that $\mathrm{ECB}_K(P) = \mathrm{ECB}_K(\mathrm{pad}_b(P\|0))$. Therefore, upon decryption, one cannot determine the plaintext that was actually encrypted!

A simple *injective* padding is one that adds a single one and then the minimum number of zeros:

$$\mathrm{pad}_b \colon P \mapsto P\|1\|0^{-|P|-1 \bmod b}\,.$$

It is easy to verify that this padding operation is injective. The consequence is that the plaintext can be recovered unambiguously from an ECB-deciphered ciphertext. Namely, it suffices to remove all trailing zeros and a single 1. The resulting scheme is given in Figure 5.1.

Myriad padding rules exist for satisfying different requirements. For example, the message length can be somewhat obfuscated by applying random-length padding prior to encryption. Another requirement is that the padding bits must be unpredictable, or rather *hard to guess by an adversary.* This is useful in public-key encryption of low-entropy plaintexts such as PINs or passwords. In this case the padding must include random bits. What all these padding rules have in common is that it must be possible to undo the padding. Padding may seem like a simple operation, badly defined or implemented padding have been the main cause of many protocols being broken.

5.1.2 Cipher Block Chaining

As said, in ECB equal plaintext blocks result in equal ciphertext blocks: if $P_1 = P_2$ in Figure 5.1, then also $C_1 = C_2$.

One can salvage this issue by randomizing the input blocks to B_K in some way. A well-established mode that does this is Cipher Block Chaining (CBC). As in ECB, the plaintext P is injectively padded to a string with length a multiple of b bits and then split into ℓ blocks $P_1, P_2 \ldots, P_\ell$. Prior to encryption with B_K, each block P_i is *randomized* by bitwise adding to it the previous block of the ciphertext C_{i-1}. In other words, we have for all $i > 1$: $C_i = \mathrm{B}_K(P_i + C_{i-1})$. For encrypting P_1 there is no *previous block of ciphertext* and there the CBC mode specifications prescribe to use a (random) initial value IV: so $C_1 = \mathrm{B}_K(P_1 + IV)$. Encryption is depicted in Figure 5.2.

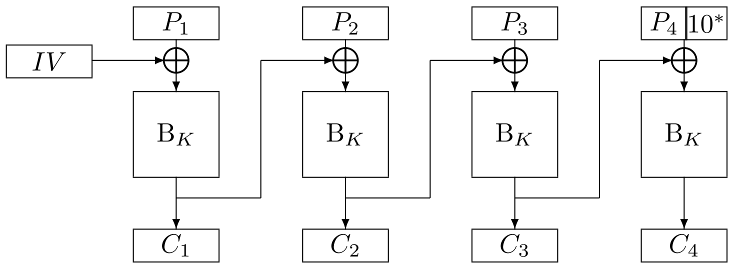


Figure 5.2: Cipher Block Chaining mode encryption.

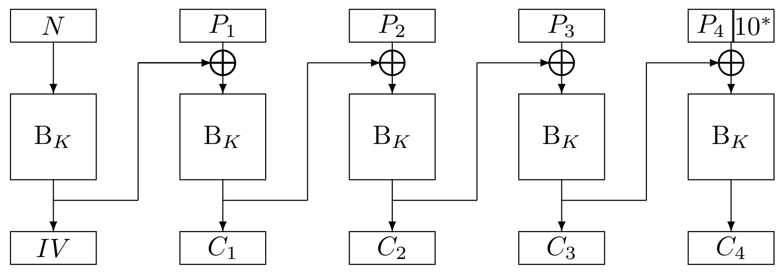
For a secure block cipher, ciphertext blocks C_i corresponding to different inputs will appear fully random, and hence so will the inputs to B_K and subsequently the outputs. For the input to the first application of B_K, namely $P_1 + IV$ to be random, it is hence necessary for IV to be random.

Even when using a secure block cipher and random IVs, bad things that can still happen. Namely, colliding inputs to B_K will still lead to equal ciphertext blocks. Such a collision leaks information about the difference between plaintext blocks: for example, $C_i = C_j$ implies $P_i + C_{i-1} = P_j + C_{j-1}$ and hence $P_i + P_j = C_{i-1} + C_{j-1}$. As C_{i-1} and C_{j-1} are known to the adversary, she learns the value of $P_i + P_j$, similar to the case of re-use of a one-time pad. The collision probability depends on the total number of blocks encrypted and the block length b and follows the birthday bound $\Pr(\text{collision}) \approx N^2 2^{-(b+1)}$ with N the total number of blocks encrypted.

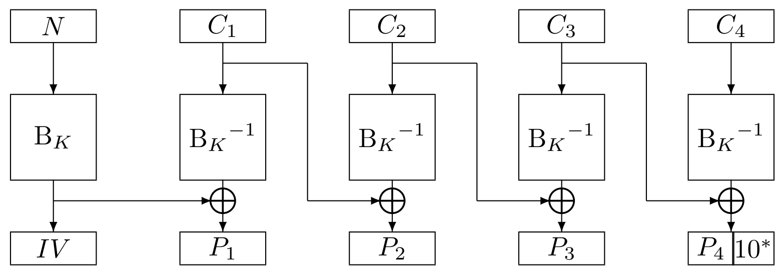
Upon decryption, P_i is obtained by applying the inverse block cipher to C_i and then *subtracting* C_{i-1}: $P_i = \mathrm{B}_K^{-1}(C_i) + C_{i-1}$. For decrypting the first block, we obviously have $P_1 = \mathrm{B}_K^{-1}(C_1) + IV$.

Generating IV randomly is a burden as it requires the presence of a decent random generator. Moreover, Alice must send the IV along with the cryptogram C to Bob as he needs it to do the decryption. It is possible to turn the requirement of IV randomness into the requirement of a unique diversifier D by computing IV as $IV = \mathrm{B}_K(D)$. The scheme is depicted in Figure 5.3.

CBC encryption has the disadvantage that it is inherently sequential: the encryption of plaintext block P_i can not be started before C_{i-1} is available. This can be applied recursively and hence imposes that encryption of blocks is done serially start with P_1 and ending with P_ℓ. Remarkably,



(a) Encryption.



(b) Decryption.

Figure 5.3: Cipher Block Chaining mode with random IV.

decryption can be done in parallel as a plaintext block P_i only depends on ciphertext blocks C_{i-1} and C_i. As ECB, CBC suffers from message expansion: due to padding, the ciphertext is longer than the plaintext.

With respect to security, CBC improves over ECB as equal plaintext blocks $P_i = P_j$ encrypt to different and seemingly unrelated ciphertext blocks C_i and C_j. However, CBC disappoints when we try to define its security according to modern notions. Thanks to the presence of the IV, it is reasonable to see *one-time pad encryption* using the output of a random oracle, discussed in Section 1, as the ideal CBC encryption can strive for. For achieving some level of security, CBC requires a random IV compared to a stream cipher that only requires a unique diversifier D. Note that a random IV must be generated for each message and be sent along with the ciphertext. When considering the extended CBC mode that computes the IV from a unique diversifier D, it reaches security under the same conditions as a stream cipher.

This is rather disappointing as the whole point of introducing (wide) block encryption in Section 3.1 was to get rid of the nonce requirement on D inherent in stream encryption. As a last straw, block encryption advocates desperately argue that CBC has the advantage over stream encryption that leakage due to nonce misuse (diversifier repetition) is much smaller than for stream encryption. The underlying idea – that it would be acceptable to tolerate nonce misuse in protocol implementations – makes this a particularly pathetic argument.

5.2 Stream encryption

As we saw in the previous section, block encryption modes have failed to deliver on their promise. Still, we are stuck with a lot of block cipher implementations, especially AES, and we may make the best of it.

For encryption this is the following: we can use block ciphers to build secure stream ciphers. These are *stream cipher modes of use* and in these lecture notes we will discuss two of them. The first is the output feedback mode (OFB) that is mostly of historical importance and we can use it to show what not to do. The second is counter mode and is the only decent block cipher mode of encryption left.

Stream encryption has a number of advantages over block encryption and this immediately follows from their architecture. Recall that a stream cipher takes a diversifier D and generates a keystream Z as in Figure 5.4. This keystream Z can then be used for encryption of a plaintext P to a ciphertext by adding the keystream: $C = P + Z$. Decryption consists of subtracting the keystream: $P = C + Z$. Sender and receiver generate the keystream Z in the same way and so no *inverse function* is needed for decryption as is the case for block ciphers. Moreover, thanks to the fact that encryption is done by adding a keystream, there is no need for plaintext padding and hence there is no message expansion.

5.2.1 Output Feedback Mode

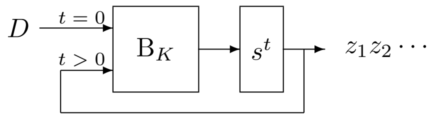


Figure 5.4: Output Feedback mode.

Output feedback mode (OFB) builds a stream cipher that has as updating function B_K and as output function the identity. Its state consists of two parts: the block cipher key K that is fixed and the *rolling state* s^t that has length the block length b of the block cipher B. We have:

$$z_t = s^t \text{ and } s^{t+1} = \mathrm{B}_K(s^t)\,. \tag{5.1}$$

The initialization consists in writing the key K and setting the initial state to an IV that is (at least partially) defined by the user: $s^0 = IV$.

As we saw, a stream cipher takes as input a diversifier D and the OFB mode takes an IV. At first sight, it seem logical to take $IV = D$. However, this gives a stream cipher that can be distinguished in two queries from a random oracle:

- First query $\mathrm{SC}_K(X, \ell = 2b)$ resulting in $Z = z_1 z_2$;
- Then query $\mathrm{SC}_K(z_1, \ell = b)$.

For the OFB mode, the second query will output z_2 (make sure you see that), while for a random oracle this is very unlikely.

A solution to this is to restrict the range of diversifiers to a subset of the space of IV values. For example, one can restrict diversifiers to $r < b$ bits and build the initial value as $IV = D \| 0^{b-r}$. If $b - r$ is sufficiently large, this prevents the problem discussed above as it is unlikely that z_1 will have its last $b - r$ bits equal to zero. Clearly, increasing r decreases the level of security one can achieve.

The space of possible running state values of the OFB mode has size 2^b and hence when generating streams longer than 2^b blocks, it is bound to cycle and generate a periodic keystream Z. As a secure block cipher behaves like a random permutation, it is hard to predict exactly how long these cycles are. However, statistically it is very unlikely that the initial state is in a short cycle. Namely, a priori the probability that s_0 is in a cycle shorter than ℓ is at most $\ell/2^b$.

Cycles would be a way to distinguish OFB from a random oracle but would require huge data complexity. The security of OFB mode is however limited by another effect. When generating a keystream that is not periodic, all state values are different s^t implying that all keystream blocks z_t are different. This is not necessarily the case for a random oracle, that will exhibit collisions between b-bit blocks in its output stream. This limits the maximum security strength of OFB mode to $b/2$ (birthday bound, see later).

Note that OFB is inherently serial, both for encryption and decryption.

5.2.2 Counter mode

A more suitable block cipher mode is Counter mode (CTR). In counter mode, the block cipher takes the place of the output function and the state-updating function is a simple counter. The state consists of two parts: the block cipher key K that is fixed and the *rolling state* s^t that has length the block length b of the block cipher B. We have:

$$z_t = \mathrm{B}_K(s^t) \text{ and } s^{t+1} = \mathrm{ItoS}_b(\mathrm{StoI}(s^t) + 1)\,.$$

The initialization consists in writing the key K and setting the initial state to an IV that is (at least partially) defined by the user: $s^0 = IV$.

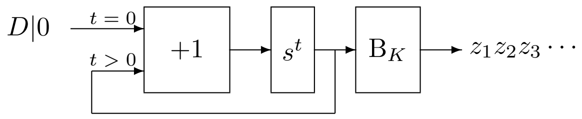


Figure 5.5: Counter mode.

In counter mode we face a similar problem as in OFB when mapping the diversifier D to the IV. Also here, a straightforward mapping $IV = D$ would allow distinguishing a block cipher in counter mode from a random oracle:

- First query $\mathrm{SC}_K(X, \ell = 2b)$ resulting in $Z = z_1 z_2$;
- Then query $\mathrm{SC}_K(\mathrm{ItoS}_b(\mathrm{StoI}(X) + 1), \ell = b)$.

For the CTR mode, the second query will output z_2 (make sure you see that), while for a random oracle this is very unlikely.

Also here a solution is to restrict the range of diversifiers to a subset of the space of IV values and restrict the length of keystreams that can be generated for a given diversifier. In this way, one can avoid deterministically avoid collisions in the state. For example, one can restrict diversifiers to $r < b$ bits and build the initial value as $IV = D \| 0^{b-r}$. If the keystream length is restricted to at most 2^{b-r}, this prevents the problem discussed above.

Unlike OFB, counter mode has a very predictable cycle structure: all states are in a single cycle of length 2^b. If the described scheme for IV computation and maximum length of keystream is respected, cycles simply pose no problem.

The security of counter mode is limited by a similar effect than OFB mode. When generating a keystream, all state values are different s^t's, implying that all keystream blocks z_t are different. This is not necessarily the case for a random oracle, that will exhibit collisions between b-bit blocks in its output stream. This limits the maximum security strength of OFB mode to $b/2$ (birthday bound, see later).

Note that counter mode is fully parallel, both for encryption and decryption.

Table 5.1: Comparison of ECB, CBC, OFB, and CTR.

	ECB	CBC	OFB	CTR
parallel encryption	✓	—	—	✓
parallel decryption	✓	✓	—	✓
inverse free	—	—	✓	✓
absence of message expansion	—	—	✓	✓
tolerant to bit flips in $C \rightarrow P$	—	—	✓	✓
graceful degradation if nonce violation	n/a	✓	—	—

5.3 Comparison

Table 5.1 provides a comparison of ECB, CBC, OFB, and CTR mode. It reports on the following aspects:

- Parallel encryption: specifies whether plaintext blocks can be encrypted in parallel or not;
- Parallel decryption: specifies whether ciphertext blocks can be decrypted in parallel or not. If so, this implies that one can separately decipher blocks of the ciphertext, wherever their position. This is usually called *random access to the ciphertext*;
- Inverse freeness: specifies whether the inverse block cipher B_K^{-1} is required;
- Absence of message expansion: whether the message must be padded or not before encryption;
- Tolerant to bit flips in ciphertext: whether bit flips in the ciphertext (e.g., due to a noisy communication channel) stay confined to the same bits in the plaintext after decryption;
- Graceful degradation in case of nonce violation: if a user uses twice the same value for the diversifier, does this lead to serious leakage about the plaintext?

5.4 Provable security of modes: the case of counter mode

We will demonstrate how one can formally argue that counter mode based on AES, called CTR[AES], is a secure stream cipher. Stated differently, we will derive an expression for an upper bound on the distinguishing advantage $\mathrm{Adv}^{\mathrm{prf}}_{\mathrm{CTR[AES]}}(\mathcal{A})$ of Definition 4.4.1.

As explained in Chapter 4, we cannot prove security of any concrete encryption scheme such as CTR[AES]. Instead, we can express an upper bound on the distinguishing advantage of CTR[AES] by a sum of two terms:

- A first term that bounds the PRP security of AES. This term cannot be proven but can be given in a claim, in this case in Claim 4.6.2;
- A second term that bounds the security of counter mode where AES with a secret key is replaced by a random 128-bit permutation. This term can be proven.

For someone who is willing to believe that Claim 4.6.2 is valid, this reasoning provides a lower bound the security of CTR[AES].

Theorem 5.4.1 (Security of AES in counter mode)**.** *Consider* CTR *mode with* AES *with key size* k*. For any adversary* $\mathcal{A}$ *with data complexity* $|Q_d|$ *blocks and computational complexity* $|Q_c|$*,*

$$\mathrm{Adv}^{\mathrm{prf}}_{\mathrm{CTR[AES]}}(\mathcal{A}) \leq \frac{\binom{|Q_d|}{2}}{2^{128}} + \mathrm{Adv}^{\mathrm{prp}}_{\mathrm{AES}}(\mathcal{A}'),$$

where $\mathcal{A}'$ *is an adversary with the same data complexity and almost the same computational complexity as* $\mathcal{A}$*.*

Claim 4.6.2 states that $\mathrm{Adv}^{\mathrm{prp}}_{\mathrm{AES}}(\mathcal{A}') \leq \frac{|Q_c|}{2^k}$. For someone believing Claim 4.6.2, Theorem 5.4.1 implies CTR[AES] is a secure stream cipher as long as:

- The data complexity of $\mathcal{A}$ satisfies $|Q_d| \ll 2^{64}$ blocks;
- The computational complexity of $\mathcal{A}$ satisfies $|Q_c| \ll 2^k$.

The limitation in data complexity is a manifestation of the *birthday paradox* of Section 15.5. The proof is given at very high detail in the remainder of this section.

Proof (of Theorem 5.4.1). Consider any adversary $\mathcal{A}$ with data complexity $|Q_d|$ blocks and computational complexity $|Q_c|$. Let K be randomly drawn from the set of all keys, and let $\mathcal{RO}$ be a random oracle. From Definition 4.4.1, we can conclude that our goal is to bound

$$\mathrm{Adv}^{\mathrm{prf}}_{\mathrm{CTR[AES]}}(\mathcal{A}) = \Delta_{\mathcal{A}}(\mathrm{CTR[AES]}_K\ ;\ \mathcal{RO}). \tag{5.2}$$

Step 1: isolating AES_K**.** Note that $\mathcal{A}$ has *indirect* access to AES_K: upon querying $\mathrm{CTR[AES]}_K$ with D it gets a sequence of values

$$\mathrm{AES}_K(\mathrm{ItoS}_b(\mathrm{StoI}(D)+1)),\ \mathrm{AES}_K(\mathrm{ItoS}_b(\mathrm{StoI}(D)+2)),\ \ldots.$$

As a first step, we will move to an adversary $\mathcal{A}'$ that has more power: one that can fully choose the input to AES_K. More detailed, $\mathcal{A}'$ is an adversary that has access to either AES_K or $\mathcal{RO}$ and tries to distinguish between them:

$$\Delta_{\mathcal{A}'}(\mathrm{AES}_K\ ;\ \mathcal{RO}). \tag{5.3}$$

The adversary $\mathcal{A}'$ will use its own oracle to *simulate* the oracle of $\mathcal{A}$ as follows. If $\mathcal{A}$ makes a query for input D and requested length ℓ bits, $\mathcal{A}'$ queries its own oracle for ℓ' blocks with ℓ' the smallest value that satisfies $\ell' \cdot b \geq \ell$:

$$\mathrm{ItoS}_b(\mathrm{StoI}(D)+1),\ \mathrm{ItoS}_b(\mathrm{StoI}(D)+2),\ \ldots,\ \mathrm{ItoS}_b(\mathrm{StoI}(D)+\ell').$$

It concatenates the outputs to Z and returns the first ℓ bits. In the end, $\mathcal{A}$ outputs a bit expressing its decision (see Section 4.4). Adversary $\mathcal{A}'$ outputs the same decision bit. The simulation is depicted in Figure 5.6.

The step from $\mathcal{A}$ to $\mathcal{A}'$ is called a *reduction*. As $\mathcal{A}'$ uses its own oracle (namely AES_K or $\mathcal{RO}$) to simulate the oracle of $\mathcal{A}$ (namely $\mathrm{CTR[AES]}_K$ or $\mathcal{RO}$), we can conclude that $\mathcal{A}'$ correctly distinguishes its oracles if $\mathcal{A}$ manages to do so. Stated differently, the distinguishing advantage of $\mathcal{A}'$ cannot be smaller than the distinguishing advantage of $\mathcal{A}$, and the quantities of (5.2) and (5.3) satisfy

$$\mathrm{Adv}^{\mathrm{prf}}_{\mathrm{CTR[AES]}}(\mathcal{A}) = \Delta_{\mathcal{A}}(\mathrm{CTR[AES]}_K\ ;\ \mathcal{RO}) \leq \Delta_{\mathcal{A}'}(\mathrm{AES}_K\ ;\ \mathcal{RO}). \tag{5.4}$$

Note that $\mathcal{A}'$ has the same data complexity as $\mathcal{A}$ and a slightly larger computational complexity (because it needs to put some negligible effort in performing the simulation). However, this change in the computational complexity is negligible and typically ignored (see also the discussion of adversarial complexities in Section 4.5.1).

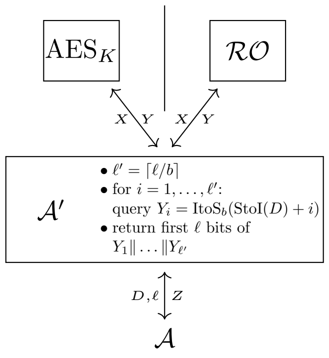

AES_K
$\mathcal{RO}$
X
Y
X
Y
$\mathcal{A}'$
• $\ell' = \lceil \ell/b \rceil$
• for $i = 1, \ldots, \ell'$:
query $Y_i = \mathrm{ItoS}_b(\mathrm{StoI}(D) + i)$
• return first ℓ bits of
$Y_1 \| \ldots \| Y_{\ell'}$
D, ℓ
Z
$\mathcal{A}$

Figure 5.6: The simulation step in the proof of Theorem 5.4.1.

Step 2: a hybrid argument. Our goal is now to upper bound the advantage of $\mathcal{A}'$ in distinguishing AES_K from $\mathcal{RO}$ (of (5.3)), where $\mathcal{A}'$ has data complexity $|Q_d|$ blocks and computational complexity $|Q_c|$. Unfortunately, (5.3) is not quite the same as the pseudorandom permutation security of AES of Definition 4.6.1, and we cannot readily apply Claim 4.6.2.

Let $\mathcal{P}$ be a random permutation with width 128. We have

$$\begin{aligned}\Delta_{\mathcal{A}'}(\mathrm{AES}_K\ ;\ \mathcal{RO}) &= \mathbf{Pr}\left(\mathcal{A}'^{\mathrm{AES}_K} = 1\right) - \mathbf{Pr}\left(\mathcal{A}'^{\mathcal{RO}} = 1\right)\\ &= \left(\mathbf{Pr}\left(\mathcal{A}'^{\mathrm{AES}_K} = 1\right) - \mathbf{Pr}\left(\mathcal{A}'^{\mathcal{P}} = 1\right)\right) + \left(\mathbf{Pr}\left(\mathcal{A}'^{\mathcal{P}} = 1\right) - \mathbf{Pr}\left(\mathcal{A}'^{\mathcal{RO}} = 1\right)\right)\\ &= \Delta_{\mathcal{A}'}(\mathrm{AES}_K\ ;\ \mathcal{P}) + \Delta_{\mathcal{A}'}(\mathcal{P}\ ;\ \mathcal{RO})\,. \end{aligned} \tag{5.5}$$

Step 3: permutation-to-function switch. We now have to upper bound the remaining terms in (5.5). The first term is exactly $\mathrm{Adv}^{\mathrm{prp}}_{\mathrm{AES}}(\mathcal{A}')$ and the subject of Claim 4.6.2. For the second term, note that $\mathcal{A}'$ has only access to either random permutation $\mathcal{P}$ or random oracle $\mathcal{RO}$. It does not have access to any concrete keyed primitive like AES_K. The only way $\mathcal{A}'$ can learn about $\mathcal{P}$ or $\mathcal{RO}$ is by querying them. They have no algorithmic descriptions, so the adversary cannot do computations as it can do, e.g., for AES. So even though $\mathcal{A}'$ has a certain computational complexity around $|Q_c|$, she cannot use this to increase her changes in distinguishing $\mathcal{P}$ from $\mathcal{RO}$. We can therefore ignore the computational complexity. For ease, we therefore move to an adversary $\mathcal{A}''$ that has data complexity $|Q_d|$ as before, but that has unbounded computational complexity. Clearly, the distinguishing advantage of $\mathcal{A}''$ cannot be smaller than the distinguishing advantage of $\mathcal{A}'$, and we have

$$\Delta_{\mathcal{A}'}(\mathcal{P}\ ;\ \mathcal{RO}) \leq \Delta_{\mathcal{A}''}(\mathcal{P}\ ;\ \mathcal{RO})\,. \tag{5.6}$$

The distance can now be measured by simple probability theory. Both $\mathcal{P}$ and $\mathcal{RO}$ return uniformly random 128-bit responses for each query, but those of $\mathcal{P}$ never collide whereas those of $\mathcal{RO}$ may do. Adversary $\mathcal{A}''$ can thus only distinguish the two oracles if there are colliding outputs from $\mathcal{RO}$. As the data complexity of $\mathcal{A}''$ is $|Q_d|$ blocks, the probability that two of the $|Q_d|$ blocks from $\mathcal{RO}$ collide is at most $\frac{\binom{|Q_d|}{2}}{2^{128}}$. We therefore obtain

$$\Delta_{\mathcal{A}''}(\mathcal{P}\ ;\ \mathcal{RO}) \leq \frac{\binom{|Q_d|}{2}}{2^{128}}\,. \tag{5.7}$$

Step 4. We conclude by combining the individual steps. From (5.4), (5.5), (5.6), and (5.7), we obtain

$$\mathrm{Adv}^{\mathrm{prf}}_{\mathrm{CTR[AES]}}(\mathcal{A}) \overset{(5.4)}{\leq} \Delta_{\mathcal{A}'}(\mathrm{AES}_K\ ;\ \mathcal{RO})$$

$$\begin{aligned}
&\overset{(5.5)}{=} \Delta_{\mathcal{A}'}(\mathrm{AES}_K\ ;\ \mathcal{P}) + \Delta_{\mathcal{A}'}(\mathcal{P}\ ;\ \mathcal{RO}) \\
&\overset{(5.6)}{\leq} \Delta_{\mathcal{A}'}(\mathrm{AES}_K\ ;\ \mathcal{P}) + \Delta_{\mathcal{A}''}(\mathcal{P}\ ;\ \mathcal{RO}) \\
&\overset{(5.7)}{\leq} \mathrm{Adv}^{\mathrm{prp}}_{\mathrm{AES}}(\mathcal{A}') + \frac{\binom{|Q_d|}{2}}{2^n}\,.
\end{aligned}$$

This completes the proof. □

Chapter 6

Data authentication

So far we have focused on the protection of confidentiality of messages by encryption. Another important concern is *authentication*: the assurance of Bob that messages he thinks he is receiving from Alice were actually sent by Alice in the first place and not modified in transit.

6.1 What is authentication?

When Alice sends a message to Bob, it is not only important to her that no one else (in particular that nasty Eve) can read the message, but also that Bob gets the message she sent, and not something else. Similarly, when Bob receives a message from "Alice", he wants to be sure that the message is indeed from Alice, and even more. All this is gathered under the term *authentication*. Below are some examples of what can go wrong.

Example 6.1.1. Suppose that Alice and Eve are friends and that Bob is a bank. Alice sends the following message to Bob: "Hereby Alice authorizes Bob to transfer 10 euros from her account to the account of Eve."

Eve turns to be of questionable morale and somehow intercepts this message. Eve replaces the message with the message "Hereby Alice authorizes Bob to transfer 1000 euros from her account to the account of Eve." How can Bob know that this is not Alice's message? ♠

The preceding is not the only problem that we can think of in terms of authentication. There are many more:

Example 6.1.2. Suppose that Alice and Eve are still friends and that Bob is a bank. Alice sends the following message (*with* a signature) to Bob: "Transfer 100 euros from my account to the account of Eve."

Again, Eve has a questionable morale and she copies the message (*and* the signature) "Transfer 100 euros from my account to the account of Eve." and sends it to Bob again. Now Bob, who reads the signature at the bottom of the message, thinks this is Alices message and again transfers 100 euros to Eves account.

This kind of attack is called a *replay attack*. ♠

Example 6.1.3. In this example Alice has a badge to access a specific part of a building (e.g. a laboratory) that should be inaccessible to others. We let Bob take the role of the interface where the badge needs to be swiped. So the message gets sent from the badge to the interface and checked for authorization.

Eve, being as malicious as ever, somehow during the lunch break, goes into Alice's office and scans the badge. On the device that Eve uses for this action the authentication method is stored. By presenting the device to the interface (Bob), the authorization message from the badge gets sent from the device to the interface and the door opens, giving Eve unauthorized access to the laboratory.

Something is needed to make sure that it is indeed Alice holding the badge to open the door. For in the preceding scenario, there is no way for Alice to know that her authorization is used. ♠

A more real-life example perhaps, is with cars that are equipped with 'smart' keys.

Example 6.1.4. New cars come with 'smart' keys. That is, the car unlocks whenever you are close with the key, and the car can start while you're just sitting inside of the car, without having to put the key into the ignition. (a good thing, since most of those pompous cars don't even have an ignition anymore.)

While seemingly handy, this is in fact dangerous and tv-commercials are broadcasted where you are urged to store the key in some metal box or at least far away from the front door.

What could happen is that someone would 'hack' the key and put the signal on some device and afterwards access your car and be able to drive away with it. (Is this a different example than the previous one?)

Another problem is the distance that the signal should be allowed to carry. A signal that is too weak might not open the car door when you're next to it, while a too strong signal will open the car door so far away that a burglar might get in the car and steal some of your stuff. (We leave this example open for the reader to think of more difficulties with these 'smart' keys, while noting that most have been solved by the distributors already.) ♠

6.2 Message authentication code functions

Like discussed in the previous section, authentication of messages, transactions and entities can be solved using electronic signatures. In cryptography, these are sub-divided in two types: those that can be verified using a public key and those that require for verification the secret key that was used to generate it.

The term electronic signatures has been claimed by the public-key crypto lobby and as a consequence it is now quasi exclusively used for public-key signatures. Symmetric crypto signatures are now called *message authentication code* functions, MAC functions for short. The output of a MAC function, so the symmetric-key electronic signature itself, is called a *tag* or sometimes also a MAC.

So when Alice sends a message M to Bob and she wishes Bob to authenticate the message, she applies a MAC function to M resulting in $T = \mathrm{MAC}_K(M)$ where K is a symmetric key that she shares with Bob. Alice sends the couple (M, T) to Bob and Bob can authenticate the received message M by computing $T' \leftarrow \mathrm{MAC}_K(M)$ and verifying that the computed tag T' equals the received tag T.

A MAC function MAC is a cryptographic primitive that takes a secret key K and an arbitrary-length message M and returns an ℓ-bit tag T. See also Figure 6.1. The process of verifying a tag is formalized in the concept of a verification function VFY. On input of a secret key K, an arbitrary-length message M, and an ℓ-bit tag T, it verifies whether $\mathrm{MAC}(K, M) \stackrel{?}{=} T$, and outputs OK/NOK accordingly.

Protection against replay attack, as required in the use case of Example 6.1.2, can be achieved with a simple message counter and a MAC function. It is sufficient for Bob to keep track of the number of received messages in a counter N and include this counter in the MAC computation.

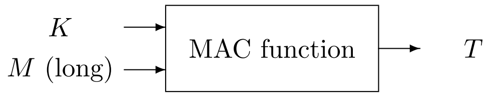


Figure 6.1: Schematic of a MAC function.

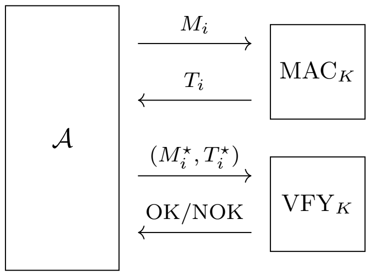


Figure 6.2: Unforgeability game. The adversary can make a certain amount of MAC requests M_i to receive tags T_i. It can then make a certain amount of forgery attempts $(M_i^\star, T_i^\star)$ in such a way that a forgery attempt never repeats an older MAC request. The adversary "wins" if the verification oracle every returns OK.

In particular, Alice computes the tag $T \leftarrow \text{MAC}_K(\text{ItoS}(N)\|M)$ and sends (N, M, T). Bob verifies whether this is actually the N-th message he receives from Alice and rejects the message if not. Otherwise he authenticates (N, M, T) by means of $\text{VFY}_K(\text{ItoS}(N)\|M, T)$ and accepts the message if this is successful. Note that this requires Bob to keep track of the number of received messages N and hence Bob's processing becomes *stateful*.

If freshness is required as in Example 6.1.3, it suffices for Bob to send an unpredictable challenge R to Alice and for Alice to include the challenge in the MAC function input, as $T \leftarrow \text{MAC}_K(R\|M)$. Here M may contain data that Bob may wish to authenticate.

Lastly, we cannot use MACs to solve issues like in Example 6.1.4, there is no solution in cryptography for problems like these. Instead, a more simple approach is the only option: Just press the button!

6.2.1 MAC security

The main security goal of a MAC function is that is should be infeasible to forge a message and its tag (M, T) for an adversary that does not know the key K. If an attacker Eve obtains a certain amount of message-tag pairs, and then succeed in computing a tag T for a *new* message M, we call this a *forgery*. The hardness of generating a forgery for a MAC function MAC is formalized by its *unforgeability*, widenoted as $\text{Adv}^{\text{unf}}_{\text{MAC}}(\mathcal{A})$.

Definition 6.2.1 (unforgeability)**.** Let MAC be a MAC function with key size k. Let K be randomly drawn from the set of all keys. The advantage of an adversary $\mathcal{A}$ in mounting a forgery against MAC_K is defined as

$$\text{Adv}^{\text{unf}}_{\text{MAC}}(\mathcal{A}) = \mathbf{Pr}\left(\mathcal{A}^{\text{MAC}_K, \text{VFY}_K} \text{ ever gets OK from } \text{VFY}_K\right) . \tag{6.1}$$

Here, $\mathcal{A}$ is not allowed to make a forgery attempt for a message-tag pair (M, T) that it ever received from MAC_K.

The security model is depicted in Figure 6.2.

One can prove that if MAC_K is a secure pseudorandom function (in light of Definition 4.4.1), it is unforgeable.

Theorem 6.2.2 (a good PRF is a good MAC)**.** *Consider a MAC function* MAC *with key size k and output size ℓ. For any adversary $\mathcal{A}$ with data complexity $|Q_d|$ blocks and computational complexity $|Q_c|$, that makes $|Q_f|$ forgery attempts,*

$$\mathrm{Adv}^{\mathrm{unf}}_{\mathrm{MAC}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathrm{prf}}_{\mathrm{MAC}}(\mathcal{A}') + \frac{|Q_f|}{2^\ell} ,$$

where $\mathcal{A}'$ is an adversary with at most the computational complexity as $\mathcal{A}$.

Proof. Consider any adversary $\mathcal{A}$ with data complexity $|Q_d|$ blocks and computational complexity $|Q_c|$, that makes $|Q_f|$ forgery attempts. Let K be randomly drawn from the set of all keys, and let $\mathcal{RO}$ be a random oracle.

Step 1: simulation. We will construct an adversary $\mathcal{A}'$ for the PRF security of MAC_K that uses $\mathcal{A}$. Note that $\mathcal{A}'$ has access to either MAC_K or $\mathcal{RO}$. The adversary $\mathcal{A}'$ will use its own oracle to *simulate* the oracle of $\mathcal{A}$ as follows. If $\mathcal{A}$ makes a MAC query $\mathrm{MAC}_K(M)$, adversary $\mathcal{A}'$ will simply query its own oracle ($\mathrm{MAC}_K(M)$ or $\mathcal{RO}$) on input M and return the resulting tag T. At the end, $\mathcal{A}$ outputs a forgery $(M^\star, T^\star)$. Adversary $\mathcal{A}'$ queries its oracle on input $M^\star$ and verifies whether the outcome equals $T^\star$. It outputs $b' = 1$ if this is the case, and outputs $b' = 0$ if this is not the case.

Step 2: success of the simulation. Recall that $\mathcal{A}'$ has access to either MAC_K or $\mathcal{RO}$, and its distinguishing advantage is defined as

$$\mathrm{Adv}^{\mathrm{prf}}_{\mathrm{MAC}}(\mathcal{A}') = \mathbf{Pr}\left(\mathcal{A}'^{\mathrm{MAC}_K} = 1\right) - \mathbf{Pr}\left(\mathcal{A}'^{\mathcal{RO}} = 1\right) . \tag{6.2}$$

As $\mathcal{A}'$ outputs 1 if and only if $\mathcal{A}$ mounts a successful forgery, we have

$$\mathbf{Pr}\left(\mathcal{A}'^{\mathrm{MAC}_K} = 1\right) = \mathbf{Pr}\left(\mathcal{A}^{\mathrm{MAC}_K} \text{ outputs } (M,T) \text{ such that } \mathrm{MAC}_K(M) = T\right) , \tag{6.3}$$

$$\mathbf{Pr}\left(\mathcal{A}'^{\mathcal{RO}} = 1\right) = \mathbf{Pr}\left(\mathcal{A}^{\mathcal{RO}} \text{ outputs } (M,T) \text{ such that } \mathcal{RO}(M) = T\right) . \tag{6.4}$$

The term of (6.3) equals $\mathrm{Adv}^{\mathrm{unf}}_{\mathrm{MAC}}(\mathcal{A})$. The term of (6.4) is a purely probabilistic event, and equals $|Q_f|/2^\ell$. We conclude from (6.2), (6.3), and (6.4) that

$$\mathrm{Adv}^{\mathrm{prf}}_{\mathrm{MAC}}(\mathcal{A}') = \mathrm{Adv}^{\mathrm{unf}}_{\mathrm{MAC}}(\mathcal{A}) - \frac{|Q_f|}{2^\ell} ,$$

or equivalently,

$$\mathrm{Adv}^{\mathrm{unf}}_{\mathrm{MAC}}(\mathcal{A}) = \mathrm{Adv}^{\mathrm{prf}}_{\mathrm{MAC}}(\mathcal{A}') + \frac{|Q_f|}{2^\ell} .$$

This completes the proof. □

6.3 CBC-MAC

We can build a MAC function from the CBC mode of Chapter 5. In CBC a ciphertext block C_i depends on all message blocks up to M_i. The idea of CBC-MAC is to use CBC encryption with $IV = 0$ and take the tag equal to the last ciphertext block, while at the same time throwing out the previous ciphertext blocks. The latter is important as failing to do this makes the MAC function easy to distinguish from a random oracle.

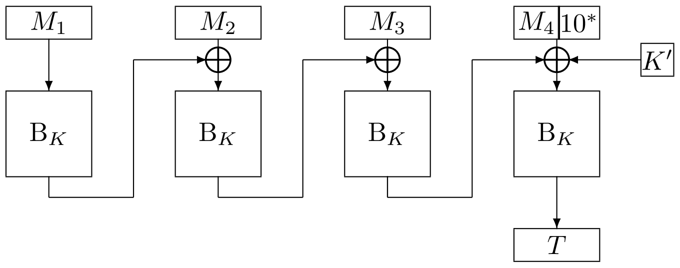


Figure 6.3: The mode CBC-MAC.

6.3.1 Security of CBC-MAC for fixed-length messages

CBC-MAC can be proven PRF secure if applied only to messages of the same length. Intuitively, if we consider any two distinct messages of the same length, they must have a "first message block" that is different. That first message block, say it is the i-th one, will incur a difference in the input to the i-th block cipher call. This difference makes any subsequent block cipher evaluations mutually independent as long as there will never be an input collision to any two block cipher calls. In the end, this means that the tags corresponding to the two messages look independent.

There is one catch in this reasoning: "as long as there will never be an input to any two block cipher calls." This reveals an exposition of the birthday paradox, namely that one cannot get a better security bound than $\binom{|Q_d|}{2}/2^n$. Indeed, for a random oracle there is no thing as input collisions; it outputs random strings for any query.

In fact, one can also distinguish CBC-MAC from a random oracle in computational complexity around $2^{n/2}$:

- Query multiple 2-block inputs $M^{(i)}$ of the form $M_1 \| M_2$, such that both the M_1's and the M_2's are distinct for all queries $M^{(i)}$;
- If we have a collision $B_K(M_1^{(i)}) + M_2^{(i)} = B_K(M_1^{(i)}) + M_2^{(i)}$, then we have colliding tags.

The attack is depicted in Figure 6.4. Note that an attacker may, indeed, find a desired collision in the birthday bound, i.e., in data complexity $|Q_d| \approx 2^{n/2}$.

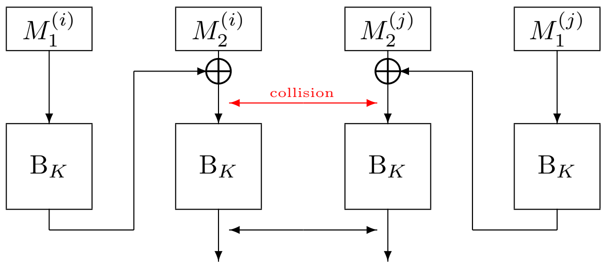


Figure 6.4: Distinguishing attack against CBC-MAC for fixed-length messages.

6.3.2 Insecurity of CBC-MAC for arbitrary-length messages

Unfortunately, CBC-MAC is trivially weak for arbitrary-length messages: we can distinguish it from a random oracle in two queries. If we query M_1, then we get $T = B_K(M_1)$. For the second query, we apply $M_1 \| M_2$ with $M_2 = M_1 \oplus T$. Then the resulting tag will be

$$\mathrm{B}_K(M_2 \oplus \mathrm{B}_K(M_1)) = \mathrm{B}_K(M_1 \oplus T \oplus \mathrm{B}_K(M_1)) = \mathrm{B}_K(M_1) = T\,.$$

So we can construct two messages with the same tag, while a random oracle would give two unrelated tags. (See also Figure 6.5.) This weakness is sometimes called the *length extension* weakness.

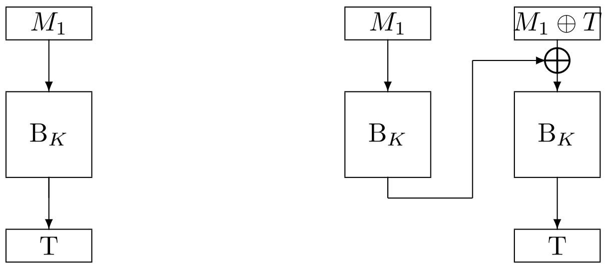


Figure 6.5: Weakness of CBC-MAC.

The attack as displayed here does not take into account that messages should be padded, but this can be accounted for. One could also truncate the tag, to have it be only the first so many bits of C_n (the last ciphertext block). This helps, but the security strength can be increased to at most $b/2$.

6.4 C-MAC

The problem with CBC-MAC is that we can apply length extension, i.e., first compute the MAC for a message, and after that for a slightly longer message. A fix for this attack consists of *doing something different at the end of the computation.* In this section we demonstrate this solution alongside the discussion of the MAC function C-MAC that was standardized by NIST in NIST special publication SP 800-38. Before going to this construction, we first discuss a simplified construction in Section 6.4.1.

6.4.1 Simplified construction

C-MAC is defined as CBC-MAC, with one difference at the end: C-MAC adds some subkey K' right before the last application of B_K. See Figure 6.6.

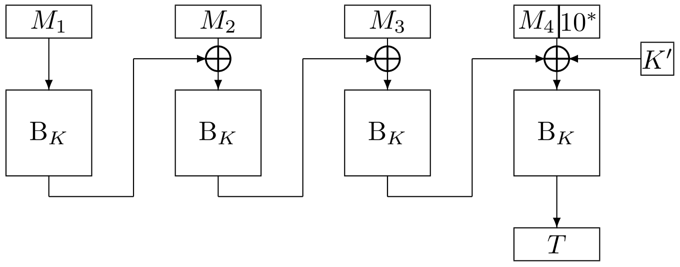


Figure 6.6: A simplified version of C-MAC.

The trick resolves the attack with variable-length messages. The reason for this is that the "last input" in each MAC computation is blinded. This means that this last input cannot be determined by the adversary anymore, and the attack fails.

Note, however, that the construction still achieves birthday-bound security at best. The attack of Section 6.3.1 carries over with the difference that one must focus on 3-block inputs $M^{(i)}$ of the form $M_1\|M_2\|M_3$, with the added notion that M_3 is the same for all $M^{(i)}$ while (M_1, M_2) is different in each $M^{(i)}$. (Verify this!)

6.4.2 NIST standard

The simplified construction of Figure 6.6 is close to a NIST standard, but not quite. Note that if this construction is evaluated on a message M of length exactly a multiple of b bits, one must pad it with 10^{b-1}. This means that the last evaluation of B_K only processes padding bits. Some might say that this is a waste of resources: if it has to process padding bits only, why do you need it in the first place? Because of this, NIST finds it necessary to do some kind of "domain separation" between the case where the message is of size exactly a multiple of b bits, and the case where it is not. The construction is given in Figure 6.7.

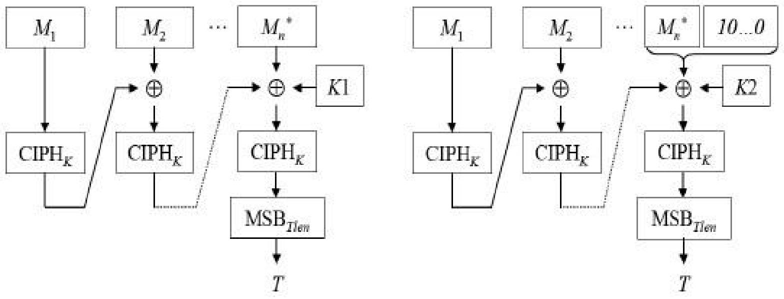


Figure 6.7: NIST standard C-MAC.

Chapter 7

Hash functions

A hash function h is a primitive that takes as input an arbitrary-length string and returns an n-bit *digest* H (also known as the hash result or sometimes just even hash), with n some fixed length:

$$\mathrm{h}(M) = H\,.$$

Figure 7.1 depicts a hash function.

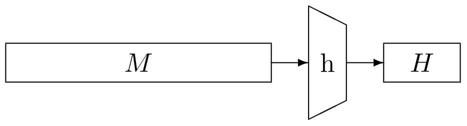


Figure 7.1: A schematic representation of a hash function.

In programming there are many use cases for hash functions. In most of these use cases, one hopes that the digest for different inputs look *unrelated*.

Cryptographic hash functions are a special class of hash functions that offer strength in the presence of adversaries. In this course we will only discuss that class of hash functions and for brevity we will just refer to them as hash functions.

Informally, a cryptographic hash function is secure if in an attack scenario it offers the same resistance as a random oracle, with output truncated to n bits, would. In other words, digests of different inputs appear random.

7.1 Applications and requirements

In this section we will treat the classical security requirements for hash functions by discussing some typical use cases.

7.1.1 Collision resistance

In several modern applications, including the version control management tool Git and the downloading software bittorrent, files or file fragments are identified by their digest. Their correct functioning relies on the unicity of this identifier: for each identifier there shall be only a single file. Hence, for these applications it is essential that no files can be found that hash to the same digest. These are not cryptographic applications but they do nicely illustrate the requirement of *collision resistance*. A cryptographic application that relies on collision resistance is that of modern digital signatures. Here the owner of a private key generates a digital signature over a document by applying a private key operation on the digest of that document. If multiple documents can be generated that have the same hash, the signature would be valid for all of them.

In more detail, we call a hash function *collision resistant* if it is infeasible to find distinct messages $M_1 \neq M_2$ with the same digest $\mathrm{h}(M_1) = \mathrm{h}(M_2)$.

Of course, any hash function will exhibit collisions due to the fact that its domain is larger (in principle infinite) than its range. However, generating collisions should be *computationally* hard.

For a random oracle, we can exactly quantify how hard it is by expressing the success probability of finding a collision as a function of the number of random oracle calls, that we will denote by $|Q_c|$. As the output of a random oracle is unpredictable, the best an adversary can do is try different inputs until a collision is found. After computing the digest for $|Q_c|$ messages, we have obtained $\binom{|Q_c|}{2}$ pairs of messages. The collision probability is simply determined by the birthday bound in the digest length n (see Section 15.5). This probability becomes close to 1 if $|Q_c|$ gets close to $2^{n/2}$. It follows that for a hash function the best possible collision resistance security strength that can be achieved is $n/2$.

7.1.2 Second pre-image resistance

One use case of hash functions is in the Windows Software Update system. Microsoft uses hash functions as part of their attempt to sign software updates that are broadcast. However, in 2012, the malware *Flame* was uncovered. It spread itself by acting as a properly signed Windows Software Update security patch. Intuitively, this should not be possible: signing should only be possible by Microsoft, that owns the signing key. So what happened here? It turned out that the Windows Updater used an old hash function, MD5 (see Section 7.5.2).

In 2007, Stevens et al. mounted collision attacks on this hash function. They showed that one can quite easily and efficiently construct collisions for MD5, or in other words, derive two different messages M_1, M_2 such that

$$\mathrm{h}(M_1) = \mathrm{h}(M_2)\,.$$

However, colliding messages would not be enough for the Flame malware writers to do its job. It required substituting the security patch signed by Microsoft by one of their own choice. In other words, given M_1 and $\mathrm{h}(M_1)$, they had to generate a malware file M_2 that has the same hash as M_1.

For a decent hash function this is a significantly harder challenge than generating a collision. But MD5 is not a decent hash function. It turned out that the developers of Flame exploited weaknesses of MD5 to do exactly that. They embedded their malware in one of the messages, M_2, and turned it into a collision with an original (and properly signed) earlier message M_1.

In more detail, we call a hash function *second pre-image resistant* if it is infeasible, given a message M, to find another message M' that has the same hash, i.e., $\mathrm{h}(M') = \mathrm{h}(M)$.

For a random oracle, the success probability of finding a second pre-image after $|Q_c|$ attempts is close to $|Q_c|/2^n$. The expected cost of finding a second pre-image is hence about 2^n, so the second pre-image resistance security strength of any hash function is at most n. Note that a second pre-image is just a special case of a collisions, but for a secure hash function it is much harder to achieve.

7.1.3 Pre-image resistance

In many cryptographic protocols one of the participants must *commit* to a value that it may reveal later. An example is the Schnorr zero-knowledge protocol that is treated in the lecture of this course on asymmetric cryptography. This commitment is typically implemented with a hash function: for a value x, the *prover* sends $y = \mathrm{h}(x)$ to the *verifier*. Later in the protocol the prover will reveal x to the verifier. If an adversary can compute x from $\mathrm{h}(x)$, she may be able to impersonate the legitimate prover.

We call a hash function *(first) pre-image resistant* if it is infeasible, given a digest H, to find a message M that hashes to H, i.e., $\mathrm{h}(M) = H$.

For a random oracle, the success probability of finding a pre-image after $|Q_c|$ attempts is close to $|Q_c|/2^n$. The expected cost of finding a pre-image is hence also about 2^n, so the pre-image resistance security strength of any hash function is at most n.

7.1.4 Keyed hashing

A very widespread use of hashing is password, or passphrase, protection. Suppose you register at a web service. You are asked to enter a user name, say “johndoe”, and a password, say “1qaz2wsx”.

user name	password
...	...
johndoe	1qaz2wsx
...	...

One way for the web service to store the data, is to store them in their database in plaintext. However, if hackers ever get their hands on their password database, your well-chosen password “1qaz2wsx” is leaked, and so are the passwords of thousands of other users. In an attempt to limit the consequences of password database breaches, web services store the *digest of the password* alongside the user name.

user name	h(password)
...	...
johndoe	fn34034859jfp23582
...	...

If an attacker ever obtains the password database, she only learns the *hashes of the passwords.*

For this application to work properly, the passwords shall be hard to guess. This is by itself something hard to achieve as users tend to choose easy to remember passwords and those are often also easy to guess. However, let us now assume for the sake of argument that users behave responsible and choose good passwords. Then the hash function used should be such that it is hard to compute the password from the digest. It should actually be the case that guessing passwords and hashing them is the most efficient attack. This reminds a bit of exhaustive key search, where the *key* is actually a (secret) password.

In Chapter 5 we described how to build stream ciphers from block ciphers and in Chapter 6 how to build MAC functions from block ciphers. Both can be built from hash functions as well.

For example, if h is a secure hash function, one can build a MAC function from it according to $\mathrm{MAC}_K(M) = \mathrm{h}(K\|M)$, where the digest H serves as tag, possibly after truncation.

One can also build a stream cipher $\mathrm{SC}_K(D, \ell) = Z$, with z_i computed as $\mathrm{h}(K\|D\|i) = z_i$.

Besides these, hash functions can be used for key derivation. We have a master key K_m and we want, given some diversifier D, to generate *base keys* keys K_i. The key derivation function is then $\mathrm{h}(K_m \| D) = K_i$.

This is often used in bank card payment systems. The central site of the bank has the master key K_m in a hardware security module (HSM), while K_i is in the chip of your bank card.

Very important to note is that it should be infeasible to recover K_m from knowing K_i and D.

The security of keyed hashing can be expressed as the infeasibility of distinguishing $\mathrm{h}(K\|\dot{)}$ from a random oracle.

7.2 Merkle-Damgård mode

In his 1979 PhD thesis, Merkle suggested an algorithm for h by building it from a fixed-input-length compression function, denoted as F. This function compresses an input of length $n + r$ to an output of length n.

7.2.1 Definition

The algorithm of Merkle is depicted in Figure 7.2. It is currently known as the “Merkle-Damgård construction”, for reasons that become apparent in Section 7.2.2.

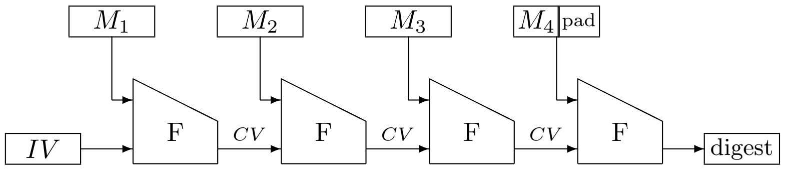


Figure 7.2: The Merkle-Damgård mode

The construction pads M injectively to a string of length a multiple of r, splits it into a sequence of r-bit blocks M_i and iteratively absorbs them into an n-bit chaining value CV:

$$CV \leftarrow \mathrm{F}(CV \| M_i)\,.$$

For absorbing the first block M_1, the construction requires that the CV is a fixed value, called *initial value IV*.

7.2.2 Collision preservation

Originally, the IV was not fixed. One might, for example, decide to fill these n bits with n bits of message, and therewith speed-up the hashing. However, it is easy to see that this leads to trivial second preimage attacks, even if F is a very strong function. For any evaluation $\mathrm{h}(M) = H$, one can remove the first block and set IV to be the first CV of the original computation.

In 1989, Damgård and Merkle independently proved that if the padding includes an encoding of the length of M and if the IV is fixed, the collision resistance of the hash function h follows from the collision resistance of the compression function F. In other words, a collision in h implies a collision in F.

7.3 Limitation to hash security: internal collisions

In this section we will discuss a property of the Merkle-Damgård mode that is in fact shared by all iterated hashing modes that operate on a chaining value of finite size.

The limitation to the security that an iterative hash function can offer is internal collisions. Suppose that one finds two different strings M, M' such that the chaining values CV, CV' after hashing the strings are equal: $CV = CV'$. Then, for any bitstring X, the messages $M\|X$ and $M'\|X$ yield the same hash result. The reason for this is that after processing of M and M', the resulting chaining values are equal, and as the subsequent message blocks (coming from X) are equal as well, the outputs of the compression function F are equal as well. The situation is depicted in Figure 7.3.

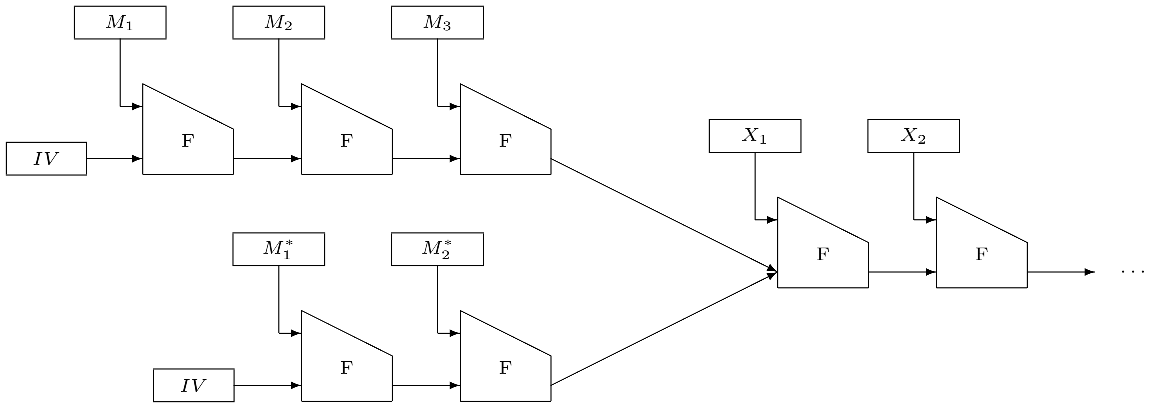


Figure 7.3: Internal collisions with Merkle-Damgård.

Hence an inner collision gives rise to an infinite family of colliding message pairs. This does not occur for a random oracle, and hence this property can be used to distinguish an iterative hash function from a random oracle. What "distinguishing" means in this context is more subtle than in Chapter 4, but for now the intuition from Chapter 4 suffices.

The distinguishing attack would work as follows:

- Determine some sequence of messages $M^{(i)}$ and a message X;
- Query the oracle (either the hash function h or a random oracle $\mathcal{RO}$) with $|Q_c|$ messages of the form $M^{(i)}\|X$;
- If there is no collision, then the oracle is most likely the random oracle;
- Otherwise, determine a new message X' and query $M^{(i)}\|X'$ and $M^{(j)}\|X'$ for the collision that was obtained;
- If the results are equal, then the oracle is most likely the hash function. Otherwise, it is most likely the random oracle.

The advantage of distinguishing the hash function from a random oracle in this way is approximately $|Q_c|^2 \cdot 2^{-(|CV|+1)}$.

The bottom line is that one can never get hashing security beyond the birthday bound on the size of the chaining value.

7.4 Weaknesses of the Merkle-Damgård mode

7.4.1 Second preimages

Generically, one expects that a second preimage attack (see Section 7.1) on a hash function can be mounted with complexity $|Q_c|/2^n$, where n is the output size. This was also believed to be the case for Merkle-Damgård by its advocates, before they understood the concept of internal collisions and the limitations it imposes on hash function security.

However, in the Merkle-Damgård construction, we have $|CV| = n$. In 2005, Kelsey and Schneier discovered a way to exploit this, and to obtain a second preimage attack on Merkle-Damgård hash function with a success probability higher than $|Q_c|/2^n$. For example, there exists a second preimage attack of a 2^d-block message that takes only about $2^{|CV|-d}$ evaluations of F. We explain this attack with help of Figures 7.4 and 7.5.

Suppose that we have the hash H of a message M of 2^d blocks (after padding). Our goal is to obtain a different M' with the same hash value. First, we select a single block message M', and compare the result after hashing it (Figure 7.5) with every CV_i in Figure 7.4. If d is very large, then the probability that there is a match is approximately, $2^d/2^n$, so the expected number of attempts we have to make is 2^{n-d}. If there is a collision between $\mathrm{h}(M')$ and chaining value CV_i, one extends M' with the blocks $M_{i+1}, \ldots, M_{2^d}$ in order to end up with the same digest as M:

$$\mathrm{h}(M) = \mathrm{h}(M' \| M_{i+1} \| \ldots \| M_{2^d}) .$$

Admittedly, the attack does not take into account length encoding yet. It can be resolved using a trick that can be found in literature but that is out of scope for the lecture notes.

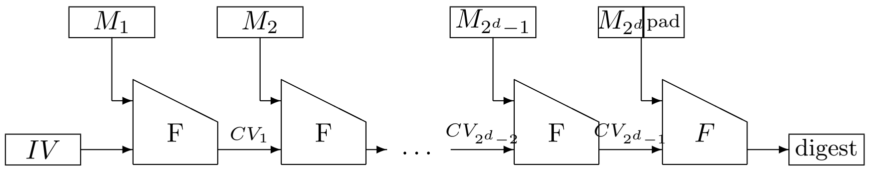


Figure 7.4: The situation of the attack.

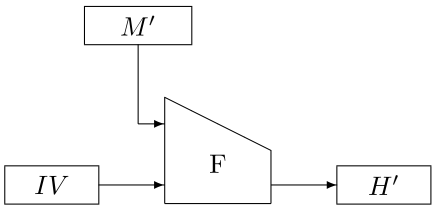


Figure 7.5: The trial part of the second preimage attack against Merkle-Damgård.

A remedy against the attack is to set the size of the chaining value larger than the size of the digest, for instance $2n$. This is called *wide-pipe* hashing. If F now has outputs of length $2n$, above attack only succeeds with probability around $2^d/2^{2n}$, and one must make on average $2^{2n-d} \geq 2^n$ attempts (provided that $2^d \leq 2^n$, which is reasonable to assume). A pleasant side effect of wide-pipe hashing is that the collision resistance preservation goes out the window.

The bottom line is that the birthday bound on the size of the chaining value also applies to second preimages.

7.4.2 Length extension attack

While the second preimage problem of Merkle-Damgård is essentially one of too great expectations, the length extension problem is a fundamental flaw that has given rise to painful fixes, similar to what we witnessed with DES being patched to Triple-DES.

The length extension attack relies on the observation that for Merkle-Damgård, the following structural property holds for any arbitrarily long M and r-bit X (with proper padding):

$$\mathrm{h}(M\|X) = \mathrm{F}(\mathrm{h}(M)\|X)\,. \tag{7.1}$$

7.5 Old style hashing standards

Back in the old days, the Merkle-Damgård approach was considered sound: it reduced the hard problem of building a collision resistant hash function to an easier problem, namely building a collision resistant fixed-input-length compression function. In practice, nobody ever found a way of building a collision resistant compression function from scratch. They were built as a mode of a block cipher or a cryptographic permutation using constructions that by themselves could be considered sound.

In this section, we explain how the idea finds use in the hash functions MD5 (Section 7.5.2), SHA-1 (Section 7.5.3), and SHA-2 (Section 7.5.4). These modes all use a compression function that is built on a block cipher in the so-called "Davies-Meyer mode" (Section 7.5.1).

7.5.1 Building a compression function: Davies-Meyer

The Davies-Meyer compression function uses a block cipher B inside, and compresses a chaining value CV_i and a message block M_i as

$$CV_{i+1} = \mathrm{B}_{M_i}(CV_i) \oplus CV_i\,.$$

The mode is depicted in Figure 7.6.

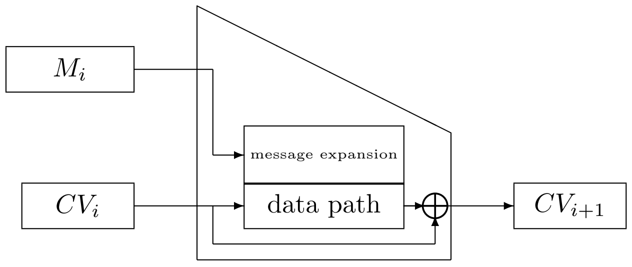


Figure 7.6: The Davies-Meyer mode.

We call the addition of CV_i at the end a *feedforward*. The feedforward is present to prevent collisions. Indeed, as we mentioned in Section 7.2.2, the Merkle-Damgård mode preserves collision resistance, but this is only meaningful if the underlying compression function is collision resistant. If the block cipher B is assumed to be an ideal cipher, one can prove that one requires at least $2^{n/2}$ evaluations of Davies-Meyer to obtain a collision. An ideal cipher is one that is chosen uniformly from the set of all block ciphers with the given parameters. (Try to figure out why Davies-Meyer *without* feedforward is *not* collision resistant.)

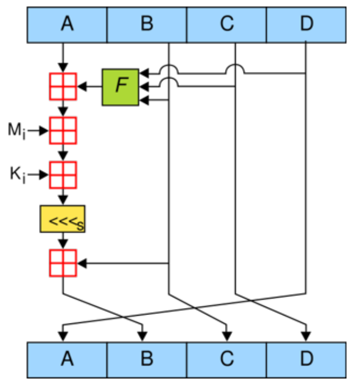


Figure 7.7: MD5 round function.

7.5.2 MD5

Rivest designed MD5 (*message digest* 5)in 1991 based on an earlier design of his hand, MD4. It uses the Merkle-Damgård mode with the Davies-Meyer mode. The digest of this hash function is 128 bits.

MD5 internals:

- 32 bit words;
- 4-word state;
- 16-word message blocks;
- 64 rounds, each taking one message word

The round function of MD5 can be seen in Figure 7.7.

The $\boxplus$ denotes integer addition, and the $\lll$ denotes a cyclic shift. Combining bitwise addition (XOR), integer addition and rotation gave hope that this hash function was strong.

In 1993 already, it was shown that F was weak. At that time MD5 was not adopted that much. Ten years later, there was made great progress in breaking MD5. Even with all this progress towards breaking MD5, many companies were unwilling to let MD5 go in exchange for a better alternative.

In 2005 Lenstra, Wang and De Weger use MD5 collisions to generate fake TLS certificates.

Finally in 2016 MD5 was mostly replaced by a version of SHA-2.

7.5.3 SHA-1

SHA-1 was then installed by NIST in 1995 drawing inspiration from MD5. The NSA designed SHA-1 that predictably uses the Merkle-Damgård mode with Davies-Meyer. It has a digest of 160 bits.

The internals of SHA-1 resemble those of MD5, except that it uses 80 rounds and 5-word states. In SHA-1, a message extension takes place following the rule:

$$w[i] = \begin{cases} m[i] & \text{if } i < 16; \\ (w[i-3] \oplus w[i-8] \oplus w[i-14] \oplus w[i-16]) \lll 1 & \text{if } i \geq 16. \end{cases}$$

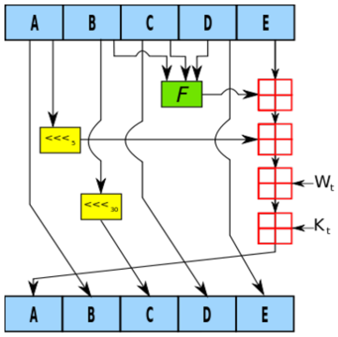


Figure 7.8: SHA-1 round function.

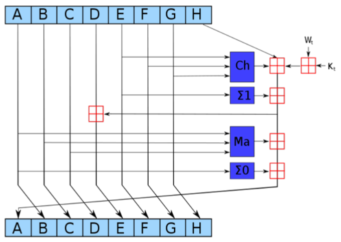


Figure 7.9: SHA-2 round function.

The round function can be observed in Figure 7.8. The similarity to MD5 is apparent.

Theoretical collision attacks with effort 2^{61} were published in 2004-2007, while in 2017 collisions were published at shattered.io by Marc Stevens et al.

7.5.4 SHA-2

In 2001 and 2008 the NSA designed SHA-2 as reinforced versions of SHA-1. The SHA-2 consists of 6 hash functions with digests of $224, 256, 384$ and 512 bits.

SHA-2 is a reinforced version of SHA-1 in the sense that we have 8-word states and a *nonlinear* message expansion. For SHA-256 and SHA-224 we have 32-bit words and 64 rounds.

In the other versions (SHA-512, SHA-384, SHA-512/256, SHA-512/224) we have 64-bit words and 80 rounds. The round function can be observed in Figure 7.9.

Since SHA-2 is built on Merkle-Damgård, length extension is a problem. However, at the moment there are no serious other threats.

7.6 Extendable Output Functions

In previous sections, the security strength achievable by a hash function against specific types of attack is determined by its digest length n. However, in many use cases the digest length is determined by the application and the desired security strength is fixed by real-world constraints. These use cases are addressed by a generalization of hash functions called *extendable output functions* (XOFs).

7.6.1 XOFs

A XOF allows generating output Z (the term digest is no longer appropriate) of arbitrary length. This can be implemented in different ways, but in our lecture notes we follow the convention that the function takes an input parameter ℓ that specifies the requested output length. So we have:

$$\text{XOF}(M, \ell) = Z\,.$$

Figure 7.10 depicts a XOF.

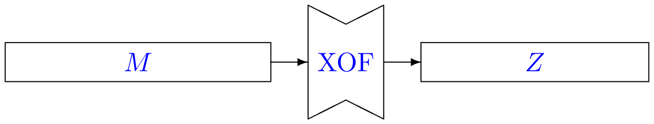


Figure 7.10: A schematic representation of a XOF.

Similarly to hash functions, a XOF is considered secure if it offers the same resistance as a random oracle would. However, in concrete XOF functions, the achievable security strength is upper bound by a parameter, usually called the *capacity*.

One single XOF can cover all use cases, since we can choose the output length. If we need multiple independent hash functions, we can just apply the XOF to the message $M\|0$ and $M\|1$, since they are considered to be independent, according to the security requirements from the introductions, specifically the fourth property. This process is called *domain separation* and can be generalized to create 2^w functions, using any $D \in \{0,1\}^w$ in:

$$\text{XOF}_D(M) = \text{XOF}(M\|D)\,.$$

7.6.2 Patching Merkle-Damgård

A simple way to turn Merkle-Damgård into a hash function with variable-length output is by repeating the hash computation multiple times while including a counter. This construction is called MGF1 and it is depicted in Figure 7.11.

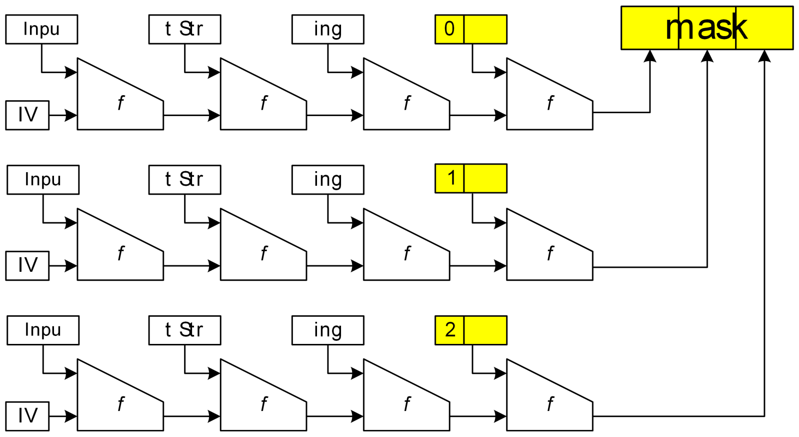


Figure 7.11: The mode MGF1 as a workaround XOF for Merkle-Damgård.

The computation can be done in parallel, since only the last block has to be processed multiple times (the remainder is the same in each evaluation).

7.6.3 Sponges

With lessons learned from the problems of the Merkle-Damgård designs, in 2007 Bertoni et al. introduced "sponge functions". The sponge construction uses a cryptographic permutation f of $b = c + r$ bits. It operates on a state consisting of an inner part of c bits (the capacity) and an outer part of r bits (the rate). Message M is injectively padded into r-bit blocks, and each block is compressed by adding it to the outer part and transforming the state using the cryptographic permutation f. This is called the "absorbing" phase. At the end, once all message blocks are processed, the "squeezing" phase begins. In the squeezing phase hash digest bits are taken from the outer part of the state r bits at a time, interleaved with evaluations of the cryptographic permutation f. This makes the sponge construction a XOF.

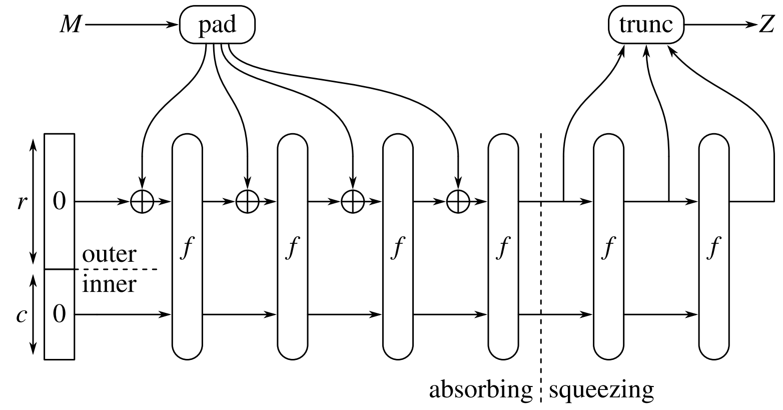


Figure 7.12: The sponge construction.

A first notable difference between the sponge construction and Merkle-Damgård is that the collision preservation property is released. The property was once considered to be very important, but in fact it is not essential: one can obtain a collision secure hash function even if the underlying compression function is not. This happens for the sponge construction. One can see the sponge as having the following compression function that is obviously not collision resistant (Verify this!):

$$\mathrm{F}(CV \| M) = f(CV \oplus (M \| 0^c)) \, ,$$

This is an important observation: the typical compression function used in Merkle-Damgård, Davies-Meyer of Section 7.5.1, was designed to be collision resistant and therefore came with a feed-forward that imposes an efficiency penalty. The efficiency penalty is eliminated here.

A second difference is that the cryptographic primitive within the sponge is not a block cipher, but rather a cryptographic permutation. This makes the mode conceptually simpler, noting that a block cipher was not developed to be used in hashing in the first place.

A third difference is that the sponge uses a b-bit state and that digest extraction is from part of the state only. It leaves c bits of the state "untouched", and therewith eliminates the second preimage and length extension attack of Section 7.3.

As a matter of fact, the capacity c determines the security strength of the sponge construction, as we will see in Section 7.7.4. It is equivalent to the size of the CV.

7.6.4 SHA-3

We do not include material on Keccak and SHA-3 in this version and for now, we refer to the introductory descriptions on `keccak.team` for further reading.

7.7 Security of hash functions

7.7.1 Distinguishing advantage

Inspired by Chapter 4, one might be tempted to model hash function security by bounding the advantage of an adversary $\mathcal{A}$ in distinguishing h from a random oracle $\mathcal{RO}$:

$$\mathrm{Adv}_{\mathrm{h}}^{\mathrm{ind}}(\mathcal{A}) = \Delta_{\mathcal{A}}(\mathrm{h}\ ;\ \mathcal{RO})\,. \tag{7.2}$$

This approach does not work. The reason for this is that in one of the two oracles in (7.2), no randomness is involved. More detailed, adversary $\mathcal{A}$ has access to either the hash function h (a mathematical function with a specification), or a random oracle (which is an abstract object without a specification). What this concretely means is that $\mathcal{A}$ can evaluate h offline on its own computer, and it can check if the outcomes match the responses from the oracle. If this is the case, it is conversing with h; if not, it is conversing with $\mathcal{RO}$.

So lesson one is that we must have to find some way to incorporate randomness in the "real world". One way to do so is to consider an abstracted version of the hash function, where a certain atomic building block is idealized. This way, h is viewed as a mode $\mathcal{M}$ that operates on top of a primitive $\mathcal{F}$. For example, $\mathcal{M}$ is the Merkle-Damgård construction and $\mathcal{F}$ is a compression function, or $\mathcal{M}$ is a sponge construction and $\mathcal{F}$ is a permutation. Then, the atomic building block $\mathcal{F}$ is idealized. This means that it is viewed as a random primitive. It results in the distinguishing game of Figure 7.13.

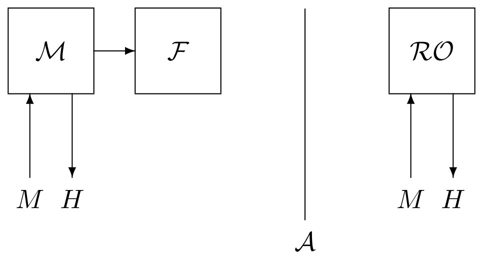


Figure 7.13: Distinguishing a hash mode from a random oracle (attempt 1).

Note that by idealizing the primitive of the hash construction, we cannot have any illusion of *actually proving security of a hash function*. The only thing we can achieve is proving that an idealized version of it, with the underlying primitive being idealized, is a secure hash function.

Back to the model of Figure 7.13: one might recognize a variant of Kerckhoff's principle in it. We do not want to rely on the secrecy of $\mathcal{F}$ for the security of our hash function. So we should allow our adversary $\mathcal{A}$ to query $\mathcal{F}$. By symmetrical reasons, this means that the adversary must also be able to query $\mathcal{F}$ in the ideal world. This leads to defining hash function security by bounding the advantage of an adversary $\mathcal{A}$ in distinguishing a hashing mode $\mathcal{M}$ based on ideal primitive $\mathcal{F}$ from a random oracle $\mathcal{RO}$ (see also Figure 7.14):

$$\mathrm{Adv}_{\mathcal{M}}^{\mathrm{ind}}(\mathcal{A}) = \Delta_{\mathcal{A}}(\mathcal{M},\mathcal{F}\ ;\ \mathcal{RO},\mathcal{F})\,. \tag{7.3}$$

Unfortunately, this model does not work either. *Any* mode $\mathcal{M}$ can be distinguished from $\mathcal{RO}$ in only a few queries. The adversary queries the mode $\mathcal{M}$ with some chosen message M, and then simulates the mode by making calls to $\mathcal{F}$ as described by the mode. In the real world, the outcomes will match; in the ideal world, they will very likely not match.

Note that the issues in the model come from the fact that in the ideal world, the primitive is "detached" from the construction. In security models for keyed modes (Chapter 4), the problem was absent by the secret key, as the reader can easily verify for him-/herself.

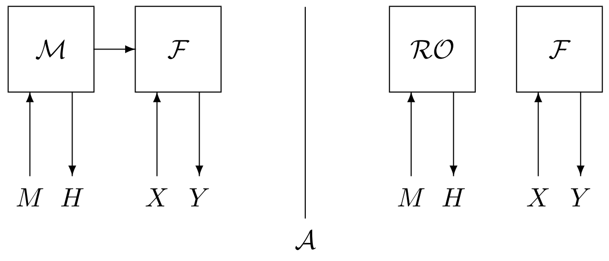


Figure 7.14: Distinguishing a hash mode from a random oracle (attempt 2).

7.7.2 Differentiating advantage

The model we are after at is that of "differentiating adversaries" of Maurer et al. This model replaces the function $\mathcal{F}$ in the ideal world by a *simulator* $\mathcal{S}$. See also Figure 7.15. This simulator gets access to the random oracle $\mathcal{RO}$, and its goal is to give outputs that are consistent with the random oracle. Indeed, in the real world $(\mathcal{M}, \mathcal{F})$ there is some relation among the queries and the adversary may actually "recompute" outcomes from oracle $\mathcal{M}$ with its oracle $\mathcal{F}$. The goal of the simulator is that if the adversary would "recompute" outcomes from $\mathcal{RO}$, these actually match with the outcomes of $\mathcal{S}$.

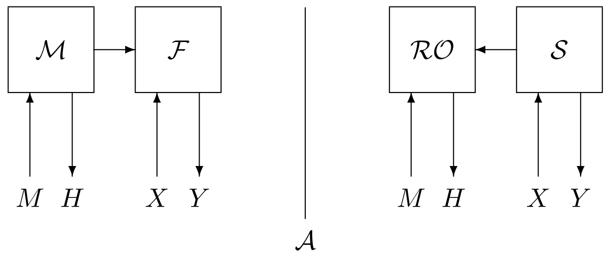


Figure 7.15: Differentiating a hash mode from a random oracle.

It is yet unclear how such a simulator actually looks like. The answer is, that it is specific to the mode: one develops a simulator $\mathcal{S}$ such that, hopefully, outcomes of $(\mathcal{M}, \mathcal{F})$ are hard to distinguish from $(\mathcal{RO}, \mathcal{S})$. Security bounds can then be proven for a given simulator $\mathcal{S}$.

We end up with the following definition of differentiating advantage.

Definition 7.7.1 (differentiability security)**.** Let $\mathcal{M}$ be a hashing mode based on random primitive $\mathcal{F}$. Let $\mathcal{RO}$ be a random oracle and $\mathcal{S}$ be a simulator. The advantage of an adversary $\mathcal{A}$ in differentiating $(\mathcal{M}, \mathcal{F})$ from a $(\mathcal{RO}, \mathcal{S})$ is defined as

$$\mathrm{Adv}^{\mathrm{iff}}_{\mathcal{M}}(\mathcal{A}) = \Delta_{\mathcal{A}}(\mathcal{M}, \mathcal{F}\ ;\ \mathcal{RO}, \mathcal{S})\,. \tag{7.4}$$

7.7.3 Merkle-Damgård is easy to differentiate from random

The length extension attack of Section 7.3 allows to differentiate plain Merkle-Damgård from random in three oracle queries.

The distinguisher has access to either (h, F) or $(\mathcal{RO}, \mathcal{S})$, and makes the following queries:

- Determine a message M and X;
- Query M to the hash oracle, and denote the result by H;
- Query $M' = M \| X$ to the hash oracle, and denote the result by H';

- Query $H\|X$ to the primitive oracle, and denote the result by Y;
- If $Y = H'$, the oracle is most likely (h, F). Otherwise, it is most likely $(\mathcal{RO}, \mathcal{S})$.

If the distinguisher interacts with (h, F), then $Y = H'$ holds by virtue of (7.1). In order for the simulator $\mathcal{S}$ to fool the distinguisher, it must guess H' correctly. But $H' = \mathrm{h}(M\|X)$, and the simulator never learns M (it only learns $H = \mathrm{h}(M)$), and it outputs $Y = H'$ with negligible property.

The attack is depicted in Figures 7.16-7.18. The attack succeeds in differentiating Merkle-Damgård from a random oracle in three queries.

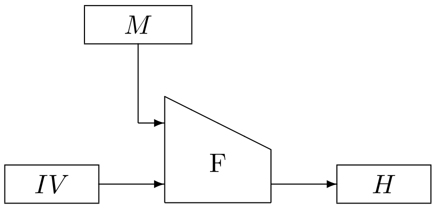


Figure 7.16: When h is Merkle-Damgård #1.

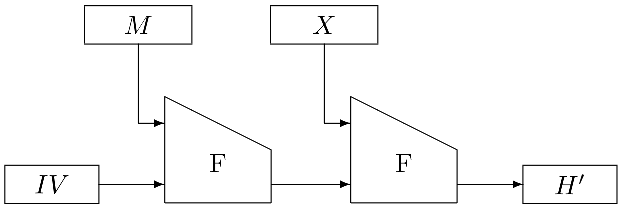


Figure 7.17: When h is Merkle-Damgård #2.

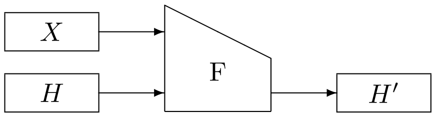


Figure 7.18: When h is Merkle-Damgård #3.

The crucial property of Merkle-Damgård that is exploited in the attack is that the chaining values can be obtained by querying the mode while choosing a suitable message. An easy fix for this weakness is making sure that the CVs can only be obtained from $\mathcal{F}/\mathcal{S}$ queries. For example, we could add some finalization to the last message block, as we did when fixing CBC-MAC in Section 6.4.

7.7.4 Sponge is hard to differentiate from random

The sponge construction, on the other hand, turns out to be indifferentiable. Bertoni et al. proved the following result.

Theorem 7.7.2 (Sponge indifferentiability)**.** *Consider the sponge construction based on random permutation f. There exists a simulator $\mathcal{S}$ such that for any adversary $\mathcal{A}$ with data complexity $|Q_d|$ blocks,*

$$\mathrm{Adv}^{\mathrm{iff}}_{\mathrm{sponge}}(\mathcal{A}) \leq \frac{|Q_d|^2}{2^{c+1}} \,.$$

The sponge thus solves all problems that the Merkle-Damgård mode had! If we consider the sponge with output truncated to n bits, above theorem also implies security against the following attacks as indicated:

- collision resistance: $\min(c/2, n/2)$;
- preimage resistance: $\min(c/2, n)$;
- second preimage resistance: $\min(c/2, n)$.

Unfortunately, Theorem 7.7.2 does not mean that a concrete hash function making use of the sponge construction is a secure hash function. It only implies that the construction *if instantiated with a random permutation*. Such a thing does not exist in reality; we are always stuck with an explicit permutation f.

Theorem 7.7.2 should thus be interpreted as stating that the sponge construction resists generic attacks. If f has some exploitable weakness, this may give rise to a weakness in the hash function. This is where cryptanalysts come in. They try to find flaws in the permutation f that would be exploitable when the permutation is used in the sponge function. This is not specific for sponge but for ALL hashing modes.

7.8 Message authentication from hashing

One can build message authentication from hash functions. As suggested in Section 7.1.4, one ideally constructs a MAC function from a hash function as

$$\mathrm{MAC}_K(M) = \mathrm{h}(K\|\cdot) \,. \tag{7.5}$$

In this section, we elaborate on the possibility of this construction.

7.8.1 Message authentication from Merkle-Damgård

For Merkle-Damgård, construction (7.5) does not work. This is due to the length extension attack of Section 7.4.2. Verify this!

There is a patch for message authentication using Merkle-Damgård: the HMAC mode in Figure 7.19. HMAC uses two passes of the hash computation. First, the secret key is used to derive two keys, an inner key and an outer key. This is done by two distinct constants *ipad* and *opad*:

$$\begin{aligned} K_{in} &= K \oplus ipad \,, \\ K_{out} &= K \oplus opad \,. \end{aligned}$$

The first pass of the hash produces an internal hash from the inner key and the message:

$$H = \mathrm{h}(K_{in} \| M)\,.$$

The second pass of the hash produces the output of the HMAC from the outer key and the internal hash:

$$\mathrm{HMAC}_K(M) = \mathrm{h}(K_{in} \| T)\,.$$

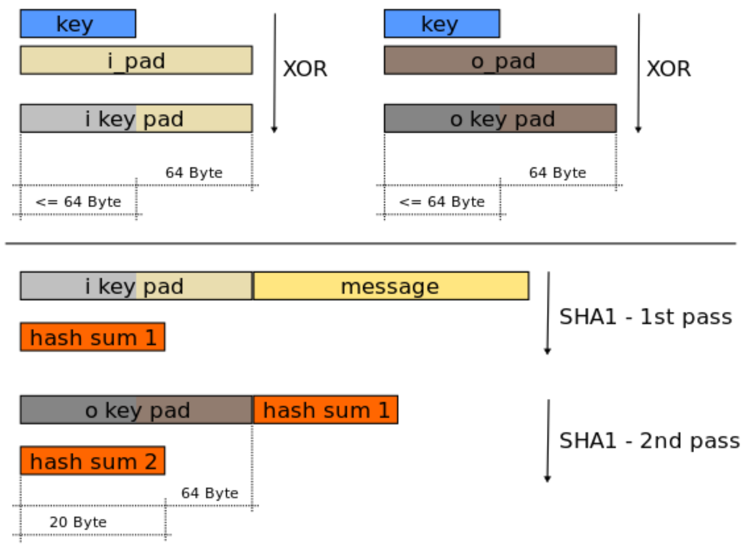
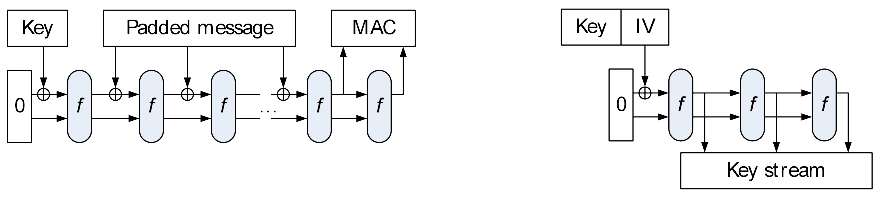


Figure 7.19: HMAC as a patch for the length extension weakness.

7.8.2 Message authentication from sponge

For the sponge, the construction (7.5) works. In fact, one can use the sponge construction likewise for the design of stream ciphers and key derivation functions. Figure 7.20 illustrates this.



(a) Message authentication.

(b) Stream encryption.

Figure 7.20: Modes for the sponge construction.

Chapter 8

Further reading

- The incentive of these lecture notes are not to give a fully theoretical understanding, nor to treat all possible more practical scenarios.
- These, in turn, appeared in other literature.
- Suggested further reading includes (and brief highlights of the books so the readers know which/why to choose):
  - **The Codebreakers** by David Kahn (on history)
    This book comprehensively chronicles the history of cryptography from ancient Egypt to the time of its writing (1967). However it does not cover the breaking of Enigma and the appearance of public cryptography.
  - **The Code Book** by Simon Singh (on history)
    A fascinating explanation of the history of cryptography. It covers stories from Mary, Queen of Scots, trapped by her own code, the Navajo Code Talkers. It includes the famous Enigma story of how Alan Turing and his brilliant team cracked the German code during WWII and continues with PGP.
  - **Handbook of Applied Cryptography** by Alfred Menezes, Paul van Oorschot and Scott Vanstone
    This book is encyclopedia in the field of Cryptography especially when it first appears during 1997. However, after more than 20 years, the book has not been updated. It still remains an interesting base.
  - **Everyday Cryptography: Fundamental Principles and Applications** by Keith Martin
  - **A Graduate Course in Applied Cryptography** by Dan Boneh and Victor Shoup (theoretical)
  - **Introduction to Modern Cryptography** by Mihir Bellare and Phillip Rogaway (theoretical)
  - **Elliptic Curves: Number Theory and Cryptography** by Lawrence Washington (very theoretical)
    A book focusing on elliptic curves, beginning at an undergraduate level (at least for those who have had a course on abstract algebra), and progressing into much more advanced topic.
  - **Modern cryptanalysis** by Christopher Swenson
    This book provides a foundation in traditional cryptanalysis, examines ciphers based on number theory, explores block ciphers, and teaches the basis of all modern cryptanalysis: linear and differential cryptanalysis.

– **The Design of Rijndael** by Joan Daemen and Vincent Rijmen (about AES)
  This books provide a quick math reminder before diving in the construction of Rijndael. It explains the design and rationale of AES while also proving security bounds.

Part II

Mathematics

Preface

This part of the lecture notes is considered with pure mathematics. All mathematics discussed here is needed somewhere in the course. We stress that the proofs are not considered part of the course material. They are included for the math-enthusiast and for completeness. Furthermore, some sections marked with an (*) are not considered part of the course material.

The exercises at the end of each section vary in difficulty. In no way are they ordered as such, easy and hard exercises are mixed. Exercsises marked with an (*) are again considered beyond the scope of the course.

Chapters 9 until 14 are considered Algebra and Number Theory, Chapter 15 is more-or-less standalone on Probability Theory.

As prerequisites we take little arithmetic (i.e. numbers, addition, multiplication, division, etc) and some understanding of set theory. (i.e., sets, functions, subsets, intersection, power sets, etc)

We are aware that a lot of the material covered here has also been discussed in other courses. These parts can be studied to a lesser extent in preparation for the exam if the material is known.

The mathematics needed for the course is best studied as follows:

1. Study the slides and exercises;
2. When there is something you don't understand, look it up in the lecture notes;
3. Start a few paragraphs earlier if it is still unclear.

When in doubt, it is always possible to send e-mails to any of the instructors.

Chapter 9

Integers and division

Everyone knows that we cannot divide 5 evenly by 2, or for that matter 1 by any number other than 1 itself. However, we can say that the quotient (a fancy word for a division) of 5 by 2 is 2, while the remainder (the part of 5 that we cannot divide by 2) is 1. Or in an expression

$$5 = 2 \times 2 + 1.$$

In this chapter we explain what integers (or whole numbers) are and how division on them might work.

9.1 What are integers?

In short, the set of integers is the set

$$\mathbb{Z} := \{\ldots, -2, -1, 0, 1, 2, \ldots\}$$

so, in particular any amount of 1s added together is an integer. Likewise, any amount of -1s added together is an integer.

Mathematically, one can define the integers rigorously, by first defining the natural numbers ($\mathbb{N} = \{1, 2, 3, \ldots\}$) inductively and then taking a quotient to an equivalence relation from the Cartesian product $\mathbb{N} \times \mathbb{N}$. This, however, goes beyond the scope of this text. For example, see *Getallen* by F. Keune (dutch).

We assume familiarity with addition, multiplication and subtraction of integers. So any reader can determine what

$$2 + 3$$

$$3 \times 17$$

$$123 + 321$$

$$14 \times 14$$

are, for example.

9.1.1 Rules for addition, multiplication and subtraction

Given any two integers a and b we know how to add them. For this addition we have some rules:

$$\begin{aligned} a+b &= b+a; \\ a+(b+c) &= (a+b)+c; \\ a+0 &= a. \end{aligned}$$

The first rule we call *commutativity*, we say that a and b commute under addition. The second is called *associativity.* The order in which you do the additions does not matter. Lastly, adding 0 to any integer has no effect. We call 0 the *neutral element* with respect to addition.

For the product of integers we also have some rules:

$$\begin{aligned} a\times b &= b\times a; \\ a\times(b\times c) &= (a\times b)\times c; \\ a\times 1 &= a. \end{aligned}$$

The similarity to the rules for addition should be clear. For example, the neutral element for multiplication is 1.

We also have the following rules concerning both addition and multiplication:

$$a\times(b+c) = a\times b + a\times c.$$

We say that multiplication is *distributive* over addition.

An example of this: $2\times(3+5) = 2\times 8 = 16$, but we can also say $2\times(3+5) = 2\times 3 + 2\times 5 = 6+10 = 16$. This can be a useful trick when multiplying larger numbers, for example:

$$7\times 123 = 7\times 100 + 7\times 20 + 7\times 3 = 700+140+21 = 861.$$

What's more, concerning addition, for every integer a there exists some integer b such that $a+b=0$. We write $-a$ for this integer b. Clearly, then with the notation of the integers given above, this *additive inverse* is easily found. Using this, we can define subtraction of integers:

$$a-b = a+(-b).$$

From

$$a+b-b = a$$

we see that subtraction is in a way the inverse operation of addition.

One might now wonder what the inverse operation to multiplication is, if it exists. One might shout division, and one would be partially right. Let us write a/b for a divided by b for now. Then

$$a\times b/b = a$$

indeed. But let us consider $1/2$. This does not make any sense in the world of integers. So clearly, division is not an operation on integers. We do have something that comes quite close, which we will discuss in the next section.

Also, from now on we will write $a\cdot b$ instead of $a\times b$, or sometimes even omit the $\cdot$ altogether and just write ab. (Of course when we mean $2\cdot 3$ we will not write 23.)

9.1.2 Ordering the integers

We need to define an order on the integers, so we can speak about which is larger and which is smaller. The natural order

$$\ldots < -2 < -1 < 0 < 1 < 2 < \ldots$$

will be the order we work with. The notation $<$ means *less than*, hence $>$ means *greater than.* We also have a notion $\leq$ for *less than or equal to* and $\geq$ for *greater than or equal to.* The special cases regarding 0 are as follows:

> 0 : is called *positive*;

≥ 0 : is called *non-negative*;

< 0 : is called *negative*;

≤ 0 : is called *non-positive.*

When a is positive (resp. non-negative), then $-a$ is negative (resp. non-positive). The number 0 is neither positive nor negative. (And 0 is both non-positive ánd non-negative.)

9.2 Division in $\mathbb{Z}$

The closest we get to division in $\mathbb{Z}$ is division with remainder. Before we discuss that, we talk about when an integer is (evenly) divisible by another integer. Everyone knows that one can evenly split 6 among 2, or 6 among 3.

9.2.1 Divisibility

With the above example in mind, we say that 6 *is divisible by* 2, or 2 *is a divisor of* 6. In particular we can list all (positive) divisors of 6:

$$1, 2, 3, 6.$$

We can also list the negative divisors of 6:

$$-1, -2, -3, -6.$$

These are all divisors of 6.

Definition 9.2.1. Let a, b be integers, then a is a divisor of b iff $b = na$ for some integer n. We write $a \mid b$. If a is not a divisor of b, then we write $a \nmid b$.

So we have $1 \mid 6, 2 \mid 6, 3 \mid 6$ and $5 \nmid 6$ for example.

Lemma 9.2.2. *Let a, b, c be integers. Then*

(a) If $a \mid b$, then $a \mid bc$.

(b) If $a \mid b$ and $a \mid c$, then $a \mid b + c$.

(c) If $a \mid b$ and $a \mid c$, then $a \mid b - c$.

Proof. In any mathematical proof it is wise to begin with what one knows. This can work both ways, from start and from finish. For example, when proving (a), we begin with $a \mid b$. We further know the definition of $a \mid b$, that is:

$$b = na$$

for some integer n. What we need to prove is that $a \mid bc$, using the definition, we know that this means that there exists some integer m for which $bc = ma$.

We also have $bc = cna$ by multiplying the first equation with c on both sides. Now $cna = ma$ and we are done since by setting $m = cn$ we have found the integer we were looking for. Below we write this shorthand, and also for the remaining statements:

(a) If $a \mid b$, then there exists some $n \in \mathbb{Z}$ with $b = na$. Hence $bc = cna$ and $a \mid bc$.

(b) If $a \mid b$ and $a \mid c$, then there exist some $n, m \in \mathbb{Z}$ with $b = na$ and $c = ma$. Then $b + c = na + ma = (n + m)a$ and $a \mid b + c$.

(c) If $a \mid b$ and $a \mid c$, then there exist some $m, n \in \mathbb{Z}$ with $b = na$ and $c = ma$. Then $b - c = na - ma = (n - m)a$ and $a \mid b - c$.

□

We have some more results on divisors, of which we leave the proofs as exercises for the reader.

Lemma 9.2.3. *Let a be an integer. Then*

(a) $a \mid a$;

(b) $a \mid 0$;

(c) $1 \mid a$;

(d) $a \mid 1 \iff a = \pm 1$.

Next we consider the statement

$$\text{If } a \mid c \text{ and } b \mid c\text{, then } a + b \mid c. \tag{9.1}$$

The hypothesis (or If-statement) can be translated into

$$c = na \text{ and } c = kb \text{ for some integers } n \text{ and } k.$$

We need to prove that there exists some integer l such that $c = l(a + b) = la + lb$. How do we find such an integer? Is it even real to expect that such an integer exists? Let's consider $a = 2$, $b = 3$ and $c = 6$. Then clearly, $2 \mid 6$ and $3 \mid 6$ but $5 \nmid 6$.

Clearly now, statement 9.1 is false, and the triple $(2, 3, 6)$ is called a *counterexample*.

9.2.2 Greatest common divisor

Since we can list for every integer b its positive divisors by just checking for each $0 < a \leq b$ whether it is a divisor or not, we can also list the negative divisors of b. Altogether we then get the complete list of divisors of an integer. From now on we usually are only interested in the positive divisors of an integer.

When one has two integers a and b, one can list the divisors of a and simultaneously list the divisors of b. These lists have some numbers in common. They both always include 1.

For example, when we list the positive divisors of 24 and 28 we have:

$$24 : 1, 2, 3, 4, 6, 8, 12, 24$$

and

$$28 : 1, 2, 4, 7, 14, 28.$$

We see that the *common divisors* are $1, 2$ and 4.

Definition 9.2.4. For every two integers a and b, their *greatest common divisor*, written $\gcd(a, b)$ is a common divisor d of a and b such that for all integers e that divide both a and b, we have $e \leq d$.

We call integers a, b for which $\gcd(a, b) = 1$ *coprime*.

By the above we have $\gcd(24, 28) = 4$. So for every two integers we can find their greatest common divisor by just writing down their divisors. In subsection 9.2.4 we will explain a more efficient way of finding the greatest common divisor of two integers.

Lemma 9.2.5. *Let a, b be integers. Then*

(a) $\gcd(a, a) = a$;

(b) $\gcd(a, 0) = a$;

(c) $\gcd(a, 1) = 1$;

(d) $\gcd(a, b) = \gcd(b, a)$.

Proof. (a) The greatest divisor of a is a, hence $\gcd(a, a) = a$.

(b) Since 0 is divisible by every non-zero integer and the greatest divisor of a is a, hence $\gcd(a, 0) = a$.

(c) The only positive divisor of 1 is 1 itself, which is a divisor of all integers, hence $\gcd(a, 1) = 1$.

(d) The common divisors of a and b are exactly the same as the common divisors of b and a, hence the greatest common divisor of a and b is equal to the greatest common divisor of b and a.

□

Furthermore, we have

$$\gcd(a, b) = \gcd(-a, b) = \gcd(-a, -b) = \gcd(a, -b).$$

For now, we just list some important properties of the greatest common divisor.

Proposition 9.2.6. *Let a, b, c be integers. Then*

$$\gcd(a, b) = \gcd(a + cb, b).$$

Proof. If d is a common divisor of a and b, then d is a divisor of $a + cb$ as well, by Lemma 9.2.2 (b). Conversely, if d is a common divisor of $a + cb$ and b, then d is a divisor of a as well, by Lemma 9.2.2 (c).

Thus a and b have the same common divisors as $a + cb$ and b, hence also have the same greatest common divisor. □

Remark. Of course the same holds in the second argument:

$$\gcd(a, b) = \gcd(a, b - ca).$$

Theorem 9.2.7. (Bézout's Identity) *Let a, b be positive integers. Then there exist integers x, y such that*

$$xa + yb = \gcd(a, b).$$

Proof. We prove this statement via the strong principle of induction on b.

Base of the induction: $b = 1$. Let a be a positive integer. By Lemma 9.2.5 we have $\gcd(a, 1) = 1$. Then let $x = 0$ and $y = 1$, such that $0 \cdot a + 1 \cdot 1 = 1$ as required.

Induction hypothesis: For all positive integers $n < b$ the statement:

$$\text{For all positive } a \in \mathbb{Z} \text{ there exist integers } x, y \text{ such that } xa + yb = \gcd(a, b).$$

is true.

Induction step: Let a be an arbitrary positive integer. Apply Proposition 9.2.6 with $c = -1$. Then we have

$$\gcd(a, b) = \gcd(a, b - a).$$

By the induction hypothesis, since $b - a < b$ we know that there exist integers x, y such that $xa + y(b - a) = \gcd(a, b - a) = \gcd(a, b)$.

Then we can rewrite the left-hand-side of the equation:

$$(x - y)a + yb = \gcd(a, b).$$

Since $x - y$ is again an integer, we are done. □

Not only do we have an efficient way of finding the greatest common divisor (see subsection 9.2.4) we can also efficiently find the integers x and y with an alteration of the algorithm discussed there.

The last theorem has some interesting corollaries:

Lemma 9.2.8. *Let a, b be integers. Then for any common divisor d of a and b we have*

$$d \mid \gcd(a, b).$$

Proof. By Theorem 9.2.7 we can find x, y such that

$$xa + yb = \gcd(a, b).$$

Then since $d \mid a$ and $d \mid b$, by Lemma 9.2.2 we find that $d \mid xa + yb = \gcd(a, b)$. □

Proposition 9.2.9. *Let a, b, c be positive integers. Then $\gcd(ca, cb) = c \gcd(a, b)$.*

Proof. Let d be a common divisor of a, b. Then cd is a common divisor of ca, cb. In particular for $\gcd(a, b)$ we find that $c \gcd(a, b)$ is a divisor of ca and cb. Hence by Lemma 9.2.8 we find that

$$c \gcd(a, b) \mid \gcd(ca, cb).$$

By Theorem 9.2.7 we can find x, y such that $xa + yb = \gcd(a, b)$. Then $cxa + cyb = c \gcd(a, b)$. By Lemma 9.2.2 and $\gcd(ca, cb) \mid cxa$ and $\gcd(ca, cb) \mid cyb$ we find

$$\gcd(ca, cb) \mid cxa + cyb = c \gcd(a, b).$$

Finally, by Exercise 9.E we are done. □

9.2.3 Division with remainder

Division of integers is not always possible, but when a is a divisor of b, then b/a is indeed an integer and dividing b through a makes sense. We now want to make this more general. For that we have *division with remainder*.

Proposition 9.2.10. *For all positive integers a, b there exist integers q and r such that*

$$b = qa + r$$

where $r < a$.

Proof. If $b < a$, then clearly $q = 0$ and $r = b$ satisfy the conditions set.

If $b = a$, then $q = 1$ and $r = 0$, which is also clearly a solution.

The remaining case where $b < a$, is the most cumbersome. But one can compare the list

$$a, 2a, 3a, 4a, \dots$$

to b and when na is larger than b, we know that $q = n - 1$. $\square$

We call q the *quotient* of a and b and r the *remainder*. Finding the q and r is called *division with remainder*.

Both the quotient and the remainder are unique. In each of the three above cases, we have determined the largest integer q such that $qa \leq b$. Since there is only one largest integer such that an inequality holds, the quotient is unique. Since the remainder is defined by $r = b - qa$, we find that it is also unique.

For example, we try to divide 256 by 31. We list $31, 2 \cdot 31, \dots$:

$$31, 62, 93, 124, 155, 186, 217, 248, 279, 310, \dots$$

We see that $8 \cdot 31 = 248 < 256$ and $9 \cdot 31 = 279 > 256$. So $q = 8$ and $r = 256 - 248 = 8$.

Division with remainder is just normal division (i.e., $r = 0$) when the larger is a multiple of the smaller.

The remainder of this section is devoted to long-division, which is an excellent tool to determine quotient and remainder more efficiently than just listing the multiples.

We use the example of when we divide 123456 by 789. It turns out that $123456 = 156 \cdot 789 + 372$. See below:

$$
\begin{array}{r}
156\\
789\overline{)123456}\\
\underline{789}\\
4455\\
\underline{3945}\\
5106\\
\underline{4734}\\
372
\end{array}
$$

So the quotient is displayed at the top, while the remainder is at the bottom.

The procedure goes as follows:

- 789 does not go into 1, not into 12, not into 123 but it does go into 1234 once.
- We subtract from 1234 the 789 and write in our quotient the 1. The result from the subtraction is 445. Since 789 does not go into 445, we add the 5 from above. Then 789 goes into 4455 five times.
- We subtract from 4455 the $5 \cdot 789 = 3945$ and write in our quotient the 5, which yields 15. Then 789 does not go into 510, but it does go into 5106, and exactly six times.

- We subtract from 5106 the $6 \cdot 789 = 4734$ and write in our quotient the 6, yielding 156. The remainder 372 is left at the bottom.

This procedure is based on the representation of digits in the decimal system. Using long division, one can also divide polynomials through each other, compute decimal expansions of fractions like $\frac{1}{7}$ and more. See below for the example of $\frac{1}{7}$:

$$
\begin{array}{rl}
 & 0.\overline{142857} \\
7 & \hspace{-0.5em}\big)\overline{1.000000} \\
 & \underline{7} \\
 & 30 \\
 & \underline{28} \\
 & 20 \\
 & \underline{14} \\
 & 60 \\
 & \underline{56} \\
 & 40 \\
 & \underline{35} \\
 & 50 \\
 & \underline{49} \\
 & 1
\end{array}
$$

The interested reader will be able to determine how this works for fractions. The notation $0.\overline{142857}$ means

$$0.142857142857142857142857142857142857142857\ldots$$

or, in other words, a repetition of the six digits 142857. The polynomial longdivision will be discussed later on.

9.2.4 The (Extended) Euclidean Algorithm

In this section, we discuss two algorithms, one, the Euclidean Algorithm, is an efficient way of computing the greatest common divisor of two integers. The other algorithm, the Extended Euclidean Algorithm, is an efficient way of not only computing the greatest common divisor of two integers, but in the process, also generates the integers x, y from Theorem 9.2.7. We begin with a discussion and example of the former.

The Euclidean Algorithm The Euclidean Algorithm draws heavily on division with remainder and Proposition 9.2.6. We first show the algorithm on an example. We want to determine the greatest common divisor of 65 and 23.

By Proposition 9.2.6, we know that $\gcd(65, 23) = \gcd(65 - 23, 23) = \gcd(42, 23) = \gcd(42 - 23, 23) = \gcd(19, 23)$. Again, $\gcd(19, 23) = \gcd(19, 23 - 19) = \gcd(19, 4) = \gcd(19 - 4 \cdot 4, 4) = \gcd(3, 4) = \gcd(3, 4 - 3) = \gcd(3, 1) = 1$.

Now we have not been as efficient as possible and haven't even used division with remainder yet. A more efficient way would be:

$$
\begin{aligned}
\gcd(65, 23) &= \gcd(65 - 2 \cdot 23, 23) = \gcd(19, 23) \\
\gcd(23, 19) &= \gcd(23 - 19, 19) = \gcd(4, 19) \\
\gcd(19, 4) &= \gcd(19 - 4 \cdot 4, 4) = \gcd(3, 4) \\
\gcd(4, 3) &= \gcd(4 - 3, 3) = \gcd(1, 3)
\end{aligned}
$$

Or, written more succinctly and more suitable to adapt to the Extended Euclidean Algorithm later on:

$$\begin{aligned}65 &= 2\cdot 23+19\\ 23 &= 1\cdot 19+4\\ 19 &= 4\cdot 4+3\\ 4 &= 1\cdot 3+1\\ 3 &= 3\cdot 1+0\end{aligned}$$

In each new line we use a division with remainder, and the last non-zero remainder is the greatest common divisor.

The Extended Euclidean Algorithm We begin this paragraph with an alternative proof to Theorem 9.2.7, which might be a little bit harder to understand, but proper understanding of this theorem gives an algorithmic way of finding the x, y. The difference is limited to the induction step.

Theorem 9.2.11. *Let a, b be positive integers. Then there exist integers x, y such that*

$$xa + yb = \gcd(a, b).$$

Proof. We prove this statement via the strong principle of induction on b.

Base of the induction: $b = 1$. Let a be a positive integer. By Lemma 9.2.5 we have $\gcd(a, 1) = 1$. Then let $x = 0$ and $y = 1$, such that $0\cdot a + 1\cdot 1 = 1$ as required.

Induction hypothesis: For all positive integers $n < b$ the statement:

For all positive $a \in \mathbb{Z}$ there exist integers x, y such that $xa + yb = \gcd(a, b)$.

is true.

Induction step: Let a be an arbitrary positive integer. We apply division with remainder to a and b and write $a = qb + r$. Then by Proposition 9.2.6 we find that

$$\gcd(a, b) = \gcd(a - qb, b) = \gcd(r, b).$$

Since $r < b$ we can determine integers u, v with $ur + vb = \gcd(r, b) = \gcd(a, b)$. Then also

$$u(a - qb) + vb = ua + (v - uq)b = \gcd(a, b),$$

where $v - uq$ is an integer, as required. □

We now extend the Euclidean algorithm to the Extended Euclidean Algorithm. We use the example from before:

$$\begin{aligned}65 &= 2\cdot 23+19\\ 23 &= 1\cdot 19+4\\ 19 &= 4\cdot 4+3\\ 4 &= 1\cdot 3+1\\ 3 &= 3\cdot 1+0\end{aligned}$$

So we have found that $\gcd(65, 23) = 1$. We remove the last line and rewrite the expressions:

$$19 = 65 - 2\cdot 23$$

$$\begin{aligned} 4 &= 23 - 1 \cdot 19 \\ 3 &= 19 - 4 \cdot 4 \\ 1 &= 4 - 1 \cdot 3 \end{aligned}$$

Then we "work bottom-up" and substitute the expressions we have found above into the first, simplifying as we go:

$$\begin{aligned} 1 &= 4 - 1 \cdot 3 \\ 1 &= 4 - 1 \cdot (19 - 4 \cdot 4) = -19 + 5 \cdot 4 \\ 1 &= -19 + 5 \cdot (23 - 19) = 5 \cdot 23 - 6 \cdot 19 \\ 1 &= 5 \cdot 23 - 6(65 - 2 \cdot 23) = -6 \cdot 65 + 17 \cdot 23 \end{aligned}$$

So this would mean that $-6 \cdot 65 + 17 \cdot 23 = 1$.

Below we present somewhat more efficient pseudocode for the EEA:

```
Data: Integers $a, b$
Result: Integers $d, x, y$ with $d = xa + yb$
1. $A \leftarrow [a, 1, 0]$;
2. $B \leftarrow [b, 0, 1]$;
3. while $B[1] \neq 0$ do
    (a) $C \leftarrow A - (A[1]\ div\ B[1])B$;
    (b) $A \leftarrow B$;
    (c) $B \leftarrow C$;
end
4. $d \leftarrow A[1]$, $x \leftarrow A[2]$, $y \leftarrow A[3]$;
return $d, x, y$.
```

Algorithm 1: Extended Euclidean Algorithm

In some literature the result of this procedure is called $X\text{gcd}(a, b)$. We prefer to call the x and y the Bézout coefficients.

9.3 Exercises

9.A List *all* divisors of the first ten positive integers.

9.B Prove Lemma 9.2.3.

9.C Is the following statement true or false?

$$\text{If } a \mid b \text{ and } b \mid c\text{, then } a \mid c.$$

When you deem the statement true, provide a proof, while if you think it is false provide a counterexample, i.e., give integers a, b, c such that $a \mid b$ and $b \mid c$ but not $a \mid c$.

9.D Find a *minimal* counterexample to statement 9.1. That is, find a positive triple (a, b, c) for which the statement does not hold such that for all triples (a', b', c') for which the statement does not hold, we have $a \leq a'$ and $b \leq b'$ and $c \leq c'$.

9.E Let m, n be positive integers. Suppose that $m \mid n$ and $n \mid m$. Prove that $m = n$.

9.F Use longdivision to compute the quotient and remainder when dividing 223092870 by 31.

9.G Compute $\gcd(625, 1147)$ using the Euclidean algorithm.

9.H Compute $\gcd(1147, 1209)$ and the Bézout coefficients (or $X\text{gcd}(1147, 1209)$) using the Extended Euclidean Algorithm.

9.I Compute $\gcd(1987, 1993)$ and the Bézout coefficients (or $X\gcd(1987, 1993)$) using the Extended Euclidean algorithm, explicitly following the pseudocode variant of the algorithm and writing down all steps.

9.J Let a, b be odd integers with $\gcd(a, b) = 1$. Show that $\gcd(a + b, a - b) = 2$. [Hint: Use Proposition 9.2.9.]

9.K Find integers x, y such that

$$x \cdot 89 + y \cdot 94 = 17.$$

9.L Given integers a, b we could also define $\gcd'(a, b)$ as a common divisor d of a and b such that for all common divisors e of a and b, we have $e \mid d$.

Show that $\gcd(a, b) = \gcd'(a, b)$.

9.M For each positive integer $1 \leq i \leq 15$ try to find an integer n with exactly i positive divisors.

Chapter 10

Prime numbers and factorization

This chapter is devoted to a special class of integers named *prime numbers* and the result that states that any positive integer can be written uniquely as a product of prime numbers.

10.1 Prime numbers

The first leg of the chapter is devoted to prime numbers. A *prime number* is a positive integer p such that p has exactly two positive divisors.

Note that the expression, "a positive integer p such that the only divisors of p are 1 and p" is not the same. The integer 1 satisfies the latter expression, but not the former, and is *not* a prime number.

Below we list the first 100 prime numbers:

2	3	5	7	11	13	17	19	23	29
31	37	41	43	47	53	59	61	67	71
73	79	83	89	97	101	103	107	109	113
127	131	137	139	149	151	157	163	167	173
179	181	191	193	197	199	211	223	227	229
233	239	241	251	257	263	269	273	277	281
283	293	307	311	313	317	331	337	347	349
353	359	367	373	379	383	389	397	401	409
419	421	431	433	439	443	449	457	461	463
467	479	487	491	499	503	509	521	523	541

(see Exercise 10.G.)

10.1.1 Infinitude of prime numbers

Definition 10.1.1. A positive integer p is called a *prime number* if p has exactly two positive divisors.

The 100 primes listed above are not the only primes, in fact, there are primes that have way more digits, for example, the largest known prime number to date, found by the Great Internet Mersenne Prime Search (see), is

$$2^{77232917} - 1,$$

a prime number comprising 23249425 digits. This number, although so large that it is already difficult to write down in its decimal notation, is still finite. And if we would naively read newspapers headlining "Largest prime number found" we could conclude that there exists no bigger prime. The following theorem contradicts this conclusion, as there are in fact infinitely many primes.

Theorem 10.1.2. (Infinitude of primes) *There are infinitely many prime numbers.*

Proof. Suppose that we have a finite list of prime numbers $p_1, p_2, \ldots, p_n$. Then we will show that there exists a prime number p that is not on this list. To that end, consider $P = p_1 \cdot p_2 \cdots p_n + 1$, the product of all the prime numbers in our list, plus 1. Then there are two possibilities:

P is a prime number, then we have found the prime that is not on the list, and we are done;

or

P is not a prime number. In that case there must exist some proper divisor, i.e., not equal to 1 or equal to P. Call this d. Then d is again either a prime, or not, in the latter case, we can find a new divisor, and keep doing so until we find a prime divisor of P.

So we may assume that d is a prime number. Then, suppose that d is on our finite list of primes. That means that $d \mid p_1 \cdot p_2 \cdots p_n$ by Lemma 9.2.2 and $d \mid p_1 \cdot p_2 \cdots p_n + 1$. Then by Lemma 9.2.2 we find that $d \mid 1$, a contradiction by Lemma 9.2.3.

Hence d is a prime number not already on our list. ☐

We give some examples. Suppose our prime list consists of only the numbers 2 and 3. Then $P = 2 \cdot 3 + 1 = 7$, which is a prime.

Now suppose we take as prime list $2, 3$ and 7, then $P = 2 \cdot 3 \cdot 7 + 1 = 43$ which is again prime.

Also including this in our list, yields $P = 2 \cdot 3 \cdot 7 \cdot 43 + 1 = 1807$ which is not a prime, but divisible by 13, (in fact $1807 = 13 \cdot 139$.)

The above theorem was at least known to Euclid in 300 BC already, and since then many different proofs have been found of the infinitude of primes.

Definition 10.1.3. Any positive integer that is not 1 and not a prime number is called a *composite number.*

10.1.2 Coprimality

Recall that two integers a and b are called *coprime* if $\gcd(a, b) = 1$.

Example 10.1.4. In particular, when $p \neq q$ are prime numbers, we have $\gcd(p, q) = 1$ and indeed p and q are coprime.

Proof. The divisors of p are $1, p$. The divisors of q are $1, q$. Since $p \neq q$, the only common divisor of p and q is 1, hence $\gcd(p, q) = 1$. ☐

We have another class of numbers coprime to a prime number, we leave the proof of this as an exercise to the reader:

Example 10.1.5. When $0 < n < p$ we have $\gcd(n, p) = 1$, so every such n is coprime to p. ♠

We have a more general statement:

Lemma 10.1.6. *Let p be a prime number and n an integer such that $p \nmid n$, then* $\gcd(n, p) = 1$.

Proof. The divisors of p are 1 and p. Since $p \nmid n$, the only common divisor of p and n is 1, hence $\gcd(n, p) = 1$. □

Theorem 10.1.7. (Euclidean lemma) *Let p, a and b be integers. If p is a prime number, then $p \mid ab$ implies that $p \mid a$ or $p \mid b$.*

Conversely if p is such that $p \mid ab$ implies $p \mid a$ or $p \mid b$, then p is a prime number.

Proof. Suppose that p is a prime number and $p \mid ab$. Suppose that $p \nmid a$, then by Lemma 10.1.6, p and a are coprime, hence using the Extended Euclidean Algorithm, we can find x, y such that

$$xa + yp = 1.$$

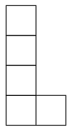
If we multiply this expression by b, we get

$$xab + ypb = b.$$

Now since $p \mid ab$, we find that both xab and ypb are divisible by p, hence $p \mid xab + ypb$. But then $p \mid b$, as required.

Conversely, suppose that p is not a prime number. Then we can write $p = m \cdot n$. Clearly now, $p \mid mn$, but $p \nmid m$ and $p \nmid n$. This completes the proof. □

An application of the Euclidean Lemma is the following. Suppose that we have an area of n meters by m meters and we have L-shaped tiles with dimensions as in the figure below:



Then this area can be tiled using these tiles only if $5 \mid n$ or $5 \mid m$, by the Euclidean Lemma:

The total area is nm square meters and the area of each such tile is 5 square meters, if we can tile this area, then nm should be multiple of 5, or $5 \mid mn$. Then since 5 is a prime number, we must have $5 \mid m$ or $5 \mid n$.

10.2 Factorisation

This section is mainly concerned with the *composite numbers* defined earlier. A composite number n can be written as $n = ab$ with $1 < a, b < n$. One can iterate this, by checking whether a or b is composite. For example, when we consider the number 18. Then clearly, $18 = 2 \cdot 9$. The number 2 is a prime, but 9 is not, in fact, $9 = 3 \cdot 3$. So we can write

$$18 = 2 \cdot 3 \cdot 3 = 2 \cdot 3^2.$$

This called the *factorisation* of the composite number 18.

10.2.1 Factorisation - Existence and uniqueness

Definition 10.2.1. Let a be an integer. If there exist a prime number p and a positive integer n such that $a = p^n$, then we call a a *prime power*.

Every integer can be factored as a product of prime-powers and in a unique way. This is the statement of the next theorem.

Theorem 10.2.2. (Fundamental Theorem of Arithmetic) *Let n be any positive integer. Then n can be factored as*

$$n = p_1^{e_1} \cdots p_k^{e_k},$$

where $p_1, \ldots, p_k$ are prime numbers and $e_1, \ldots, e_k$ are positive integers, for some k.

Moreover, this factorisation is unique (up to order), i.e., if we also have

$$n = q_1^{f_1} \cdots q_l^{f_l},$$

then $k = l$, $\{p_1, \ldots, p_k\} = \{q_1, \ldots, q_k\}$ and whenever $q_j = p_i$ we have $f_j = e_i$.

The proof of this theorem needs quite some work that we will not use later on, hence will be omitted.

As an example, we consider the number 729. This is both the square of 27 and the cube of 9. Hence we have

$$729 = 27^2$$

and

$$729 = 9^3$$

as possible factorisations. However, the factorisation should be unique, as a factorisation of prime powers. Indeed, both are equal to

$$729 = 3^6.$$

10.2.2 Connection with greatest common divisor

If we know the factorisation of two integers, finding their greatest common divisor is easy. Recall the factorisation of 18:

$$18 = 2 \cdot 3^2.$$

Then all divisors of 18 are the numbers of the form $2^i 3^j$ with $i = 0, 1$ and $j = 0, 1, 2$. Hence we have $2 \cdot 3 = 6$ divisors of 18:

$$1 = 2^0 \cdot 3^0, 3 = 2^0 \cdot 3^1, 9 = 2^0 \cdot 3^2,$$

$$2 = 2^1 \cdot 3^0, 6 = 2^1 \cdot 3^1, 18 = 2^1 \cdot 3^2.$$

Now we can easily extend this principle of factorisation to finding the greatest common divisor of two integers. Suppose that we have two integers a and b of which the factorisation is known, i.e.,

$$a = \prod_{i=1}^{n} p_i^{e_i},$$

and

$$b = \prod_{i=1}^{k} q_j^{f_j}.$$

We first define $P = \{p_1, \ldots, p_n, q_1, \ldots, q_k\}$. (This set does not necessarily have $n + k$ elements!) Then we can write

$$a = \prod_{p \in P} p^{e_p}$$

where $e_p = e_i$ if $p \in \{p_1, \ldots, p_n\}$ and $e_p = 0$ if $p \notin \{p_1, \ldots, p_n\}$. Similarly, we can write

$$b = \prod_{p \in P} p^{f_p}.$$

The greatest common divisor then is given by

$$\gcd(a,b) = \prod_{p\in P} p^{\min\{e_p, f_p\}}. \tag{10.1}$$

We first give some examples. Let the factorisation of $120 = 2^3 \cdot 3 \cdot 5$ be given. The greatest common divisor of 120 and 18 can then be found to be

$$2^{\min\{1,3\}} \cdot 3^{\min\{1,2\}} \cdot 5^{\min\{0,1\}} = 2 \cdot 3 = 6.$$

Another example, let $600 = 2^3 \cdot 3 \cdot 5^2$ and $250 = 2 \cdot 5^3$ be two integers with their factorisations. Then the primes that occur in the factorisations are $2, 3$ and 5. Since 2 occurs only once in 250 and thrice in 600, we know that it occurs once in $\gcd(600, 250)$. Since 3 does not occur in 250, it does not occur in $\gcd(600, 250)$. The prime 5 occurs twice in 600 and thrice in 250. Hence it occurs twice in $\gcd(600, 250)$. Indeed, $\gcd(600, 250) = 2 \cdot 5^2 = 50$. Written as above:

$$\gcd(600, 250) = 2^{\min\{3,1\}} \cdot 3^{\min\{1,0\}} \cdot 5^{\min\{2,3\}} = 2 \cdot 5^2.$$

10.2.3 Factorisation is not easy

Although the method to finding the greatest common divisor of two numbers by factoring them may seem like an easy approach, one must not forget that the factorisation must be known for this approach to work.

In particular, finding the factorisation of an integer is an extremely difficult problem, it is indeed so hard that we can base cryptographic systems on it, for example RSA.

Given, there are some numbers for which it is easy to find a prime factor:

- If the last digit of an integer is $0, 2, 4, 6, 8$, then the integer is divisible by 2.
- If the last digit of an integer is 0 or 5, then the integer is divisible by 5.
- If the sum of the digits of an integer is divisible by 3, then the integer is divisible by 3.
- If the last digit of an integer is 0, then the integer is divisible by 10.

All these can be verified experimentally, and the one concerning 3 will be proved at the end of Chapter 11 as a fun application. The others should be clear.

We give as an example the number 2761259439. Then the sum of the digits is:

$$2 + 7 + 6 + 1 + 2 + 5 + 9 + 4 + 3 + 9 = 48$$

and the sum of the digits of 48 is $4 + 8 = 12$, of which the sum of the digits is $1 + 2 = 3$, hence 12 is divisible by 3 and hence 48 is divisible by 3 and therefore 2761259439 is divisible by 3.

10.2.4 *Least common multiple

If we can consider

$$\prod_{p\in P} p^{\min\{e_P, f_P\}}$$

as in equation 10.1, we can of course also consider

$$\prod_{p\in P} p^{\max\{e_P, f_P\}}. \tag{10.2}$$

This is called the *least common multiple* of a and b.

Definition 10.2.3. Let a and b be integers. Then a *common multiple* of a and b is a positive integer m such that $a \mid m$ and $b \mid m$. We define the *least common multiple* of a and b, notation $\operatorname{lcm}(a, b)$ as a common multiple m such that for all common multiples n of a and b we have $n \geq m$.

One can verify that this definition coincides with equation 10.2.

With the formulas about factoring, one can easily see that

$$ab = \operatorname{lcm}(a, b) \gcd(a, b).$$

Indeed, when $a = \prod_{p \in P} p^{e_p}$ and $b = \prod_{p \in P} p^{f_p}$, then we have

$$ab = \prod_{p \in P} p^{e_p + f_p}$$

and

$$\operatorname{lcm}(a, b) \gcd(a, b) = \prod_{p \in P} p^{\min\{e_p, f_p\} + \max\{e_p, f_p\}}.$$

So what remains to prove, is that

$$e_p + f_p = \min\{e_p, f_p\} + \max\{e_p, f_p\},$$

which is clearly true.

The least common multiple of 6 and 12 is of course 12, the least common multiple of 17 and 19 is just $17 \cdot 19 = 323$ since $\gcd(17, 19) = 1$.

With least common multiples, we have the following proposition:

Proposition 10.2.4. *Let a, b, c be integers with $a \mid c$ and $b \mid c$. Then $\operatorname{lcm}(a, b) \mid c$.*

Proof. Since $a \mid c$ and $b \mid c$, we find that c is a common multiple of a and b. Therefore $\operatorname{lcm}(a, b) \leq c$. So we can do a division with remainder:

$$c = q \cdot \operatorname{lcm}(a, b) + r$$

where $0 \leq r < \operatorname{lcm}(a, b)$.

Now since $a \mid c$ and $a \mid \operatorname{lcm}(a, b)$, we have $a \mid c - \operatorname{lcm}(a, b) = r$. Similarly, we have $b \mid r$. So r is a common multiple of a and b. Since $\operatorname{lcm}(a, b)$ is the least common multiple of a and b, we find that $r = 0$. Hence indeed $\operatorname{lcm}(a, b) \mid c$. □

Corollary 10.2.5. *Let a, b, c be integers with $a \mid c$, $b \mid c$ and $\gcd(a, b) = 1$. Then $ab \mid c$.*

10.3 Exercises

10.A Prove the statement in Example 10.1.5.

10.B Factor 19409079690. [Hint: Use the tests described at the end of this chapter.]

10.C Factor 828202365628109119 using nothing more than a plain calculator. [Hint: You may want to give up after trying the first twenty or so primes for divisibility. Later in the course, this problem will turn up again anyway. (Exercise **??**.)]

10.D Explain/Prove the easy factoring options in Section 10.2.3. [Hint: We mean divisibility by 2, 5, 10.]

10.E Explain/Prove that when checking whether a positive integer is a prime number, it is enough to check prime divisors.

10.F Explain/Prove that when checking whether a positive integer n is a prime number, it is enough to check prime divisors up to $\sqrt{n}$.

10.G In the list of the first 100 prime numbers is one error. Which number is not prime? What is its factorisation?

10.H Which prime divisors does one need to check in order to find out whether 541 is prime or not? [Hint: see Exercise 10.F.]

10.I We define the factorial $n!$ on positive integers as:

$$n! := \begin{cases} 1 & \text{if } n = 1; \\ n \cdot (n-1)! & \text{else.} \end{cases}$$

So for example, $3! = 3 \cdot 2 \cdot 1 = 6$, $5! = 5 \cdot 4! = 5 \cdot 4 \cdot 3! = 120$.

Use your knowledge of the factorisation of 65! to determine the amount of zeroes at the end of 65!.

10.J Given integers a, b we could also define $\text{lcm}'(a, b)$ as a common multiple m of a and b such that for all common multiples n of a and b we have $m \mid n$.

Show that $\text{lcm}(a, b) = \text{lcm}'(a, b)$.

10.K Compute $\text{lcm}(35, 60)$ by first computing $\gcd(35, 60)$ and also by factoring 35 and 60.

Chapter 11

Modular Arithmetic

Integers are not the only numbers we can interact with, we can do this also with rationals (fractions), with real numbers or even complex numbers (for those who have knowledge of those.) But, perhaps more interestingly, there are many other things we can add, subtract and multiply. Here we discuss an important example.

11.1 Calculation modulo an integer

There are many examples where we calculate modulo an integer, the most well-known and perhaps most used example is the calculation modulo 12. This example comprises the first subsection, after which we will discuss the more general notion.

11.1.1 Telling time

When telling time, one looks at a clock mostly, whether that clock is digital, or analog, does not really matter. But let's consider the analog case first. When it's currently 5 o'clock, then in two hours time it will be 7 o'clock. But what will the time be in 10 hours time? It will not be 15 o'clock, but instead 3 o'clock:

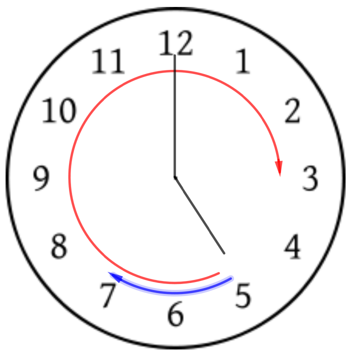


Figure 11.1: Calculating modulo 12.

What happens is that we do in fact calculate to 12, but then when further calculations are necessary, we set 12 equal to 0 and continue.

Indeed, $5 + 10 = 5 + 7 + 3 = 12 + 3$ "=" $0 + 3 = 3$.

Similarly, when it's 1 o'clock and someone ask us what time it was two hours ago, then we will not speak of -1 o'clock, but just give the answer as 11 o'clock.

Indeed, $1 - 2 = 1 - 1 - 1 = 0 - 1$ "=" $12 - 1 = 11$.

This example can be stretched to setting 24 to equal 0 when calculating on a digital clock. How does this work if we only care about the minutes part of a digital clock?

11.1.2 Precise definition

In the preceding introductory section we were calculating modulo 12. This means that we consider 12 to be equal to 0, and hence 13 equal to 1, 14 equal to 2,. . ., 23 equal to 11. But also -1 equal to 11 and so on.

Definition 11.1.1. Let n be a positive integer. Let a and b be integers. Then we say that a and b are *congruent modulo n* if $a = b + kn$ for some integer k. Or equivalently, if $n \mid a - b$. We write "$a \equiv b \pmod{n}$" for the phrase "a is congruent to b modulo n".

So above, we have seen that $5 + 10 \equiv 3 \pmod{12}$, and that $1 - 2 \equiv 11 \pmod{12}$.

More generally, if we want to know that a is modulo n, we can do division with remainder and find:

$$a = qn + r,$$

hence $a \equiv r \pmod{n}$.

One notices, for example, that when calculating modulo 12, there are only 12 numbers that can occur:

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.$$

We write $\mathbb{Z}/12\mathbb{Z}$ for the set of *integers modulo 12*.

Definition 11.1.2. Let n be an integer. Then we write

$$\mathbb{Z}/n\mathbb{Z} := \{0, 1, 2, \ldots, n - 1\}$$

for *set of integers modulo n*.

Remark. Sometimes, $\mathbb{Z}_n$ is written for $\mathbb{Z}/n\mathbb{Z}$ instead.

In order to define the integers modulo n rigorously, we need to make explicit that congruence modulo n is an equivalence relation, and consider the equivalence classes of the integers $0, \ldots, n - 1$ under this relation. See e.g. .

As we have seen in the example, we can add integers modulo 12. This phrase can be read in two different ways. As, we can add (integers modulo 12) or as, we can (add integers) modulo 12. We consider both ways to be equal.

The general procedure of adding integers a and b modulo an integer n is as follows. First determine the remainder r_a when dividing a through n and the remainder r_b when dividing b through n. Then proceed to calculate $r_a + r_b$ as usual and then determining the remainder $r_{r_a+r_b}$ when dividing $r_a + r_b$ through n.

The algorithm:

```
Data: Integers $a, b, n$
Result: An integer $c$ such that $a + b \equiv c \pmod{n}$.
1. Determine $q_a, r_a$ such that $a = q_a n + r_a$.
2. $a \leftarrow r_a$;
3. Determine $q_b, r_b$ such that $b = q_b n + r_b$.
4. $b \leftarrow r_b$;
5. $c \leftarrow a + b$;
6. Determine $q_c, r_c$ such that $c = q_c n + r_c$.
7. $c \leftarrow r_c$;
return $c$
```

Algorithm 2: Addition modulo n

For example when we want to calculate the sum of 123 and 456 modulo 78, we first write

$$123 = 1 \cdot 78 + 45$$

(which is just $123 \equiv 45 \pmod{78}$) and

$$456 = 5 \cdot 78 + 66$$

(again just $456 \equiv 66 \pmod{78}$). Then $45 + 66 = 111$, which is congruent to 33 modulo 78.

So $123 + 456 \equiv 33 \pmod{78}$.

Of course when $0 \leq a, b < n$, then steps 1-4 in the algorithm can be omitted.

For subtraction, we just note that $-a \equiv n - a \pmod{n}$. Hence we can always do addition:

$$17 - 12 \equiv 17 + (23 - 12) \equiv 17 + 11 \equiv 28 \equiv 5 \pmod{23}.$$

We note that for the addition as defined above, for any two $a, b \in \mathbb{Z}/n\mathbb{Z}$ we have $a + b \in \mathbb{Z}/n\mathbb{Z}$. And also the following rules hold:

$$a + b = b + a$$
$$a + (b + c) = (a + b) + c$$
$$a + 0 = a$$
$$a + (-a) = 0$$

When we multiply numbers modulo n, we follow the same procedure, but just replace the $+$ with a $\cdot$:

```
Data: Integers $a, b, n$
Result: An integer $c$ such that $ab \equiv c \pmod{n}$.
1. Determine $q_a, r_a$ such that $a = q_a n + r_a$.
2. $a \leftarrow r_a$;
3. Determine $q_b, r_b$ such that $b = q_b n + r_b$.
4. $b \leftarrow r_b$;
5. $c \leftarrow a \cdot b$;
6. Determine $q_c, r_c$ such that $c = q_c n + r_c$.
7. $c \leftarrow r_c$;
return $c$
```

Algorithm 3: Multiplication modulo n

For example, we want to multiply 123 and 456 modulo 78. By steps 1-4 of the algorithm, we need to calculate $45 \cdot 66$ modulo 78. First of all, $45 \cdot 66 = 2970$. Then $2970 = 38 \cdot 78 + 6$ and we have

$$123 \cdot 456 \equiv 6 \pmod{78}.$$

For integers modulo 6 we give here the addition and multiplication tables in Tables 11.1 and 11.2:

$+$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

Table 11.1: Addition table of $\mathbb{Z}/6\mathbb{Z}$.

$\cdot$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

Table 11.2: Multiplication table of $\mathbb{Z}/6\mathbb{Z}$.

$\cdot$	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11
2	0	2	4	6	8	10	0	2	4	6	8	10
3	0	3	6	9	0	3	6	9	0	3	6	9
4	0	4	8	0	4	8	0	4	8	0	4	8
5	0	5	10	3	8	1	6	11	4	9	2	7
6	0	6	0	6	0	6	0	6	0	6	0	6
7	0	7	2	9	4	11	6	1	8	3	10	5
8	0	8	4	0	8	4	0	8	4	0	8	4
9	0	9	6	3	0	9	6	3	0	9	6	3
10	0	10	8	6	4	2	0	10	8	6	4	2
11	0	11	10	9	8	7	6	5	4	3	2	1

Table 11.3: Multiplication table of integers modulo 12.

11.2 Division in Modular Arithmetic

Now that we have talked about addition, multiplication and subtraction of integers modulo n, we want to know how division works. What one notices especially when working with rational numbers (allowing fractions), then division is just multiplication with an inverse:

$$\frac{\frac{1}{2}}{\frac{3}{4}} = \frac{1}{2} \cdot \frac{4}{3}.$$

In modular arithmetic, division is exactly that same, multiplication with the inverse. However, not every number has an inverse.

11.2.1 Invertibility check

What is an inverse? When considering addition, the inverse of an integer a is the integer $-a$. Considering multiplication, we would like this to be $\frac{1}{a}$ or a^{-1}. We will write a^{-1} as preferred notation.

Definition 11.2.1. Let n be an integer and a an integer modulo n. Then a is *invertible* with *inverse* b if $ab \equiv 1 \pmod{n}$.

As mentioned above, not every number has an inverse. Suppose we want to check which elements are invertible modulo 12. Then we can construct the multiplicative table, as we did in Table 11.3. We see that there are four multiplications that yield 1: $1 \cdot 1 \equiv 1 \pmod{12}, 5 \cdot 5 \equiv 1 \pmod{12}, 7 \cdot 7 \equiv 1 \pmod{12}$ and $11 \cdot 11 \pmod{12}$.

Since $11 \equiv -1 \pmod{12}$ and of course $(-1) \cdot (-1) = 1$, the latter was expected.

We now describe an easy check for invertibility:

Lemma 11.2.2. *If* $\gcd(a,n)=1$, *then* a *is invertible modulo* n.

Proof. By Bézout's Identity (Theorem 9.2.7) we find that there exist $x, y \in \mathbb{Z}$ such that

$$xa + yn = 1.$$

Hence $xa \equiv 1 \pmod{n}$ and hence $x \pmod{n}$ is the inverse of $a \pmod{n}$. □

The converse is also true:

Theorem 11.2.3. *Let* n *be a positive integer. Then some* $0 \leq a < n-1$ *is invertible in* $\mathbb{Z}/n\mathbb{Z}$ *iff* $\gcd(a,n)=1$.

Proof. $\Leftarrow$:) is already proven in Lemma 11.2.2.

$\Rightarrow$:) Suppose a is invertible in $\mathbb{Z}/n\mathbb{Z}$. Then there exists some b in $\mathbb{Z}/n\mathbb{Z}$ such that $ab \equiv 1 \pmod{n}$. But by definition, that means that

$$ab + kn = 1$$

for some integer k. Let $d = \gcd(a,n)$. Then $d \mid ab$ and $d \mid kn$, hence $d \mid ab + kn$. Hence $d \mid 1$. But then $d = 1$, so $\gcd(a,n)=1$. □

We use this to check the invertible elements found in $\mathbb{Z}/12\mathbb{Z}$ (see Table 11.4).

a	$\gcd(a,12)$	invertible in $\mathbb{Z}/12\mathbb{Z}$?
0	12	
1	1	✓
2	2	
3	3	
4	4	
5	1	✓
6	6	
7	1	✓
8	4	
9	3	
10	2	
11	1	✓

Table 11.4: Invertible elements in $\mathbb{Z}/12\mathbb{Z}$.

We write $(\mathbb{Z}/12\mathbb{Z})^*$ for the set of invertible elements modulo 12. Clearly here $(\mathbb{Z}/12\mathbb{Z})^*$ has 4 elements.

11.2.2 Finding the inverse, using the Extended Euclidean Algorithm

Just knowing that an integer is invertible modulo n does not immediately give the inverse. No, instead, we must find it. In reference to Theorem 11.2.3 and Bézout's Identity (Theorem 9.2.7) we know that when a is invertible, we can find x, y using the Extended Euclidean Algorithm such that

$$ax + yn = 1,$$

hence we can find x such that $ax \equiv 1 \pmod{n}$.

As an example, suppose we want to find the inverse of 19 modulo 35. We perform the Euclidean algorithm:

$$\begin{aligned}35 &= 1 \cdot 19 + 16\\ 19 &= 1 \cdot 16 + 3\\ 16 &= 5 \cdot 3 + 1\\ 3 &= 3 \cdot 1 + 0\end{aligned}$$

So indeed, $\gcd(19, 35) = 1$. Then we perform the Extended part of the Euclidean Algorithm:

$$\begin{aligned}1 &= 16 - 5 \cdot 3\\ 1 &= 16 - 5 \cdot (19 - 1 \cdot 16) = -5 \cdot 19 + 6 \cdot 16\\ 1 &= -5 \cdot 19 + 6 \cdot (35 - 1 \cdot 19) = 6 \cdot 35 - 11 \cdot 19\end{aligned}$$

So $6 \cdot 35 + (-11) \cdot 19 = 1$.

Then we immediately obtain $(-11) \cdot 19 \equiv 1 \pmod{35}$. Hence the inverse to 19 modulo 35 is 24. Indeed

$$24 \cdot 19 = 456 = 350 + 106 = 350 + 70 + 36 = 350 + 70 + 35 + 1.$$

Furthermore, we also find that $-19 \equiv 16$ is the inverse to 11 modulo 35. The inverse of 6 modulo 19 is $35 \equiv 16$. (See also Exercise 11.J.)

Now that we can determine the inverse modulo n (when applicable) we can do division.

11.3 *Cute application of Modular Arithmetic

We have remarked that for an integer to be divisible by 3, the sum of its digits must be divisible by 3. The same statement is true where both 3s are replaced with 9s. Again, we prove the statement for 3 and leave the proof of the statement with 9s for the reader as an exercise.

Theorem 11.3.1. *Let n be an integer, then $3 \mid n$ iff 3 is a divisor of the sum of digits of n when written in its decimal form.*

Proof. We start by writing n in its decimal form as $a_{k-1} \cdots a_0$, i.e.,

$$n = a_0 + 10 \cdot a_1 + \ldots + 10^{k-2} \cdot a_{k-2} + 10^{k-1} \cdot a_{k-1}.$$

When working modulo 3 we need to prove that $n \equiv 0 \pmod 3$ iff $a_0 + a_1 + \ldots + a_{k-2} + a_{k-1} \equiv 0 \pmod 3$. Since for every $i \geq 0$ we have $10^i \equiv 1 \pmod 3$, this is now easy:

$$\begin{aligned}n \equiv 0 \pmod 3 &\iff a_0 + 10 \cdot a_1 + \ldots + 10^{k-2} \cdot a_{k-2} + 10^{k-1} \cdot a_{k-1} \equiv 0 \pmod 3\\ &\iff a_0 + a_1 + \ldots + a_{k-2} + a_{k-1} \equiv 0 \pmod 3\end{aligned}$$

as required. □

Theorem 11.3.2. *Let n be an integer, then $9 \mid n$ iff 9 is a divisor of the sum of digits of n when written in its decimal form.*

Proof. This is left for the reader in Exercise 11.L. □

11.4 Exercises

11.A Find for all pairs (a, n) below the inverse of a modulo n:

$$(65, 89)$$
$$(12, 19)$$
$$(11, 26)$$
$$(19, 26)$$

[Hint: Only one application of the Extended Euclidean Algorithm is needed, although a solution using three applications of the Extended Euclidean Algorithm is also accepted.]

11.B Which elements of $\mathbb{Z}/15\mathbb{Z}$ are invertible?

11.C Solve the equation $7 \cdot x + 5 \equiv 8 \pmod{13}$.

11.D Solve the equation $8 \cdot x \equiv 4 \pmod{12}$.

11.E Prove that when n is composite, that

$$(n-1)! \not\equiv -1 \pmod{n}.$$

11.F Prove that when p is a prime number, that

$$(p-1)! \equiv -1 \pmod{p}.$$

[This exercise and the previous together are known as Wilson's Theorem and were known c. 1000 AD.]

11.G Prove that when $n \neq 4$ is composite, that

$$(n-1)! \equiv 0 \pmod{n}.$$

11.H Calculate for the first five primes $(2, 3, 5, 7, 11)$ the squares of all elements in $\mathbb{Z}/p\mathbb{Z}$. What do you see?

11.I Calculate the squares of all elements in $\mathbb{Z}/n\mathbb{Z}$ for the $n \leq 10$ not treated in the above exercise. What do you see?

11.J From the equation

$$6 \cdot 35 + (-11) \cdot 19 = 1$$

which conclusions can you further get in terms of integers and their inverses?

11.K Solve the equation $2x + 2 = 5$ in $\mathbb{Z}/n\mathbb{Z}$ for all $n = 3, \ldots, 10$.

What do you notice about the number of solutions?

11.L Prove Theorem 11.3.2.

Chapter 12

Group Theory

Without knowing it, we have already met some structures (sets with operations) that are known as groups in mathematics. Here we study groups in their own right, and in the next chapter apply the theory to the examples we've met.

12.1 Definition and examples

We have already seen that we have an addition on $\mathbb{Z}$ that satisfies the following rules:

$$a + (b + c) = (a + b) + c;$$

$$a + 0 = 0 + a = a;$$

$$\text{For all } a \in \mathbb{Z} \text{ there exists some } b \in \mathbb{Z} \text{ such that } a + b = 0 = b + a.$$

These three rules together precisely define the notion of a group.

Definition 12.1.1. A *group* $(G, \star)$ is a set G together with an operation $\star$, that assigns to every two elements g, h of G an element $g \star h \in G$ for which the following rules hold:

(G1) For all $g_1, g_2, g_3 \in G$ we have $g_1 \star (g_2 \star g_3) = (g_1 \star g_2) \star g_3$.

(G2) There exists some $e \in G$ such that $g \star e = e \star g = g$ for all $g \in G$.

(G3) For every $g \in G$ there exists some $h \in G$ such that $g \star h = h \star g = e$.

We call e from (G2) the *group-identity* and given g we call the h from (G3) the $\star$-*inverse* of g. When $\star = +$, then $e = 0$ and $h = -g$. When $\star = \cdot$, then $e = 1$ and $h = g^{-1}$.

Example 12.1.2. Previously we have seen $(\mathbb{Z}, +)$, $(\mathbb{Z}/n\mathbb{Z}, +)$ as groups. But also $(\mathbf{Q}, +)$ and $(\mathbf{Q} \setminus \{0\}, \cdot)$ are groups. ♠

Example 12.1.3. $(\mathbf{Q}, \cdot)$ is not a group, as there exists no inverse to 0. Likewise, $(\mathbb{Z} \setminus \{0\}, \cdot)$ is not a group, since there exists no inverse to 2. ♠

In Definition 12.1.1 we speak about *the* group-identity and *the* inverse. To show that we do not misuse the word "the" here, we have:

Lemma 12.1.4. *Let $(G, \star)$ be a group.*

(a) There exists exactly one e such that $g \star e = e \star g = g$ for all $g \in G$.

(b) For every $g \in G$ there exists exactly one h such that $g \star h = e = h \star g$.

Proof. 1. Suppose that there exist e, f such that $g \star e = e \star g = g$ and $g \star f = f \star g = g$ for all $g \in G$. Then in particular:

$$e = e \star f = f.$$

(We use in the first equality the property that $g \star f = g$ for all $g \in G$ (including e) and in the second equality that $e \star g = g$ for all $g \in G$ (including f).)

2. This proof is similar, let $g \in G$ be arbitrary. Let h, h' be such that $g \star h = h \star g = e$ and $g \star h' = h' \star g = e$. Then in particular:

$$h = h \star e = h \star (g \star h') = (h \star g) \star h' = e \star h' = h'.$$

□

Note that in the examples of groups given before (see Example 12.1.2) all groups also satisfy $g \star h = h \star g$. This is such an important property that a group *may* have, that we have a special name for it:

Definition 12.1.5. A group $(G, \star)$ is called an *abelian* group if for all $g, h \in G$ we have $g \star h = h \star g$.

Earlier, the property $g \star h = h \star g$ for all g, h was called commutativity. Hence the term *commutative group* is technically correct. But to honor the mathematician Niels Hendrik Abel for his many contributions to group theory, we call these groups abelian.

There are also groups that are not abelian. For example, let $\mathrm{GL}_n(\mathbb{Z})$ be the set of all invertible $n \times n$ matrices with integer coefficients. Then for all $n \geq 2$ this is a non-abelian group, when considered multiplicatively, as $\det(AB) = \det(A)\det(B)$:

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \neq \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

To conclude this section, we show that for every positive integer n the set $(\mathbb{Z}/n\mathbb{Z})^*$ is a group under multiplication, and how to invert products of group elements.

Proposition 12.1.6. *Let $(G, \star)$ be a group and let $g, h \in G$ be group elements. Then*

$$(g \star h)^{-1} = h^{-1} \star g^{-1}.$$

(Or, when written additively $-(a + b) = -b + -a$.)

Furthermore, $(g^{-1})^{-1} = g$.

Proof. We have

$$\begin{aligned}(g \star h) \star (h^{-1} \star g^{-1}) &= g \star (h \star (h^{-1} \star g^{-1})) \\ &= g \star ((h \star h^{-1}) \star g^{-1}) \\ &= g \star (e \star g^{-1}) \\ &= g \star g^{-1} \\ &= e\end{aligned}$$

and of course similarly we get $(h^{-1} \star g^{-1}) \star (g \star h) = e$.

Since we know that inverses are unique (see Lemma 12.1.4) this proves the claim.

The second claim is left to the reader as Exercise 12.D. □

Example 12.1.7. We have met the set $(\mathbb{Z}/n\mathbb{Z})^*$ of invertible integers modulo n before. Since not all elements in $\mathbb{Z}/n\mathbb{Z}$ are invertible, we immediately find that $(\mathbb{Z}/n\mathbb{Z}, \cdot)$ is not a group.

We show that the invertible elements together with multiplication do indeed form a group. First we note that the multiplication of two invertible elements is again an invertible element, by referring to Proposition 12.1.6. Since multiplication of integers is associative, by remembering how multiplication was defined on integers modulo n, we know that multiplication is associative. Hence we satisfy (G1).

The group-identity for multiplication is 1, which is clearly an invertible element, and moreover $1 \cdot a = a \cdot 1 = a$ for all $a \in \mathbb{Z}/n\mathbb{Z}$, hence in particular for invertible elements. So (G2) is satisfied.

For (G3) we only need to note that when a is invertible with inverse b, which means $ab \equiv 1 \pmod{n}$ (and $ba \equiv 1 \pmod{n}$), then we immediately find that b is invertible with inverse a. Hence (G3) is satisfied.

Furthermore, $(\mathbb{Z}/n\mathbb{Z})^*$ is abelian, as multiplication of integers is commutative and hence by definition so is multiplication of integers modulo n.

So $(\mathbb{Z}/n\mathbb{Z})^*$ is an abelian group. ♠

12.2 Subgroups and isomorphisms

When we consider the group $(\mathbb{Z}, +)$, we can consider the set of even numbers

$$2\mathbb{Z} := \{n \in \mathbb{Z} \mid n = 2k \text{ for some } k \in \mathbb{Z}\}.$$

Then this is both a subset of $\mathbb{Z}$ and, using the same addition as we had on $\mathbb{Z}$, a group.

Definition 12.2.1. Let $H \subset G$ be a subset of a group $(G, +)$. Then H is called a subgroup of G, written $H < G$, if $(H, +)$ is a group.

When checking whether a subset is a subgroup, one does not have to check (G1), since the associativity holds for all elements in the group, hence certainly for the elements in the subset.

Something that is important to not forget, is that we must make sure that $g \star h$ is an element of H whenever $g, h \in H$. See the following example:

Example 12.2.2. Let A be the set of all odd integers together with 0. Then A is not a subgroup, even though properties (G0), (G1) and (G2) hold. Since $1 + 3 = 4$ is not an element of A, we find that $(A, +)$ is not a group. ♠

12.2.1 Subgroup tests and examples

In this section we list some examples of subgroups and give the subgroup tests, which make checking whether certain subsets are subgroups easier.

Proposition 12.2.3. (Subgroup test I) *Let $H \subset G$ be a subset of the group $(G, \star)$. Then H is a subgroup of G if:*

(H0) For all $h_1, h_2 \in H$ we have $h_1 \star h_2 \in H$;

(H1) $e \in H$;

(H2) For all $h \in H$ we have $h^{-1} \in H$. (Or when written additively: $-h \in H$.)

Proof. Suppose that H satisfies (H0)-(H2).

By the remark above, since (G1) is satisfied for G, it is immediately satisfied for H.

Then since G satisfies (G2) and we have (H1) H also satisfies (G2).

Lastly, G satisfies (G3) and since H satisfies (H2) we find that H also satisfies (G3).

So indeed H is a subgroup. □

We can reduce the checking a little bit further:

Proposition 12.2.4. (Subgroup test II) *Let $H \subset G$ be a subset of the group $(G, \star)$. Then H is a subgroup of G if:*

(H'0) H is non-empty;

(H'1) For all $h_1, h_2 \in H$ we have $h_1 h_2^{-1} \in H$. (Or when written additively $h_1 + -h_2 \in H$.)

Proof. Suppose that H satisfies (H'0) and (H'1). We need to show that H satisfies (H0)-(H2).

Since by (H'0) H is non-empty, there exists some $h \in H$. By (H'1) then $e = hh^{-1} \in H$, so (H1) is satisfied.

Now let $h \in H$ be arbitrary. Since $e \in H$ we find by (H'1) that $eh^{-1} = h^{-1} \in H$. Hence H satisfies (H2).

Lastly, let $h_1, h_2 \in H$ be arbitrary. Then $h_2^{-1} \in H$. By (H'1) we find that $h_1(h_2^{-1})^{-1} = h_1 h_2 \in H$, and hence H satisfies (H0).

So by Subgroup test I, H is a subgroup. □

Example 12.2.5. Let $H \subset \mathbb{Z}$ be of the form

$$H = n\mathbb{Z} := \{m \in \mathbb{Z} \mid m = kn \text{ for some } k \in \mathbb{Z}\}.$$

Then H is a subgroup of $(\mathbb{Z}, +)$.

Proof. Note that we can also write

$$n\mathbb{Z} = \{m \in \mathbb{Z} \mid n \mid m\}.$$

Then we use Subgroup test II. Since $n \mid n$ we see that $n\mathbb{Z}$ is non-empty. Hence (H'0) is satisfied.

Then let m_1, m_2 be such that $n \mid m_1, m_2$. By Lemma 9.2.2 (c) we find that $n \mid m_1 - m_2$, hence $m_1 - m_2 \in n\mathbb{Z}$ and $n\mathbb{Z}$ satisfies (H'1).

Therefore $n\mathbb{Z}$ is a subgroup of $\mathbb{Z}$. □

12.2.2 Isomorphic groups

Some groups look very similar, for example, $2\mathbb{Z}/4\mathbb{Z}$ and $\mathbb{Z}/2\mathbb{Z}$ have the following additive tables:

+	0	2
0	0	2
2	2	0

Table 12.1: Addition table of $2\mathbb{Z}/4\mathbb{Z}$.

+	0	1
0	0	1
1	1	0

Table 12.2: Addition table of $\mathbb{Z}/2\mathbb{Z}$.

One clearly sees that when replacing all 2s on the left by 1s, we get the table on the right. So these groups are very similar.

Definition 12.2.6. Let $(G,\star),(H,*)$ be two groups. We say that G and H are *isomorphic*, notation $G \cong H$, if there exists a bijective map $\mathrm{f}\colon G \to H$ such that $\mathrm{f}(e_G) = e_H$ and $\mathrm{f}(g_1 \star g_2) = \mathrm{f}(g_1) * \mathrm{f}(g_2)$. The map f is called an *isomorphism*.

Clearly, the map $\mathrm{f}\colon 2\mathbb{Z}/4\mathbb{Z} \to \mathbb{Z}/2\mathbb{Z},\ 0 \mapsto 0,\ 2 \mapsto 1$ is a group isomorphism. The rule $\mathrm{f}(g_1 \star g_2) = \mathrm{f}(g_1) * f(g_2)$ is just that the tables must coincide.

Example 12.2.7. Let n be a positive integer. Then $\mathbb{Z} \cong n\mathbb{Z}$. (Both groups considered additive.)

Proof. Let n be an arbitrary positive integer. We claim that the map

$$\mathrm{f}\colon \mathbb{Z} \to n\mathbb{Z},\ x \mapsto nx$$

is in fact an isomorphism. Therefore we need to check the following properties:

(I0) f is bijective;

(I1) $\mathrm{f}(0_{\mathbb{Z}}) = 0_{n\mathbb{Z}}$;

(I2) $\mathrm{f}(a+b) = \mathrm{f}(a) + \mathrm{f}(b)$.

When noting that in $n\mathbb{Z}$ the neutral element is also 0, and $\mathrm{f}(0) = n0 = 0$, we immediately have (I1). Next, let $a, b \in \mathbb{Z}$ be arbitrary. Then $\mathrm{f}(a+b) = n(a+b) = na + nb = \mathrm{f}(a) + \mathrm{f}(b)$ by distributivity of multiplication over addition in $\mathbb{Z}$. Hence indeed, this map satisfies (I2).

Lastly, we need to show that f is bijective. Note that every element in $n\mathbb{Z}$ is divisible by n, by construction. Hence the map

$$\mathrm{g}\colon n\mathbb{Z} \to \mathbb{Z},\ y \mapsto \frac{y}{n}$$

is a well-defined map. We need to show it is indeed the inverse to f. We have

$$(\mathrm{g} \circ \mathrm{f})(x) = \mathrm{g}(\mathrm{f}(x)) = \mathrm{g}(nx) = \frac{nx}{n} = x,$$

and

$$(\mathrm{f} \circ \mathrm{g})(y) = \mathrm{f}(\mathrm{g}(y)) = \mathrm{f}(\frac{y}{n}) = n\frac{y}{n} = y,$$

as required. Hence $\mathbb{Z} \cong n\mathbb{Z}$. □

The word isomorphic comes from the Greek words *isos* and *morphe*, meaning equal and shape. So two isomorphic groups have "equal shape". The following example illustrates this. (See also Figure 12.1.)

Example 12.2.8. We have an isomorphism $\mathbb{Z}/17\mathbb{Z}^* \cong \mathbb{Z}/16\mathbb{Z}$, which can be visualized in Figure 12.1. If we now want to multiply numbers in $\mathbb{Z}/17\mathbb{Z}^*$ with each other, we can use this isomorphism to our advantage.

For example, $11 \cdot 14 \pmod{17}$ can be found by calculating $7 + 9 \equiv 0 \pmod{16}$. Hence $11 \cdot 14 \equiv 1 \pmod{17}$. ♠

12.3 Order and Cyclic groups

This section introduces specific subgroups, namely cyclic subgroups, and then passes into order of groups (and elements).

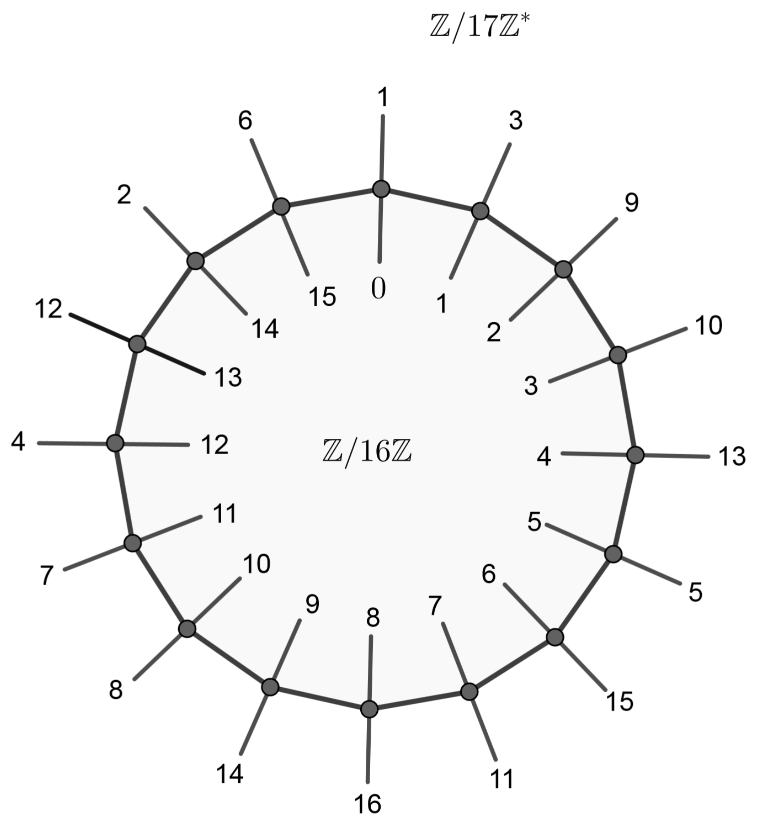


Figure 12.1: The isomorphism $\mathbb{Z}/17\mathbb{Z}^* \cong \mathbb{Z}/16\mathbb{Z}$ visualized.

12.3.1 Cyclic groups

Let G be a finite group and $g \in G$. If we consider the set

$$\{g^1, g^2, g^3, \ldots\}$$

then this set is a finite set, since $\operatorname{ord}(g)$ is finite. We write

$$\langle g \rangle = \{e, g^1, g^2, \ldots, g^{\operatorname{ord}(g)-1}\}$$

and call this the *set generated by* g.

It is straightforward to see that this $\langle g \rangle$ is a group. It is what we call a *cyclic group*.

Some examples of cyclic groups include $(\mathbb{Z}/n\mathbb{Z}, +)$ (generator 1).

Theorem 12.3.1. *Let G be a cyclic group of order n, generated by g. Then $G \cong \mathbb{Z}/n\mathbb{Z}$ as groups.*

Proof. We claim that $\mathrm{f}\colon \mathbb{Z}/n\mathbb{Z} \to G,\ a \mapsto g^a$ is an isomorphism.

1. $\mathrm{f}(0) = e$ clearly holds, since $g^0 = e$.

2. Let $a, b \in \mathbb{Z}/n\mathbb{Z}$ be arbitrary. Then

$$\begin{aligned} \mathrm{f}(a+b) &= g^{a+b} \\ &= g^a \cdot g^b \\ &= \mathrm{f}(a) \cdot \mathrm{f}(b) \end{aligned}$$

   So now f is indeed a homomorphism of groups.

3. Suppose that $a, b \in \mathbb{Z}/n\mathbb{Z}$ are such that $\mathrm{f}(a) = \mathrm{f}(b)$. That means that $g^a = g^b$. In particular then $g^{a-b} = e$. So $a - b = kn$ for some integer k. Hence $a - b \equiv 0 \pmod{n}$, hence $a \equiv b \pmod{n}$. So f is injective.

4. Since $\#G = \#\mathbb{Z}/n\mathbb{Z}$ we now have that f is a bijective homomorphism, hence an isomorphism.

Indeed, $G \cong \mathbb{Z}/n\mathbb{Z}$. □

As a direct corollary, any two (finite) cyclic groups of the same order are isomorphic.

We also have the following theorem, of which we omit the proof.

Theorem 12.3.2. *Let G be a cyclic group. Then any subgroup of G is cyclic.*

In particular, when G is cyclic of order n, then there exists a unique subgroup $H < G$ with $\#H = d$ iff $d \mid n$.

The proof of the above theorem relies on the following Lemma, which is useful in its own right.

Lemma 12.3.3. *Let G be a cyclic group of order n, with generator g. Then for every integer k the element g^k generates a subgroup of order $n/\gcd(k, n)$.*

Not all groups that we consider are cyclic. For instance, let $(\mathbb{Z}/12\mathbb{Z})^*$ be the group of invertible integers modulo 12. It consists of the elements $1, 5, 7, 11$, as we have seen before in Table 11.3. If this group would be invertible, we would have to be able to find a generator, thus an element of order 4.

But since $5^2 = 25 \equiv 1 \pmod{12}$ and $7^2 = 49 \equiv 1 \pmod{12}$, we find that there exists no one generator. Instead the group is generated by 5 and 7, with $5 \cdot 7 = 35 \equiv 11 \pmod{12}$.

In the next chapter, we will learn more about the structure of $(\mathbb{Z}/n\mathbb{Z})^*$.

Using Theorem 12.3.2, we can easily find the subgroups of, say $\mathbb{Z}/60\mathbb{Z}$. The divisors of 60 are

$$1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60.$$

Clearly, we have the subgroups

$$\{0\}, \mathbb{Z}/60\mathbb{Z},$$

as *trivial* subgroups. In particular, by Lemma 12.3.3 we know that in order to find the subgroup of order 2, we can just take the 30th power of the generator. Since we have an additive group, this means the 30th multiply of the generator, which is 1. Hence $30 \cdot 1 = 30$ is the generator of this subgroup of order 2. We list the subgroups in order of size:

$$\{0\}, 30\mathbb{Z}/60\mathbb{Z}, 20\mathbb{Z}/60\mathbb{Z}, 15\mathbb{Z}/60\mathbb{Z}, 12\mathbb{Z}/60\mathbb{Z}, 10\mathbb{Z}/60\mathbb{Z},$$

$$6\mathbb{Z}/60\mathbb{Z}, 5\mathbb{Z}/60\mathbb{Z}, 4\mathbb{Z}/60\mathbb{Z}, 3\mathbb{Z}/60\mathbb{Z}, 2\mathbb{Z}/60\mathbb{Z}, \mathbb{Z}/60\mathbb{Z}.$$

12.3.2 Order

Definition 12.3.4. Let G be a group. Then we call the number of elements of G, the *order of* G.

For example, the order of $(\mathbb{Z}, +)$ is infinite, while the order of $(\mathbb{Z}/n\mathbb{Z}, +)$ is n.

An important result which we shall not prove, is Lagrange's Theorem:

Theorem 12.3.5. (Lagrange's Theorem) *Let $(G, \star)$ be a finite group, and let $H < G$ be a subgroup of G. Then $\#H \mid \#G$.*

If we, for example have a group of 55 elements, and we are numerating the elements in a certain subgroup - upon reaching 12 distinct elements, we know that we are dealing with the entire group. (The divisors of 55 are $1, 5, 11, 55$.)

For example, in $\mathbb{Z}/55\mathbb{Z}$, the subgroup $\langle 5, 9\rangle$ contains all multiples of 5, of which there are eleven:

$$0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50$$

and also the number 9, hence this is the entire group.

Definition 12.3.6. Let G be a group and $g \in G$. The *order of g* is the smallest positive integer n such that $g^n = 1$, or when written additively $[n] \cdot g = 0$. We write $\operatorname{ord}(g) = n$.

There are elements that do not have an order as defined above, we say that the order of such an element is infinite. Luckily, most groups we consider in these lecture notes are finite.

Proposition 12.3.7. *Let G be a finite group and $g \in G$. Then* $\operatorname{ord}(g) = \#\langle g\rangle$.

Proof. Since $\langle g\rangle = \{e, g^1, g^2, g^3, \ldots\}$ has $\#\langle g\rangle$ elements, we know that

$$g^{\#\langle g\rangle} = e.$$

Then suppose that there exists a $k < \#\langle g\rangle$ such that $g^k = e$.

Then $g^{k+1} = g, g^{k+2} = g^2$ and more generally $g^{k+i} = g^i$ for all i. So indeed, $\#\langle g\rangle = k$, a contradiction.

So indeed $\#\langle g\rangle$ is the smallest positive integer n such that $g^n = e$. Thus $\operatorname{ord}(g) = \#\langle g\rangle$. □

We give some examples of the order of elements in some groups.

Example 12.3.8. Let $\mathbb{Z}/7\mathbb{Z}$ be the group of integers modulo 7. Since we consider this group as an additive group, the order of 5, say, is the smallest n such that $[n] \cdot 5 \equiv 0 \pmod 7$.

Since $\gcd(5, 7) = 1$, we know that 5 is invertible. Hence $n \equiv 0 \pmod 7$. Hence the smallest positive integer n such that $[n] \cdot 5 \equiv 0 \pmod 7$ is $n = 7$. ♠

Example 12.3.9. Let $\mathbb{Z}/8\mathbb{Z}$ be the group of integers modulo 8. Then we list the elements and their orders in the following table: ♠

Example 12.3.10. Let $\mathbb{Z}/8\mathbb{Z}^*$ be the group of invertible integers modulo 8. We first write down all elements of this group:

$$1, 3, 5, 7.$$

Since this group contains 4 elements, by Lagrange, we find that the order of each element is either $1, 2$ or 4. The order of 1 is 1, the order of 3 is 2 as $3^2 = 9 \equiv 1 \pmod 8$. The order of 5 is also 2: $5^2 = 25 \equiv 1 \pmod 8$. The order of 7 is 2, since $7 \equiv -1 \pmod 8$. And indeed $7^2 = 49 \equiv 1 \pmod 8$. ♠

An important property to remark is that order is preserved under isomorphism, i.e., if $\mathrm{f}\colon G \to H$ is an isomorphism and $g \in G$ has order n, then $\mathrm{f}(g)$ also has order n in H.

n	ord(n)
0	1
1	8
2	4
3	8
4	2
5	8
6	4
7	8

12.4 Exercises

12.A Prove that the set $\{0\}$ is a group with the usual addition on integers, i.e., $0 + 0 = 0$.

12.B Let S be a set and $\mathcal{B}(S)$ be the set of all bijections on S:

$$\mathcal{B}(S) := \{\mathrm{f}\colon S \to S \mid \mathrm{f} \text{ is bijective }\}.$$

Show that $\mathcal{B}(S)$ is a group.

12.C Let $\mathrm{GL}_n(\mathbb{Z})$ be the group of invertible $n \times n$ matrices with coefficients in $\mathbb{Z}$.

(a) Show that the following subset is a subgroup of $\mathrm{GL}_n(\mathbb{Z})$:

$$\mathrm{Dia}_n(\mathbb{Z}) := \{M \in \mathrm{GL}_n(\mathbb{Z}) \mid M_{ij} = 0 \text{ for } i \neq j\}.$$

(b) What is the order of $\mathrm{Dia}_n(\mathbb{Z})$?

(c) Show that the following subset is a subgroup of $\mathrm{Dia}_n(\mathbb{Z})$:

$$\mathrm{Sca}_n(\mathbb{Z}) := \{M \in \mathrm{GL}_n(\mathbb{Z}) \mid M_{ii} = M_{jj} \text{ for all } i \text{ and } j\}.$$

(d) What is the order of $\mathrm{Sca}_n(\mathbb{Z})$?

[Hint: When is an $n \times n$ matrix invertible?]

12.D Prove the second claim in Proposition 12.1.6.

12.E Show that $3\mathbb{Z}/12\mathbb{Z} := \{0, 3, 6, 9\}$ is a subgroup of $\mathbb{Z}/12\mathbb{Z}$.
Also show that $2\mathbb{Z}/12\mathbb{Z} := \{0, 2, 4, 6, 8, 10\}$ is a subgroup of $\mathbb{Z}/12\mathbb{Z}$.

12.F Determine the order of 17 in $(\mathbb{Z}/35\mathbb{Z}, +)$.

12.G Determine the order of 17 in $(\mathbb{Z}/35\mathbb{Z}^*, \cdot)$. [Hint: Even though we have not determined the order of $\mathbb{Z}/35\mathbb{Z}^*$ yet, you may use that it is 24.]

12.H Let G be a group such that $g^2 = e$ for all $g \in G$. Show that G is abelian.

12.I Let G be a group with an odd number of elements of order 2. Show that $\#G$ is even.

12.J Let G be a group with $\#G$ even. Show that G has an odd number of elements of order 2.

12.K *(Subgroup Test III)* Let G be a *finite* group. Show that $H \subset G$ is a subgroup if and only if

(H"0) $H \neq \emptyset$;

(H"1) for all $x, y \in H$ we have $xy \in H$.

[Hint: Use Subgroup Test I.]

12.L Let G be a group with subgroups $H, K < G$. Show that $H \cup K$ is a subgroup of G iff $H \subset K$ or $K \subset H$.

12.M Consider the group $\mathrm{GL}_2(\mathbb{Q})$ of invertible matrices with coefficients in $\mathbb{Q}$. Find matrices A, B such that $\operatorname{ord} o(A)$ and $\operatorname{ord} o(B)$ are finite, but $\operatorname{ord} o(AB)$ is infinite.

12.N How many subgroups does $\mathbb{Z}/128\mathbb{Z}$ have? What are they?

12.O Write down the isomorphism between $\mathbb{Z}/11\mathbb{Z}^*$ and $\mathbb{Z}/10\mathbb{Z}$ as in Figure 12.1. Use this to calculate the orders of $4, 8, 10$ in $\mathbb{Z}/11\mathbb{Z}^*$.

12.P Let $(G, \star)$ and $(H, *)$ be two groups. Show that $(G \times H, \circ)$ is a group when

$$(g_1, h_1) \circ (g_2, h_2) = (g_1 \star g_2, h_1 * h_2).$$

Also, show that when G and H are both abelian, that $G \times H$ is abelian.

Chapter 13

Elementary Number Theory

This chapter is mainly devoted to the group $(\mathbb{Z}/n\mathbb{Z})^*$ and exponentiation (= repeated multiplication) in $\mathbb{Z}/n\mathbb{Z}$. This will yield another way of finding the inverse to an invertible element.

13.1 Invertible elements in $\mathbb{Z}/n\mathbb{Z}$

We know that an element $a \in \mathbb{Z}/n\mathbb{Z}$ is invertible if and only if $\gcd(a, n) = 1$.

Definition 13.1.1. We define the number of $a \in \mathbb{Z}/n\mathbb{Z}$ such that $\gcd(a, n) = 1$ as $\varphi(n)$.

Remark. We trivially find that $\#(\mathbb{Z}/n\mathbb{Z}^*) = \varphi(n)$.

This $\varphi(n)$ is called the *Euler-phi* or *Euler's totient function.*

We now list some important results and examples about the Euler-phi.

Lemma 13.1.2. *Let $n > 1$ be an integer, then $\varphi(n) = n - 1$ iff n is a prime number.*

Proof. We have by Example 10.1.5 that for a prime number p all integers $1 \leq a \leq p - 1$ are invertible in $\mathbb{Z}/p\mathbb{Z}$. Hence indeed, if n is prime, then $\varphi(n) = n - 1$.

Conversely, suppose that $\varphi(n) = n - 1$. That means that for all $1 \leq a \leq n - 1$ we find that $\gcd(a, n) = 1$. If n were composite, then $n = xy$ for some $x, y \in \mathbb{Z}$ with $1 < x, y < n$. But then $x \mid n$ and $\gcd(x, n) = x \neq 1$, a contradiction to $\gcd(a, n) = 1$. Hence n is not composite, hence n is prime. □

Proposition 13.1.3. *Let p be a prime number and k a positive integer. Then*

$$\varphi(p^k) = p^k - p^{k-1}.$$

Proof. The elements in $\mathbb{Z}/p^k\mathbb{Z}$ that are not invertible are exactly those that are divisible by p. There are of course p^{k-1} such numbers, as there are p^k/p subsets of length p starting from zero. Each of those contains one element that is divisible by p. Hence the number of invertible elements $\varphi(p^k)$ is equal to $p^k - p^{k-1}$. □

For example, we have $\varphi(9) = 9 - 3 = 6$, $\varphi(5) = 4$, $\varphi(625) = 625 - 125 = 500$.

Theorem 13.1.4. (Euler's Theorem) *Let n be a positive integer. Then for all a with $\gcd(a, n) = 1$ we have*

$$a^{\varphi(n)} \equiv 1 \pmod{n}.$$

Proof. The number of elements in $\mathbb{Z}/n\mathbb{Z}^*$ is equal to $\varphi(n)$. By Lagrange's Theorem then the order of a is a divisor of $\varphi(n)$. Write $\varphi(n) = k\,\mathrm{ord}(a)$. We have:

$$a^{\varphi(n)} = a^{k\,\mathrm{ord}(a)} = (a^{\mathrm{ord}(a)})^k \equiv 1^k \equiv 1 \pmod{n}.$$

□

Corollary 13.1.5. (Fermat's Little Theorem) *Let p be a prime number, then*

$$a^{p-1} \equiv 1 \pmod{p}$$

for all $a \neq 0$. In particular,

$$a^p \equiv a \pmod{p}$$

for all a.

We use this trick to compute modular exponentiation easily. Say we would want to compute 2^{2020} modulo 37.

We do a division with remainder: $2020 = 56 \cdot 36 + 4$. So we have

$$2^{2020} = 2^{56\cdot 36+4} = 2^{56\cdot 36} \cdot 2^4 = (2^{36})^{56} \cdot 2^4 \equiv 1^{56} \cdot 2^4 = 1 \cdot 16 = 16 \pmod{37}.$$

When we have a composite number instead, it is a little bit more difficult, e.g. computing 2^{2020} modulo $38 (= 2 \cdot 19)$ cannot be done right now, as we don't know $\varphi(38)$ (Actually, we can write down the multiplicative table or check gcd's and find that $\varphi(38) = 18$, but this is cumbersome.) Fortunately, we have a way of addressing this problem.

13.1.1 Chinese Remainder Theorem

Computing the Euler-phi for composite numbers rests upon the statement

$$\varphi(mn) = \varphi(m)\varphi(n)$$

provided that $\gcd(m, n) = 1$. This result is a corollary to what is known as the Chinese Remainder Theorem.

Theorem 13.1.6. (Chinese Remainder Theorem) *Let m, n be positive coprime integers. Then for each pair (a, b) of integers, there exists an integer c with*

$$c \equiv a \pmod{m}$$

$$c \equiv b \pmod{n}$$

If d is an integer such that

$$d \equiv a \pmod{m}$$

$$d \equiv b \pmod{n}$$

then $d \equiv c \pmod{mn}$.

Proof. Since $\gcd(m, n) = 1$, we can find integers x, y that satisfy Bézout's Identity:

$$xm + yn = 1.$$

Let $c = ayn + bxm$. Then clearly $c \equiv ayn \pmod{m}$. But also $yn \equiv 1 \pmod{m}$. Hence $c \equiv a \pmod{m}$. Similarly, $c \equiv b \pmod{n}$.

Now suppose that d is an integer with $d \equiv a \pmod{m}$ and $d \equiv b \pmod{n}$. Then $d \equiv c \pmod{m}$ and $d \equiv c \pmod{n}$. Hence $m \mid d - c$ and $n \mid d - c$. But then $mn = \mathrm{lcm}(m, n) \mid d - c$. (See Corollary 10.2.5.) □

The most practical formulation of the Chinese Remainder Theorem is as follows. Let $n = pq$ with p, q coprime numbers. Consider the integer m with $m \equiv a \pmod{p}$ and $m \equiv b \pmod{p}$. Then we can determine m modulo n, without knowing what m is exactly (!), by using the following formula:

$$m \equiv a + p(b - a)u \pmod{n}.$$

Here $u \equiv p^{-1} \pmod{q}$. If we calculate modulo p, then $m \equiv a \pmod{p}$, since $p(b-a)u \equiv 0 \pmod{p}$, while modulo q, we have $m \equiv a + pu(b-a) \equiv a + b - a \equiv b \pmod{q}$ since $u \equiv p^{-1} \pmod{q}$.

Example 13.1.7. We want to know given $m \equiv 3 \pmod{7}$ and $m \equiv 5 \pmod{11}$, what is $m \pmod{77}$? We pre-calculate $7^{-1} \equiv 8 \pmod{11}$. Then we find $m \equiv 3 + 7(5-3)8 \equiv 3 + 7 \cdot 2 \cdot 8 \equiv 3 + 14 \cdot 8 \equiv 3 + 112 = 115 \equiv 38 \pmod{77}$.

Another way to calculate this, is to say:

$$m = 3 + 7k$$

$$m = 5 + 11l$$

Hence we need to find k, l such that $3 + 7k = 5 + 11l$, or $7k = 11l + 2$. Trying small l gives a solution: $2, 13, 24, 35$ and 35 is divisible by 7. So $l = 3$, which makes $k = 5$ and indeed $m = 38$. ♠

The Chinese remainder theorem has many direct corollaries:

Corollary 13.1.8. *When* $\gcd(m, n) = 1$ *we have*

$$\mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z} \cong \mathbb{Z}/mn\mathbb{Z}.$$

Proof. To provide a proof of this theorem, we need to construct an isomorphism. Let $\mathrm{f}\colon \mathbb{Z}/mn\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z},\ a \pmod{mn} \mapsto (a \pmod{m}, a \pmod{n})$ be this desired isomorphism.

Surjectivity of this map is the first statement of the Chinese Remainder Theorem (i.e., the existence of the integer c) and injectivity the second statement of the Chinese Remainder Theorem.

Next we just need to show that $\mathrm{f}(0) = (0, 0)$ and that $\mathrm{f}(a + b) = \mathrm{f}(a) + \mathrm{f}(b)$. We start with the former. Being 0 in $\mathbb{Z}/mn\mathbb{Z}$ means a multiple of mn. But this is then also a multiple of m and a multiple of n. So indeed $\mathrm{f}(0) = (0, 0)$.

Let $a, b \in \mathbb{Z}/mn\mathbb{Z}$ be arbitrary. Then $\mathrm{f}(a + b) = (a + b \pmod{m}, a + b \pmod{n})$, by definition of modular arithmetic, this is equal to $(a \pmod{m} + b \pmod{m}, a \pmod{n} + b \pmod{n})$, then using Exercise 12.P, we find that this is equal to $(a \pmod{m}, a \pmod{n}) + (b \pmod{m}, b \pmod{n}) = \mathrm{f}(a) + \mathrm{f}(b)$.

So indeed, this f is an isomorphism. □

Example 13.1.9. Let us work out this isomorphism in detail. Consider $\mathbb{Z}/14\mathbb{Z}$. Then by the previous, we know that $\mathbb{Z}/14\mathbb{Z} \cong \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/7\mathbb{Z}$. The isomorphism is $m \mapsto (m \pmod{2}, m \pmod{7})$. So we get:

$$\begin{gathered}
0 \mapsto (0,0)7 \mapsto (1,0) \\
1 \mapsto (1,1)8 \mapsto (0,1) \\
2 \mapsto (0,2)9 \mapsto (1,2) \\
3 \mapsto (1,3)10 \mapsto (0,3) \\
4 \mapsto (0,4)11 \mapsto (1,4) \\
5 \mapsto (1,5)12 \mapsto (0,5) \\
6 \mapsto (0,6)13 \mapsto (1,6)
\end{gathered}$$

Addition of $6 + 9 \pmod{14}$ is then the same as adding $(0, 6)$ and $(1, 2)$, yielding $(1, 1)$ which corresponds to 1.

The reason for the Chinese Remainder Theorem is of course not in addition. One of the main reasons has already been discussed before, namely to solve systems of modular equations. However, we have more than an isomorphism, we also have

$$\mathrm{f}(ab) = \mathrm{f}(a)\,\mathrm{f}(b),$$

and

$$\mathrm{f}(1) = (1, 1).$$

That means that in order to compute a product modulo a large number, we can compute the corresponding products modulo factors.

Example 13.1.10. Consider $6 \cdot 9 \pmod{14}$. Then that coincides with the multiplication of $(0, 6)$ and $(1, 2)$, yielding $(0, 5)$ which corresponds to 12. Indeed $6 \cdot 9 = 3 \cdot 2 \cdot 9 \equiv 3 \cdot 4 \equiv 12 \pmod{14}$.

In particular, from the rules for multiplication, it follows that

$$\mathrm{f}(a^{-1}) = \mathrm{f}(a)^{-1}.$$

Namely, $\mathrm{f}(a)\,\mathrm{f}(a^{-1}) = \mathrm{f}(aa^{-1}) = \mathrm{f}(1) = 1$. So $\mathrm{f}(a^{-1})$ is the inverse to $\mathrm{f}(a)$, since the multiplication modulo n is commutative. From this we find that $a \in \mathbb{Z}/mn\mathbb{Z}$ is invertible iff a is invertible in both $\mathbb{Z}/m\mathbb{Z}$ and $\mathbb{Z}/n\mathbb{Z}$. We have the following corollary:

Corollary 13.1.11. *When* $\gcd(m, n) = 1$ *we have*

$$\mathbb{Z}/m\mathbb{Z}^* \times \mathbb{Z}/n\mathbb{Z}^* \cong \mathbb{Z}/mn\mathbb{Z}^*.$$

This has the following immediate corollary:

Corollary 13.1.12. *When* $\gcd(m, n) = 1$ *we have*

$$\varphi(m)\varphi(n) = \varphi(mn).$$

Using this last corollary, we can easily compute the Euler-phi when we know that factorisation of our integer.

Consider $1220312709 = 3^5 \cdot 7^3 \cdot 11^4$. Then $\varphi(1220312709) = (3^5 - 3^4)(7^3 - 7^2)(11^4 - 11^3) = 633928680$.

Or an easier example, $\varphi(63) = \varphi(7)\varphi(9) = 36$.

13.1.2 More on the Chinese Remainder Theorem

This subsection is devoted to suggesting some different approaches to apply the Chinese Remainder Theorem. Of each approach we give an example.

CRT for systems of two equations The system of two equations will have the form:

$$\begin{cases} x \equiv a & \pmod{m}; \\ x \equiv b & \pmod{n}, \end{cases}$$

with $\gcd(m, n) = 1$.

The first method is the one given earlier, and is repeated here to have them all together.

Approach 1. We precompute $u \equiv m^{-1} \pmod{n}$ using EEA. Then the solution $x \equiv a+m(b-a)u \pmod{mn}$. Checking that this result statisfies the original system of equations is easy: $a+m(b-a)u \equiv a \pmod{m}$ and $a+m(b-a)u \equiv a+(b-a) \equiv b \pmod{n}$.

Example 13.1.13. We want to solve the system of equations:

$$\begin{cases} x \equiv 7 & \pmod{29}; \\ x \equiv 11 & \pmod{72}. \end{cases}$$

So we precompute $29^{-1} \equiv 5 \pmod{72}$. Then $x \equiv 7+29(11-7)5 \pmod{mn}$. This is just $x \equiv 587 \pmod{2088}$. ♠

Approach 2. We compute $s,t \in \mathbb{Z}$ with the EEA such that $sm+tn=1$. Then we have $x \equiv bsm+atn \pmod{mn}$. Checking that this result satisfies the original system of equations is easy: $bsm+atn \pmod{m} \equiv atn \pmod{m}$, while by construction $t \equiv n^{-1} \pmod{m}$.

Example 13.1.14. We want to solve the system of equations:

$$\begin{cases} x \equiv 5 & \pmod{27}; \\ x \equiv 7 & \pmod{65}. \end{cases}$$

We use EEA to find s and t: $s=-12$ and $t=5$. Then let $x \equiv 7 \cdot -12 \cdot 27 + 5 \cdot 5 \cdot 65 \pmod{mn}$. This is $-643 \equiv 1112 \pmod{1755}$. ♠

CRT for systems of N equations To solve a system of N modular equations, you can subdivide the system of N equations into many systems of 2 equations and apply one of the previous techniques, or just do it all at once as shown below.

So suppose that we want to solve a system of N equations like

$$\begin{cases} x \equiv a_0 & \pmod{n_0}; \\ x \equiv a_1 & \pmod{n_1}; \\ \vdots & \\ x \equiv a_{N-1} & \pmod{n_{N-1}}. \end{cases}$$

Approach 3. We first determine u_i such that $u_i = \delta_{ij} \pmod{n_j}$, i.e.,

$$\begin{cases} u_i \equiv 1 & \pmod{n_i}; \\ u_i \equiv 0 & \pmod{n_j}, \end{cases}$$

using the EEA. We do this by computing $s_i, t_i \in \mathbb{Z}$ with

$$s_i n_i + t_i m_i = 1,$$

where $m_i = \prod_{i \neq j} n_j$. We set $u_i := t_i m_i$.

Then the desired solution is $x = u_0 a_0 + u_1 a_1 + \ldots + u_{N-1} a_{N_1} \pmod{n_0 n_1 \cdots n_{N-1}}$.

Example 13.1.15. We show this for $N=3$:

$$\begin{cases} x \equiv 3 & \pmod{27}; \\ x \equiv 5 & \pmod{29}; \\ x \equiv 7 & \pmod{65}. \end{cases}$$

We compute $m_0 = 29 \cdot 65 = 1885$, $m_1 = 27 \cdot 65 = 1755$ and $m_2 = 27 \cdot 29 = 783$.

Then we apply the EEA to find $u_0 = 30160$, $u_1 = 3510$ and $u_2 = 17226$.

The endresult then $x \equiv 30160 \cdot 3 + 3510 \cdot 5 + 17226 \cdot 7 \equiv 228612 \equiv 25032 \pmod{50895}$. ♠

13.1.3 Calculating $\varphi(n)$ is hard

We can easily see that computing $\varphi(n)$ is easiest when we know the factorisation of n. Since factorisation is very hard, this may not be the most efficient way of computing $\varphi(n)$. In particular, computing $\varphi(n)$ is equally hard as factorisation. We show this for a particular class of composite numbers of the form $n = pq$ with p, q distinct primes. For the general case we refer to .

[[“Riemann’s hypothesis and tests for primality” by Gary L. Miller]]

[[“A Computational Introduction to Number Theory and Algebra” by Victor Shoup]]

Suppose that $n = pq$, but we don’t know p or q. We want to show that knowing $\varphi(n)$ gives us enough information to solve p (or q).

By the Chinese Remainder Theorem, we know that $\varphi(n) = (p-1)(q-1) = pq + 1 - p - q = n + 1 - p - q$. So with knowledge of n and $\varphi(n)$, we also know that value of $p + q = n + 1 - \varphi(n)$. Then using

$$(X-p)(X-q) = X^2 - (p+q)X + pq$$

we find that the quadratic formula $x^2 - (p+q)x + n$ can be solved for x and a solution is either p or q.

Example 13.1.16. Let $n = 323$ and $\varphi(n) = 288$. We want to know the factorisation of n. (We somehow know that n is the product of two distinct primes.) Write $n = pq$, then $p + q = 323 + 1 - 288 = 36$. So we need to solve the quadratic equation:

$$x^2 - 36x + 323 = 0.$$

The famous *abc*-formula gives us:

$$x = \frac{36 \pm \sqrt{36^2 - 4 \cdot 323}}{2} = 18 \pm \sqrt{18^2 - 323} = 18 \pm \sqrt{324 - 323} = 18 \pm 1.$$

Indeed, $n = 323 = 17 \cdot 19$. ♠

13.2 Exponentiation in $\mathbb{Z}/n\mathbb{Z}$

In this section we discuss exponentiation modulo an integer. There are (at least) three different ways of computing this, and each has its benefits. Of course there is always the way to compute the exponentiation with ordinary integers and then do a division with remainder, but this is a terrible idea:

We want to compute 3^{2019} modulo 73. We first compute 3^{2019} :

$$2.0314834051542953369733134720530847092601651327810912 \times 10^{963}$$

which is a way too large number. So we at least need to calculate modulo 73 in between, so let us calculate $3^{2019} = 3^{19} \cdot 3^{2000} = 3^{19} \cdot (3^{20})^{10}$.

First:

$$3^{19} = 1162261467 \equiv 70 \pmod{73},$$
$$3^{20} \equiv 70 \cdot 3 \equiv 64 \pmod{73},$$
$$64^{10} = 1152921504606846976 \equiv 64 \pmod{73}.$$

or instead

$$9^{10} = 3486784401 \equiv 64 \pmod{73},$$

(Why can we instead compute this?) Then the final answer can be found in multiplying 70 by 64, yielding, 27.

Note that the numbers get quite large, and we have to do quite some calculations.

13.2.1 Eulers theorem and Little Fermat

We repeat here a faster method that we discussed earlier as an application of Fermat's Little Theorem. This says $a^{p-1} \equiv 1 \pmod{p}$. So the preceding example, of calculating 3^{2019} modulo 73 becomes something like this:

$$3^{2019} = 3^{28\cdot 72+3} = 3^3 \cdot (3^{72})^{28} \equiv 3^3 \cdot (1)^{28} \equiv 27 \pmod{73},$$

a substantially easier method. But how does this work is we work modulo a composite number?

Remember Euler's Theorem (Theorem 13.1.4), that gives a way of computing exponentials when the *base* is coprime to the *modulus*. In the expression

$$a^e \pmod{n}$$

we call a the base, n the modulus and e the exponent.

The preceding example can be modified and we want to compute $2^{2019} \pmod{75}$. Therefore, we need to know $\varphi(75) = \varphi(5^2)\varphi(3) = (5^2 - 5) \cdot 2 = 40$. Then we have

$$2^{2019} = 2^{19+50\cdot 40} \equiv 2^{19} \cdot (2^{40})^{50} \equiv 2^{19} \pmod{75},$$

Then computing 2^{19} is still quite difficult, but we can split 19 up into $19 = 2 \cdot 8 + 3$, so we need to compute $(2^8)^2 \cdot 2^3$ modulo 75. $2^8 = 256 \equiv 31 \pmod{75}$. Then $31^2 \cdot 2^3 = 61 \cdot 8 \equiv 38 \pmod{75}$.

Moreover, if we would have a base that is not coprime to the modulus, for example computing $3^x \pmod{75}$ would be a different problem.

Therefore, we are in need of different ways of exponentiating.

13.2.2 Using the Chinese Remainder Theorem

Since we have the Chinese Remainder Theorem, if we can factor our modulus a little bit, we can do calculations in small moduli and then use the Chinese Remainder Theorem to compute the desired result. Let us compute 2^{2019} again modulo 75. We know that $75 = 3 \cdot 5^2$. So we calculate 2^{2019} modulo 3 and modulo 25.

Modulo 3 is easy, as by Fermat's Little theorem, $2^{2k} \equiv 1 \pmod{3}$ and $2^{2k+1} \equiv 2 \pmod{3}$. So $2^{2019} \equiv 2 \pmod{3}$.

Modulo 25 is little harder, we know that $\varphi(25) = 20$, so we need to calculate 2^{19} modulo 25. We have $2^{19} = 2^4 \cdot (2^5)^3 \equiv 16 \cdot 7^3 \equiv 16 \cdot 18 = 288 \equiv 13$.

Then to apply the Chinese Remainder Theorem, we want to compute the inverse of 3 modulo 25, which is 17, as $3 \cdot 17 = 51$. Then we get:

$$2 + 3(13-2)17 = 2 + 33 \cdot 17 = 563 \equiv 38 \pmod{75}.$$

Note that we used the formula from the practical formulation here.

13.2.3 Square and Multiply

An entirely different way of exponentiation is using Square and Multiply, for that we assume familiarity with the binary expansion of decimal numbers, for example:

We have $65 = 1000001_2$, $18 = 10010_2$ and $4 = 100_2$. Like in the decimal expansion, we notice that $(a_n a_{n-1} \cdots a_1 a_0)_2$ represents

$$a_n \cdot 2^n + a_{n-1} 2^{n-1} + \ldots + a_1 \cdot 2 + a_0,$$

where of course each $a_i \in \{0,1\}$.

In order to make explicit the example, we first compute $3^{45} \pmod{23}$ using the old method, namely, $3^{45} = 3^{44+1} = 3 \cdot (3^{22})^2 \equiv 3 \pmod{23}$.

Example 13.2.1. We use the square and multiply method, for that note $45 = 101101_2$. So we can write $3^{45} = 3^{101101_2}$. Working from left-to-right (starting from the most significant bit) and starting with 1, we proceed as follows:

$$\begin{cases} (-)^2 & \text{if the bit is 0;} \\ (-)^2 \cdot 3 & \text{if the bit is 1.} \end{cases}$$

So here:

$$\begin{aligned}
1 &\to 1^2 \cdot 3 \equiv 3 \pmod{23} && \text{(square and multiply)} \\
3 &\to 3^2 \equiv 9 \pmod{23} && \text{(square)} \\
9 &\to 9^2 \cdot 3 \equiv 81 \cdot 3 \equiv 13 \pmod{23} && \text{(square and multiply)} \\
13 &\to 13^2 \cdot 3 \equiv 169 \cdot 3 \equiv 1 \pmod{23} && \text{(square and multiply)} \\
1 &\to 1^2 \equiv 1 \pmod{23} && \text{(square)} \\
1 &\to 1^2 \cdot 3 \equiv 3 \pmod{23} && \text{(square and multiply)}
\end{aligned}$$

♠

By reducing modulo 23 in each step, this is a quite fast way of calculating exponents. It is easy to see that in the above example it will not hurt to start with the least significant bit instead of the most significant bit. This is in fact not generally true. As an example, we compute 3^{24} modulo 31 in both ways.

Example 13.2.2. We have $24 = 11000_2$. So, with *right-to-left* square and multiply we find

$$\begin{aligned}
1 &\to 1^2 \equiv 1 \pmod{31} \\
1 &\to 1^2 \equiv 1 \pmod{31} \\
1 &\to 1^2 \equiv 1 \pmod{31} \\
1 &\to 1^2 \cdot 3 \equiv 3 \pmod{31} \\
3 &\to 3^2 \cdot 3 \equiv 27 \pmod{31}
\end{aligned}$$

while with *left-to-right* square and multiply we find

$$\begin{aligned}
1 &\to 1^2 \cdot 3 \equiv 3 \pmod{31} \\
3 &\to 3^2 \cdot 3 \equiv 27 \pmod{31} \\
27 &\to 27^2 \equiv (-4)^2 \equiv 16 \pmod{31} \\
16 &\to 16^2 = 256 \equiv 8 \pmod{31} \\
8 &\to 8^2 = 64 \equiv 2 \pmod{31}
\end{aligned}$$

Here of course only the latter is correct. So always work from left-to-right with square and multiply. ♠

Remark. There exists a square and multiply that works from right-to-left, but it is different and inclusion here would only over-complexify things.

Square and multiply is computationally efficient. The number of multiplications and number of squarings are both small. Included is a pseudo-code for square and multiply.

```
Data: Integers $a, d, n$
Result: $x$ with $x \equiv a^d \pmod{n}$
1. Write $d = (d_{k-1}d_{k-2}\cdots d_1 d_0)_2$;
2. $x \leftarrow 1$;
3. for $i = k-1$ to $0$ do
    $x \leftarrow x^2 \bmod n$;
    if $d_i = 1$ then
        $x \leftarrow ax \bmod n$;
    end
end
4. return $x$.
```

Algorithm 4: Exponentiation modulo n

13.2.4 Finding inverse, using exponentiation

A few chapters earlier, in order to compute the inverse to some $a \in \mathbb{Z}/n\mathbb{Z}$, we used the Extended Euclidean Algorithm. With the new methods of this chapter, we know:

Corollary 13.2.3. *Let $a \in \mathbb{Z}/n\mathbb{Z}$ be invertible, (i.e.* $\gcd(a,n) = 1$*) then its inverse is given by* $a^{\varphi(n)-1}$.

Proof. We have $a \cdot a^{\varphi(n)-1} = a^{\varphi(n)} \equiv 1 \pmod{n}$ by Euler's Theorem. □

So another way of determining the inverse is to compute this exponentiation. We give an example.

Example 13.2.4. Let $n = 729 = 3^6$. Then the inverse to $a = 5$ modulo n is equal to $5^{\varphi(n)-1} = 5^{485}$. We write $485 = 111100101_2$ and compute using square and multiply:

$$\begin{aligned}
1 &\to 1^2 \cdot 5 \equiv 5 \pmod{729}\\
5 &\to 5^2 \cdot 5 \equiv 125 \pmod{729}\\
125 &\to 125^2 \cdot 5 \equiv 122 \pmod{729}\\
122 &\to 122^2 \cdot 5 \equiv 62 \pmod{729}\\
62 &\to 62^2 \equiv 199 \pmod{729}\\
199 &\to 199^2 \equiv 235 \pmod{729}\\
235 &\to 235^2 \cdot 5 \equiv 563 \pmod{729}\\
563 &\to 563^2 \equiv 583 \pmod{729}\\
583 &\to 583^2 \cdot 5 \equiv 146 \pmod{729}
\end{aligned}$$

Indeed $5 \cdot 146 = 730$. ♠

13.3 Exercises

13.A Compute $3^{65} \pmod{65}$ in three ways:

(a) Using the Chinese Remainder Theorem;

(b) Using Euler's Theorem;

(c) Using Square and Multiply.

13.B Compute the inverse to 17 modulo 35 in three ways:

(a) Using the Chinese Remainder Theorem;

(b) Using Euler's Theorem;

(c) Using the Extended Euclidean Algorithm.

13.C Compute $\varphi(252525)$. [Hint: first factor it, notice that 25 occurs three times in the decimal expansion.]

13.D Apply Corollary 13.1.8 to $\mathbb{Z}/252525\mathbb{Z}$.

13.E Compute 5^{2019} modulo 17.

13.F Let $p = 2n + 1$ be an odd prime number. Show that for any $a \neq 0$ we have

$$p \mid a^n + 1 \text{ or } p \mid a^n - 1.$$

[Hint: Use Fermat's Little Theorem.]

13.G Let $n = 257$. Calculate for each of the following numbers its multiplicative inverse modulo n:

$$37, 86, 103, 129, 201.$$

13.H Let $n = 257$. Calculate the inverse z of 68 modulo n. For each $x \in \{37, 86, 103, 129, 201\}$ calculate $z \cdot \prod_{y \neq x} y$. Compare with the preceding exercise.

13.I The previous two exercises were to demonstrate that more computations can be saved by computing multiple inverses at once. Show that the following procedure on $x_1, x_2, \ldots, x_n$ indeed yields all inverses $x_1^{-1}, x_2^{-1}, \ldots, x_n^{-1}$.

```
Data: Integers $x_1, \ldots, x_n$ modulo a prime $p$
Result: Integers $y_1, \ldots, y_n$ with $x_i \equiv y_i \pmod{p}$
1. $X \leftarrow x_1 \cdot x_2 \cdots x_n \pmod{p}$;
2. $Y \leftarrow X^{-1} \pmod{p}$;
3. for $i = 1$ to $n$ do
   |  $y_i \leftarrow Y \cdot \prod_{j \neq i} x_i \pmod{p}$;
end
return $y_1, \ldots, y_n$.
```

13.J Let $n = 198001$ and be given that $\varphi(n) = 197104$. Suppose that n is the product of two distinct primes p, q. What are p and q?

Chapter 14

Rings and Fields

When we have discussed group, the example $(\mathbb{Z}, +)$ came to mind. But we have seen a richer structure on $\mathbb{Z}$, it also admits a multiplication. A structure which admits both addition and multiplication is called a ring. We further talk about a special class of rings, which are called polynomial rings, especially polynomial rings over fields.

14.1 Rings and Fields - General framework

The most important example of a ring we have seen so far is the rings $\mathbb{Z}$. We have seen each of the following properties to hold:

1. For all $a, b \in \mathbb{Z}$, $a + b \in \mathbb{Z}$;
2. For all $a, b \in \mathbb{Z}$, $a + b = b + a$;
3. For all $a, b, c \in \mathbb{Z}$, $a + (b + c) = (a + b) + c$;
4. There exists some $0 \in \mathbb{Z}$ such that $a + 0 = a$;
5. For all $a \in \mathbb{Z}$ there exists some $(-a) \in \mathbb{Z}$ such that $a + (-a) = 0$;
6. For all $a, b \in \mathbb{Z}$, $ab \in \mathbb{Z}$;
7. For all $a, b \in \mathbb{Z}$, $ab = ba$;
8. For all $a, b, c \in \mathbb{Z}$, $a(bc) = (ab)c$;
9. There exists some $1 \in \mathbb{Z}$ such that $a1 = a$;
10. For all $a, b, c \in \mathbb{Z}$, $a(b + c) = ab + ac$.

A structure for which all these properties hold is called a *commutative ring.* We call the element 0 the *neutral element* and 1 the *identity.* Note that properties 1-5 together just state that $(\mathbb{Z}, +)$ is an abelian group.

Definition 14.1.1. Let R be a set together with binary operations $+, \cdot$ on R. Then $(R, +, \cdot)$ is a ring iff

1. $(R, +)$ is an abelian group;
2. For all $a, b, c \in R$ we have $a(bc) = (ab)c$;

3. There exists some $1 \in R$ such that $1a = a1 = a$;
4. For all $a, b, c \in R$ we have $a(b + c) = ab + ac$;
5. For all $a, b, c \in R$ we have $(a + b)c = ac + bc$.

In particular, when we also have $ab = ba$ for all $a, b \in R$, we call the ring *commutative*. The 5th property in this list then follows from the 4th.

As we have seen $(\mathbb{Z}, +, \cdot)$ is a commutative ring. There also exist non-commutative rings that are useful in mathematics. We give one as an example.

Example 14.1.2. Let R be any ring (e.g. $\mathbb{Z}$), then $\mathrm{M}_n(R)$, the set of all $n \times n$ matrices with coefficients in R, is also a ring. But this is not commutative when $n > 1$. See also the example of page 128. All ring-axioms can be checked as an exercise. The 0-matrix is of course the matrix with all entries zero, while the identity of $\mathrm{M}_n(R)$ is of course the identity matrix I_n. ♠

There are also finite rings, an example is $\mathbb{Z}/n\mathbb{Z}$ with its usual addition and multiplication. Take note that $\mathbb{Z}/n\mathbb{Z}^*$ is no longer a ring, as it does not contain a 0.

There are those rings, like $\mathbb{Z}/p\mathbb{Z}$, where each non-zero element has a multiplicative inverse. We have a special name for them.

Definition 14.1.3. Let $(R, +, \cdot)$ be a commutative ring such that for every non-zero $a \in R$ there exists some b such that $ab = 1$. Such a ring is called a *field*.

As described above, $\mathbb{Z}/p\mathbb{Z}$ is a field, while also the set of all rational numbers (=fractions) is a field.

Notation 14.1.4. For the field $\mathbb{Z}/p\mathbb{Z}$ we also have the notation $\mathbb{F}_p$, or $\mathrm{GF}(p)$. The latter notation stands for *Galois Field* after Évariste Galois, who laid the foundations for group theory and the field in mathematics we now call *Galois Theory*.

An element that has a multiplicative inverse in a ring is simply called invertible. We have the notation R^* for the set/group of all invertible elements of R. (See Exercise 14.B.)

Given two rings $(R_1, +_1, \cdot_1), (R_2, +_2, \cdot_2)$, we can define a third ring $R_1 \times R_2$ which consists of elements of the form (r_1, r_2) with $r_i \in R_i$ and both addition and multiplication defined coordinatewise:

$$(r_1, r_2) + (s_1, s_2) = (r_1 +_1 s_1, r_2 +_2 s_2)$$

and

$$(r_1, r_2) \cdot (s_1, s_2) = (r_1 \cdot_1 s_1, r_2 \cdot_2 s_2).$$

As an exercise (14.A) we leave the reader to check that this is indeed a ring. We call this the *direct product*. (See also Exercise 12.P.)

14.1.1 Subrings

Like with groups, we can ask ourselves the question which subsets of a ring are themselves rings. Indeed, the notion of subring arises.

Definition 14.1.5. Let R be a ring and $S \subset R$ a subset of R. If S is a ring, we call S a subring of R.

As with groups, we have so-called subring test:

Theorem 14.1.6. (Subring Test) *Let R be a ring and S a subset of R such that*

(S0) $(S, +)$ *is a subgroup of* $(R, +)$;

(S1) There exists a $1 \in S$;

(S2) For all $a, b \in S$ *we have* $ab \in S$.

Then S is a subring of R.

Proof. Using the numbers from Definition 14.1.1:

1. Free due to (S0);
2. Inherited from R;
3. Free from (S1);
4. Inherited from R;
5. Inherited from R.

□

Example 14.1.7. Consider the ring of integers $\mathbb{Z}$. The set $2\mathbb{Z}$ is not a subring, since it does not contain a 1. ♠

Example 14.1.8. Consider the ring $\mathbb{Z}/6\mathbb{Z}$. Then the ring $3\mathbb{Z}/6\mathbb{Z}$ is a subring of $\mathbb{Z}/6\mathbb{Z}$. We give the addition and multiplication tables of $3\mathbb{Z}/6\mathbb{Z}$ in the tables below. One can now easily verify

$+$	0	3
0	0	3
3	3	0

Table 14.1: Addition table of $3\mathbb{Z}/6\mathbb{Z}$.

$\cdot$	0	3
0	0	0
3	0	3

Table 14.2: Multiplication table of $3\mathbb{Z}/6\mathbb{Z}$.

each of the properties mentioned in the Subring Test. Clearly, S0 and S2 hold, while the identity of this ring is clearly 3. ♠

14.1.2 Ringisomorphisms

If we would replace all 3s in the tables at the end of the next section, we would get the addition and multiplication tables of $\mathbb{Z}/2\mathbb{Z}$. To make this more precise:

Definition 14.1.9. Let $(R_1, +_1, \cdot_1)$ and $(R_2, +_2, \cdot_2)$ be rings. Then we say that R_1 and R_2 are *isomorphic*, notation $R_1 \cong R_2$, if there exists a bijective map $\mathrm{f}\colon R_1 \to R_2$ with

- $\mathrm{f}(0_1) = 0_2$, $\mathrm{f}(1_1) = 1_2$;
- $\mathrm{f}(r_1 +_1 s_1) = \mathrm{f}(r_1) +_2 \mathrm{f}(s_1)$;
- $\mathrm{f}(r_1 \cdot_1 s_1) = \mathrm{f}(r_1) \cdot_2 \mathrm{f}(s_2)$.

Such a map is called an *isomorphism.*

Remark. We often just write $\mathrm{f}(ab) = \mathrm{f}(a)\,\mathrm{f}(b)$, $\mathrm{f}(0) = 0$ and such and let the reader figure out which operations are referred to.

Example 14.1.10. For every ring R, we have a trivial isomorphism $R \to \mathrm{M}_1(R)$, defined by $a \mapsto (a)$. ♠

Example 14.1.11. The isomorphism given in Corollary 13.1.8 is in fact an isomorphism of rings, so we still have

$$\mathbb{Z}/mn\mathbb{Z} \cong \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z},$$

but now as rings. ♠

From this isomorphism, using the following lemma, it easily follows that $\mathbb{Z}/mn\mathbb{Z}^*$ is isomorphic to $\mathbb{Z}/m\mathbb{Z}^* \times \mathbb{Z}/n\mathbb{Z}^*$ as groups.

Lemma 14.1.12. *Let R_1, R_2 be rings and $\mathrm{f}\colon R_1 \to R_2$ an isomorphism. Then f induces an isomorphism of R_1^* to R_2^*.*

Proof. We first show that when $a \in R_1$ is invertible, that $\mathrm{f}(a)$ is also invertible. For that, note that

$$1 = \mathrm{f}(1) = \mathrm{f}(aa^{-1}) = \mathrm{f}(a)\,\mathrm{f}(a^{-1})$$

hence $\mathrm{f}(a)^{-1} = \mathrm{f}(a^{-1})$, as required.

So indeed f induces a map from R_1^* to R_2^*. Since f already preserves multiplication and $\mathrm{f}(1) = 1$, we have immediately checked all the properties that a group isomorphism must have. So indeed $R_1^* \cong R_2^*$. □

14.2 *Polynomial Rings

Everyone has seen expressions like $x^2 + 3x + 1$ or $5x + 7$. The name objects like that have in common is polynomial. This words has origins in both Greek and Latin, where *poly* is Greek for many and *nomen* is Latin for name.

Definition 14.2.1. Let R be a ring and X an indeterminate (or variable), then we have the *polynomial ring over R in one indeterminate* (or one variable), written $R[X]$ which consists of the set of all polynomials:

$$a_0 + a_1X + a_2X^2 + \ldots + a_nX^n$$

with all $a_i \in R$. Addition and multiplication are given by the following rules:

$$\sum_{i=0}^{n} a_iX^i + \sum_{i=0}^{m} b_iX^i = \sum_{i=0}^{\max\{m,n\}} (a_i + b_i)X^i,$$

$$\sum_{i=0}^{n} a_iX^i \cdot \sum_{i=0}^{m} b_iX^i = \sum_{j=0}^{m+n} c_iX^i$$

with $c_i = \sum_{j=0}^{i} a_jb_{i-j}$.

These operations make $R[X]$ into a ring, with 0 as the zero polynomial and 1 as the identity.

Example 14.2.2. The polynomial expressions $X^2 + 3X + 1$ and $5X + 7$ are example of integer-valued polynomials. So $f(X) = X^2 + 3X + 1 \in \mathbb{Z}[X]$ and $g(X) = 5X + 7 \in \mathbb{Z}[X]$. We then have

$$f(X) + g(X) = X^2 + 8X + 8,$$

and

$$f(X)g(X) = (X^2+3X+1)(5X+7) = 5X^3+15X^2+5X+7X^2+21X+7 = 5X^3+22X^2+26X+7.$$

♠

We write $f(X)$ for elements in $\mathbb{Z}[X]$ to signify that the X is an indeterminate. Mostly, we will consider only polynomials over a field.

Definition 14.2.3. Let k be a field, then for a polynomial $f(X) \in k[X]$ we define the *degree of* f, written $\deg(f)$ as the largest n such that $f_n \neq 0$. Here we write:

$$f(X) = \sum_{i=0}^{n} f_i X^i.$$

For the zero-polynomial we have the exception of defining the degree as $-\infty$.

As we can see above $\deg f = 2$ and $\deg g = 1$. Also $\deg(f + g) = 2$, while $\deg(fg) = 3$. This generalizes:

Lemma 14.2.4. *Let k be a field and let $f, g \in k[X]$. Then*

i. $\deg(f + g) \leq \max\{\deg f, \deg g\}$;

ii. $\deg(fg) = \deg f + \deg g$.

Proof. These are in fact easy to prove, when just looking at the definitions of addition and multiplication. There are just some important remarks:

i. Let $\deg f = n$ and $\deg g = m$. Suppose $n \geq m$. If $n > m$, nothing happens to f_n when adding g, so $\deg(f + g) = n$.

   If $n = m$, however, and $f_n = -g_n$, then the degree of $f + g$ is in fact lower than $\deg f$ and $\deg g$. (Take for example $f + -f = 0$.)

ii. From the definition of multiplication of polynomials we see that the highest coefficient occurs with X^{n+m}. However, it might occur that this coefficient becomes zero. In a field, however, this can never happen. See Exercises 14.E and 14.F.

□

If one polynomial f is a multiple of a polynomial g, i.e. there exists some polynomial h with $f = hg$, then we can of course divide f by g. But this again is not possible with two arbitrary polynomials.

It is very important to take care to remember that X^{-1} does not exist in a polynomial ring!

14.2.1 Polynomial division with remainder

In this section and in the next, we stress that we consider only polynomials with coefficients in a field. The discussion here does not always generalize to polynomials with coefficients in arbitrary rings.

Proposition 14.2.5. *Let k be a field and $f(X), g(X)$ polynomials in $k[X]$. Then there exist unique $q(X), r(X) \in k[X]$ with*

$$f(X) = q(X) \cdot g(X) + r(X)$$

with $\deg r(X) < \deg g(X)$.

Proof. We start by proving existence. Write $\deg f = n$ and $\deg g = m$. Then $g_m \neq 0$ and hence invertible, since we have coefficients in a field. We use an induction on n.

As the base for the induction we have the case $n < m$, when we can just choose $q = 0$ and $r = f$.

Now let $n \geq m$. Since g_m is invertible, let c be its inverse. Then the polynomial $f_n c X^{n-m} g(X)$ has degree n and has as leading coefficient[1] $f_n g_m c = f_n$. Hence the polynomial

$$f'(X) = f(X) - f_n c X^{n-m} g(X)$$

has a degree lower than n, as both terms X^n have cancelling coefficients.

We can apply the induction hypothesis and find $q'(X), r'(X) \in k[X]$ with

$$f'(X) = q'(X)g(X) + r'(X)$$

with $\deg r' < \deg g$. Then we have

$$f(X) = f'(X) + f_n c X^{n-m} g(X) = (f_n c X^{n-m} + q'(X))g(X) + r'(X).$$

Setting $q(X) = f_n c X^{n-m} + q'(X)$ and $r(X) = r'(X)$, we have

$$f(X) = q(X)g(X) + r(X)$$

with $\deg r < \deg g$ as desired.

Now we prove uniqueness. Suppose we can write

$$f(X) = q(X)g(X) + r(X)$$

and

$$f(X) = q'(X)g(X) + r'(X)$$

with $\deg r < \deg g$ and $\deg r' < \deg g$. Then we have

$$(q(X) - q'(X))g(X) + r(X) - r'(X) = 0$$

or equivalently

$$(q(X) - q'(X))g(X) = r'(X) - r(X).$$

But the right-hand side has degree less than $\deg g$ by Lemma 14.2.4. If $q(X) \neq q'(X)$, then the left-hand side would have degree greater than $\deg g$, a contradiction. (See again Lemma 14.2.4.) Hence $q(X) = q'(X)$. But then also $r(X) = r'(X)$. $\square$

When after division with remainder we find $r(X) = 0$, we say that $f(X)$ is *divisible by* $g(X)$ or that $g(X)$ is a *divisor* of $f(X)$. In practice, we perform this division with remainder with longdivison. The method is similar to the one for integers.

Example 14.2.6. Let $f(X) = X^4 - X^3 - 2X^2 + 3X - 4$ and $g(X) = X^2 - 1$. Then we have

$$
\begin{array}{r|rrrrr}
 & & & X^2 & -X & -1 \\ \cline{2-6}
X^2-1 & X^4 & -X^3 & -2X^2 & +3X & -4 \\
 & -X^4 & & +X^2 & & \\ \cline{2-4}
 & & -X^3 & -X^2 & +3X & \\
 & & X^3 & & -X & \\ \cline{3-5}
 & & & -X^2 & +2X & -4 \\
 & & & X^2 & & -1 \\ \cline{4-6}
 & & & & 2X & -5
\end{array}
$$

So indeed $f(X) = q(X)g(X) + r(X)$ with $q(X) = X^2 - X - 1$ and $r(X) = 2X - 5$. ♠

[1] The coefficient belonging to X^n with n largest.

Remark. Notice that in the above example we get a solution even though the polynomials have integer coefficients. This is not always the case, see the next example.

Example 14.2.7. Let $f(X) = X^4 - X^3 + X - 7$ and $g(X) = 2X^2 + 6$. Then we have

$$
\begin{array}{rlllll}
 & & & \tfrac{1}{2}X^2 & -\tfrac{1}{2}X & -\tfrac{3}{2} \\ \cline{2-6}
2X^2+6 \big) & X^4 & -X^3 & & +X & -7 \\
 & -X^4 & & -3X^2 & & \\ \cline{2-4}
 & & -X^3 & -3X^2 & +X & \\
 & & X^3 & & +3X & \\ \cline{3-5}
 & & & -3X^2 & +4X & -7 \\
 & & & 3X^2 & & +9 \\ \cline{4-6}
 & & & & 4X & +2
\end{array}
$$

We see that $q(X) = \frac{1}{2}X^2 - \frac{1}{2}X - \frac{3}{2}$ and $r(X) = 4X + 2$. In particular, the quotient is not an integer-valued polynomial. ♠

Corollary 14.2.8. *Let k be a field and $f(X) \in k[X]$ be a polynomial. Let $\alpha \in k$ be arbitrary. Then*

$$f(\alpha) = 0 \iff f(X) = (X - \alpha)g(X)$$

for some $g(X) \in k[X]$.

Proof. Suppose that $f(X) = (X - \alpha)g(X)$, then we immediately have

$$f(\alpha) = (\alpha - \alpha)g(\alpha) = 0.$$

Conversely, suppose that $f(\alpha) = 0$. We do division with remainder of $f(X)$ by $(X - \alpha)$. So there exist $q(X), r(X) \in k[X]$ with

$$f(X) = (X - \alpha)q(X) + r(X)$$

with $\deg r(X) < 1$. Hence $r(X)$ is a constant. Now $f(\alpha) = 0$, hence $(\alpha - \alpha)q(\alpha) + r(\alpha) = r(\alpha) = 0$. Since r is constant, we find that $r = 0$. Hence

$$f(X) = (X - \alpha)q(X).$$

□

The preceding corollary is a nice example of the use of the following credo:

"For a polynomial (in-)equality, whichever number you substitute for X, the (in-)equality still holds."

Example 14.2.9. We want to find all solutions to $x^3 - 93x^2 - 4681x + 4773 = 0$ given that 1 is a solution. We apply a longdivision:

$$
\begin{array}{rllll}
 & & X^2 & -92X & -4773 \\ \cline{2-5}
X-1 \big) & X^3 & -93X^2 & -4681X & +4773 \\
 & -X^3 & +X^2 & & \\ \cline{2-3}
 & & -92X^2 & -4681X & \\
 & & 92X^2 & -92X & \\ \cline{3-4}
 & & & -4773X & +4773 \\
 & & & 4773X & -4773 \\ \cline{4-5}
 & & & & 0
\end{array}
$$

We see that $X^3 - 93X^2 - 4681X + 4773 = (X-1)(X^2 - 92X - 4773)$. To the latter we can apply the *abc*-formula:

$$x = \frac{92 \pm \sqrt{92^2 + 4 \cdot 4773}}{2} = 46 \pm \sqrt{46^2 - 4773} = 46 \pm \sqrt{2116 + 4773} = 46 \pm \sqrt{6889} = 46 \pm 83.$$

Hence we find that the solutions are $x = 1, x = -37, x = 129$. ♠

An immediate corollary to the preceding is the following:

Corollary 14.2.10. *Let k be a field an $f(X) \in k[X]$ be a polynomial. Then the number of roots of f is bounded by the degree of f.*

Proof. We prove this by induction to the degree of f.

Suppose that f is of degree 0, hence a nonzero constant. Then of course f has no roots.

This show the basis for the induction.

If f has no roots, then of course $0 < \deg f$. If f has a root α, then we get

$$f(X) = (X - \alpha)g(X)$$

by Corollary 14.2.8. By Lemma 14.2.4 then, we find that $\deg g = \deg f - 1$. So using the induction hypothesis, we know that g has at most $\deg f - 1$ roots. And thus f has at most $\deg f - 1 + 1 = \deg f$ roots. □

14.2.2 Extended Euclidean Algorithm for polynomials

With this division with remainder, we also have a polynomial greatest common divisor. It is defined very similarly to the greatest common divisor for integers.

Definition 14.2.11. Let k be a field and let $f(X), g(X)$ be two polynomials. Their greatest common divisor $d(X)$ is a monic polynomial that divides both $f(X)$ and $g(X)$ of highest possible degree.

A polynomial is called *monic* if its leading coefficient is 1. For example, $X^2 + 1$ is monic, but $2X + 3$ is not.

If we consider polynomials with coefficients from a field, then we still have the Euclidean algorithm. It works almost entirely the same as with integers, hence we only discuss the Extended form.

Now, the algorithm in pseudocode:

```
Data: Polynomials $f(X), g(X) \in k[X]$
Result: Polynomials $d(X), a(X), b(X)$ with $d = a(X)f(X) + b(X)g(X)$
1. $F \leftarrow [f, 1, 0]$;
2. $G \leftarrow [g, 0, 1]$;
3. while $G[1] \neq 0$ do
    (a) $H \leftarrow F - (F[1]\ div\ G[1])G$;
    (b) $F \leftarrow G$;
    (c) $G \leftarrow H$;
end
4. $l \leftarrow LC(F[1])$;
5. $F \leftarrow F/l$;
6. $d \leftarrow F[1]$, $a \leftarrow F[2]$, $b \leftarrow F[3]$;
return $d, a, b$.
```

Algorithm 5: Polynomial Extended Euclidean Algorithm

Here LC stands for leading coefficient, omitting steps 4 and 5 yields the Extended Euclidean Algorithm for integers.

Example 14.2.12. We want to compute for each positive integer n, the greatest common divisor of n^2+1 and n^3-1. We can do this for each n separately, but we can also first compute $\gcd(X^2+1, X^3-1)$ and then substitute each value for n separately.

We use the steps from the algorithm above.

We set $F = [X^3-1, 1, 0]$ and $G = [X^2+1, 0, 1]$.

Now $H = F - (F[1]\ div\ G[1])G = [X^3-1,1,0] - X[X^2+1,0,1] = [X^3-1,1,0] - [X^3+X,0,X] = [-X-1,1,-X]$.

Then $F = [X^2+1,0,1]$ and $G = [-X-1,1,-X]$.

Again $H = [X^2+1,0,1] - (-X+1)[-X-1,1,-X] = [X^2+1,0,1] - [X^2-1,-X+1,X^2-X] = [2, X-1, -X^2+X+1]$.

Hence $F = [-X-1,1,X]$ and $G = [2, X-1, -X^2+X+1]$.

Then $H = [-X-1,1,X] - (-\frac{1}{2}X - \frac{1}{2})[2, X-1, -X^2+X+1] = [0, \frac{1}{2}(X^2+1), \frac{1}{2}(-X^3+4X+1)]$.

Now $F = [2, X-1, -X^2+X+1]$ and $G = [0, \frac{1}{2}(X^2+1), \frac{1}{2}(-X^3+4X+1)]$.

We have $G[1] = 0$. So $l = 2$ and we get $F = [1, \frac{1}{2}(X-1), \frac{1}{2}(-X^2+X+1)]$. So $d = 1, a(X) = \frac{1}{2}(X-1), b(X) = \frac{1}{2}(-X^2+X+1)$.

We check the outcome:

$$\begin{aligned}
a(X)f(X) + b(X)g(X) =& \frac{1}{2}(X-1)(X^3-1) + \frac{1}{2}(-X^2+X+1)(X^2+1) \\
=& \frac{1}{2}\left(X^4 - X - X^3 + 1 - X^4 - X^2 + X^3 + X + X^2 + 1\right) \\
=& \frac{1}{2}\,(2) \\
=& 1
\end{aligned}$$

So for every positive integer n we have $\gcd(n^2+1, n^3-1) = 1$. ♠

14.3 *Polynomial Modular Arithmetic

In this section we discuss a generalization of modular arithmetic from integers to polynomial rings.

14.3.1 Early examples

The easiest example is given by $\mathbb{Z}[X]/X\mathbb{Z}[X]$ (Observe the similarity with $\mathbb{Z}/2\mathbb{Z}$ for example.) which means just that every occurrence of X should be read as a 0. Hence this is just the ring of integers. For example, take $X^2+5X+17$. Write 0 for each $X : 0^2 + 5\cdot 0 + 17 = 17$.

Definition 14.3.1. Let R be a ring and $f(X) \in R[X]$ a polynomial with coefficients in R. Then we write $R[X]/f(X)R[X]$ for the set of polynomials over R modulo $f(X)$. We will also write $R[X]/(f(X))$.

We write $\overline{g(X)}$ for the polynomial $g(X)$ modulo $f(X)$. So we have $g(X) = \overline{g(X)} + h(X)f(X)$ for some $h(X) \in R[X]$.

On these polynomials we can define an addition and a multiplication:

$$\overline{g(X)} + \overline{h(X)} = \overline{g(X) + h(X)}$$

$$\overline{g(X)} \cdot \overline{h(X)} = \overline{g(X) \cdot h(X)}.$$

Since $R[X]/f(X)R[X]$ is given by

$$R[X]/f(X)R[X] := \{r_0 + r_1X + r_2X^2 + \ldots + r_{\deg f-1}X^{\deg f-1} \mid r_0, \ldots, r_{\deg f-1} \in R\},$$

we immediately find that when $\#R = m$ and $\deg f = n$, we have $\#R[X]/f(X)R[X] = m^n$.

Theorem 14.3.2. *Let R be a ring and $f(X) \in R[X]$ be a polynomial with coefficients in R. Then $R[X]/(f(X))$ is a ring.*

Proof. We shall not show all properties, as many are similar. We start with associativity of addition:

$$\begin{aligned}
(\overline{g_1(X)} + \overline{g_2(X)}) + \overline{g_3(X)} &= \overline{g_1(X) + g_2(X)} + \overline{g_3(X)} \\
&= \overline{(g_1(X) + g_2(X)) + g_3(X)} \\
&= \overline{g_1(X) + (g_2(X) + g_3(X))} \\
&= \overline{g_1(X)} + \overline{g_2(X) + g_3(X)} \\
&= \overline{g_1(X)} + (\overline{g_2(X)} + \overline{g_3(X)})
\end{aligned}$$

Commutativity of addition is similar, as are associativity of multiplication, commutativity of multiplication (when R is commutative) and distributivity of multiplication over addition.

The role of 0 is filled by $\overline{0}$, and the role of 1 is filled by $\overline{1}$. Their respective properties can be easily checked. □

Example 14.3.3. Consider $X^2 + X + 1$ and $X^3 + 1$ in $\mathbb{Z}[X]/(X^3 + X + 1)$. Then $X^3 + 1 = -X$. Then $X^2 + X + 1 + X^3 + 1 = X^2 + 1$, and $(X^2 + X + 1)(X^3 + 1) = -X^3 - X^2 - 1 = -X^2 + X$. ♠

Actually, mathematically speaking, we must first show that the addition and multiplication defined above are indeed well-defined, but we omit that due to readability.

Example 14.3.4. Let k be a field and consider $k[X]/(X^2)$. Sometimes this is called the ring of *dual numbers*. It is the set

$$\{a + b\overline{X} \mid a, b \in k\}.$$

Addition and multiplication are given by

$$a_1 + b_1\overline{X} + a_2 + b_2\overline{X} = (a_1 + a_2) + (b_1 + b_2)\overline{X},$$

$$(a_1 + b_1\overline{X})(a_2 + b_2\overline{X}) = a_1a_2 + (a_1b_2 + b_1a_2)\overline{X} + b_1b_2\overline{X}^2 = a_1a_2 + (a_1b_2 + b_1a_2)\overline{X}.$$

We can easily verify that from $(a + b\overline{X})(a^{-1} - ba^{-2}\overline{X}) = 1$ it follows that $a + b\overline{X}$ is invertible iff $a \neq 0$. ♠

Important in cryptography are bits, or elements of $\mathbb{F}_2$ ($= \mathbb{Z}/2\mathbb{Z}$.) That is why we study polynomials over $\mathbb{F}_2$ in greater detail.

$+$	0	1	X	$X+1$
0	0	1	X	$X+1$
1	1	0	$X+1$	X
X	X	$X+1$	0	1
$X+1$	$X+1$	X	1	0

Table 14.3: Addition table of $\mathbb{F}_4$.

$\cdot$	0	1	X	$X+1$
0	0	0	0	0
1	0	1	X	$X+1$
X	0	X	$X+1$	1
$X+1$	0	$X+1$	1	X

Table 14.4: Multiplication table of $\mathbb{F}_4$.

Example 14.3.5. Let $f(X) = X^2 + X + 1$ be a polynomial in $\mathbb{F}_2[X]$. We consider the ring $\mathbb{F}_2[X]/(X^2 + X + 1)$. Note that usually we could replace every occurrence of X^2 with $-X - 1$, but modulo 2, this is just equal to $X + 1$. So $(X + 1)^2 = X^2 + 2X + 1 = X + 1 + 1 = X$, for example. Note that this ring has exactly 4 elements, $0, 1, X, X + 1$. We list below the addition and multiplication tables. We see that every non-zero element is invertible. Hence we clearly have found a field! Since this field has four elements, we have denoted it by $\mathbb{F}_4$, but we may also write GF(4). Furthermore, there may exists more fields with four elements, so this notation is preliminary now. (See section 14.3.3 for the statement that there is only one field with four elements.) ♠

Example 14.3.6. Let $f(X) = X^2 + 1$ be a polynomial over $\mathbb{F}_2[X]$. Then from $(X + 1)^2 = 0$ and $X + 1 \neq 0$ we find that $\mathbb{F}_2[X]/(X^2 + 1)$ is not a field. ♠

The next section specifies which polynomials in fact yield new fields when quotiented by.

14.3.2 Irreducible Polynomials

Definition 14.3.7. A polynomial $f(X) \in R[X]$ is called *irreducible* if there don't exist $g(X), h(X) \in R[X]$ with $f(X) = g(X)h(X)$ such that neither $g(X)$ nor $h(X)$ is a unit. We make an exception for the polynomial 0, which is not irreducible. Any polynomial that is not irreducible, is called *reducible*.

Example 14.3.8. Consider polynomials with complex coefficients. Then by the fundamental theorem of algebra, if the degree of a polynomial is at least 1, then it has a root in $\mathbb{C}$. Hence, by Corollary 14.2.8 it is only irreducible if it has degree 1. ♠

Example 14.3.9. The polynomial $3X + 6$ is of degree 1, hence irreducible in $\mathbb{C}$. It is also irreducible over the fields $\mathbb{R}$ and $\mathbb{Q}$. But when considered a polynomial over the integers, it is reducible:

$$3X + 6 = 3 \cdot (X + 2)$$

as the integer 3 is not a unit in $\mathbb{Z}$. (It is in $\mathbb{Q}, \ldots$) ♠

Example 14.3.10. The polynomial $X^4 + X^2 + 1$ is not irreducible over $\mathbb{R}$. It has no roots, since $(-)^4$ and $(-)^2$ are always non-negative, $(-)^4 + (-)^2 + 1$ is hence always positive. So it is the product of two quadratic polynomials. In fact

$$(X^2 - X + 1)(X^2 + X + 1) = X^4 + X^3 + X^2 - X^3 - X^2 - X + X^2 + X + 1 = X^4 + X^2 + 1.$$

♠

Example 14.3.11. The polynomial $X^4 + X^2 + 1$ is not irreducible over $\mathbb{F}_2[X]$. We have the above factorization here as well. Indeed:

$$(X^2 + X + 1)^2 = X^4 + X^2 + 1.$$

The polynomial $X^4 + X + 1$ is irreducible. To show this, we first show that it has no linear factors, using Corollary 14.2.8. Indeed $0^4 + 0 + 1 = 1$ and $1^4 + 1 + 1 = 1$. So $X^4 + X + 1$ is the product of two irreducible quadratic polynomials. In Exercise 14.L we show that the only irreducible quadratic polynomial is $X^2 + X + 1$. The result then follows from the factorization of $X^4 + X^2 + 1$. ♠

An algorithm to check whether a polynomial is irreducible over a finite field $\mathbb{F}_p(= \mathbb{Z}/p\mathbb{Z})$ is given by

Theorem 14.3.12. (Rabin Test) *Let $f(X) \in \mathbb{F}_p[X]$ be a polynomial of degree m. Then $f(X)$ is irreducible iff $f(X) \mid X^{p^m} - X$ and for all prime d with $d \mid m$ we have $\gcd(f(X), X^{p^{m/d}} - X) = 1$.*

Example 14.3.13. We will show that $f(X) = X^4 + X + 1$ is irreducible over $\mathbb{F}_2$ by using the Rabin Test.

Note that $\deg f = 4$, so we need $f(X) \mid X^{2^4} - X$. For that we use a longdivision and find

$$X^{16} + X = (X^4 + X + 1)(X^{12} + X^9 + X^8 + X^6 + X^4 + X^3 + X^2 + X)$$

so indeed $f(X) \mid X^{2^4} - X$.

Now the only prime d with $d \mid 4$ is $d = 2$, so we have to check $\gcd(f(X), X^4 - X) = \gcd(f(X), X^4 + X) = \gcd(X^4 + X + 1, X^4 + X + 1)$. Since $X^4 + X + 1 - X^4 + X = 1$, we indeed immediately see that $\gcd(f(X), X^4 - X) = 1$.

Hence by the Rabin Test, $X^4 + X + 1$ is irreducible. ♠

Its pseudocode counterpart:

```
Data: $f(X) \in \mathbb{F}_p[X]$ of degree $n$ and $p_1, \dots, p_k$ the distinct prime divisors of $n$
Result: “$f(X)$ is reducible” or “$f(X)$ is irreducible”
1. Str ← ;
2. for $j = 1$ to $k$ do
    $n_j \leftarrow \frac{n}{p\,j}$
end
3. for $i = 1$ to $k$ do
    $h(X) \leftarrow X^{p^{n_i}} - X \pmod{f(X)}$;
    $g(X) \leftarrow \gcd(f(X), h(X))$;
    if $g(X) \neq 1$ then
        Str ← “reducible”;
        break $i$;
    end
end
4. $g \leftarrow X^{p^n} - X \pmod{f(X)}$;
5. if $g(X) = 0$ then
    Str ← “irreducible”
    else
        Str ← “reducible”
    end
end
return $f(X)$ is “Str”
```

Algorithm 6: Rabin test for irreducible polynomials

14.3.3 New fields

Theorem 14.3.14. *Let k be a field and $f(X) \in k[X]$ an irreducible polynomial. Then $k[X]/(f(X))$ is a field.*

Proof. We need to show that for any polynomial $g(X) \in k[X]$ for which $\overline{g(X)}$ is non-zero, we have an inverse. Or that any polynomial which is not a multiple of $f(X)$ has an inverse.

So suppose that $g(X)$ is not a multiple of $f(X)$. Then what is $\gcd(g(X), f(X))$? It is either a unit or a multiple of $f(X)$, since $f(X)$ is irreducible.

Suppose that $\gcd(g(X), f(X)) = xf(X)$ for some $x \in k$. But then $xf(X)$ divides $g(X)$, so $g(X)$ is a multiple of $f(X)$, a contradiction.

So suppose that $\gcd(g(X), f(X)) = x$ for some $x \in k^*$. Then using the Extended Euclidean Algorithm for polynomials, we can find $a(X), b(X)$ with

$$a(X)f(X) + b(X)g(X) = x.$$

Then $\frac{1}{x}(a(X)f(X) + b(X)g(X)) = 1$. Since we work modulo $f(X)$, this yields $\frac{1}{x}b(X)g(X) = 1$. Hence we have found an inverse to $g(X)$ in the form of $\frac{1}{x}b(X)$. □

Conversely, $k[X]/(f(X))$ being a field implies that $f(X)$ is irreducible.

Theorem 14.3.15. *Let k be a field and $f(X) \in k[X]$ a reducible polynomial. Then $k[X]/(f(X))$ is not a field.*

Proof. (This proof uses Exercise 14.F.) Since $f(X)$ is reducible, there exist $a(X), b(X)$ with $f(X) = a(X)b(X)$ with $a(X), b(X)$ both non-invertible. Then they both have degree lower than $f(X)$. Hence they are non-zero modulo $f(X)$. Their product is, however zero, so $a(X)$ and $b(X)$ are zero-divisors, and hence $k[X]/f(X)$ is not a field. □

Remark. We most often demand an irreducible polynomial to be monic when constructing new fields. The reasons for this are considered higher-level mathematics, so will not be discussed in this text.

The observation that degree 1 polynomials are irreducible over fields can also be proven using these theorems:

Corollary 14.3.16. *Let k be a field. Then if $f(X) \in k[X]$ has degree 1, it is irreducible.*

Sketch of proof: Since f has degree 1 and k is a field, we can write $f = X - a$. So $k[X]/(f(X)) = k[X]/(X - a)$. Now all elements in $k[X]/(X - a)$ are just polynomials evaluated at a, so elements in k. Hence $k[X]/(X - a) \subset k$. Conversely, all constants are polynomials and we have $k \subset k[X]/(X - a)$. Now since $k[X]/(f(X))$ is a field, we know that f is irreducible. □

By determining irreducible polynomials over finite fields, we can construct new finite fields.

Example 14.3.17. Let $f(X) = X^2 + X + 2$ over $\mathbb{F}_3[X]$. Then $\mathbb{F}_3[X]/(X^2 + X + 2)$ is a field. It has 9 elements. We show that $X^2 + X + 2$ is irreducible. For that we only have to check whether it has roots: $f(0) = 2, f(1) = 1, f(2) = 2$. ♠

Example 14.3.18. Let $f(X) = X^4 + X + 1$ over $\mathbb{F}_2[X]$. Then $\mathbb{F}_2[X]/(X^4 + X + 1)$ is a field with 16 elements. See also Example 14.3.11. ♠

Theorem 14.3.19. *Let p be a prime number and n a positive integer. Then there exists a field with p^n elements. In particular, if we have two fields with p^n elements, they are isomorphic.*

By the above theorem, we can officially talk about *the* field with p^n elements and write $\mathbb{F}_{p^n}$ for it. So we have seen $\mathbb{F}_9$ and $\mathbb{F}_{16}$ in the previous two examples and $\mathbb{F}_4$ earlier.

We list without proof a theorem with which it is easy to calculate the number of irreducible polynomials over $\mathbb{F}_q$ of some given degree d.

Theorem 14.3.20. *Let q be a prime power. Write A_d for the number of monic irreducible polynomials in $\mathbb{F}_q[X]$ of degree d. Then*

$$q^n = \sum_{d|n} dA_d.$$

Example 14.3.21. We apply the previous theorem to find A_1, A_2, A_3, A_4, A_6 :

$$\begin{aligned} q = 1 \cdot A_1 &\implies A_1 = q \\ q^2 = 1 \cdot A_1 + 2 \cdot A_2 &\implies A_2 = \frac{1}{2}(q^2 - q) \\ q^3 = 1 \cdot A_1 + 3 \cdot A_3 &\implies A_3 = \frac{1}{3}(q^3 - q) \\ q^4 = 1 \cdot A_1 + 2 \cdot A_2 + 4 \cdot A_4 &\implies A_4 = \frac{1}{4}(q^4 - q^2) \\ q^6 = 1 \cdot A_1 + 2 \cdot A_2 + 3 \cdot A_3 + 6 \cdot A_6 &\implies A_6 = \frac{1}{6}(q^6 - 3q^3 - 2q^2 - q) \end{aligned}$$

♠

14.4 Exercises

14.A Let $(R_1, +_1, \cdot_1)$ and $(R_2, +_2, \cdot_2)$ be two rings. Define on $R_1 \times R_2 = \{(r_1, r_2) \mid r_1 \in R_1, r_2 \in R_2\}$ addition and multiplication coordinatewise:

$$(r_1, r_2) + (s_1, s_2) = (r_1 +_1 s_1, r_2 +_2 s_2)$$

and

$$(r_1, r_2) \cdot (s_1, s_2) = (r_1 \cdot_1 s_1, r_2 \cdot_2 s_2).$$

(a) Show that $(R_1 \times R_2, +, \cdot)$ is a ring.

(b) If both R_1, R_2 are commutative, is $R_1 \times R_2$ commutative? Justify your answer.

(c) If both R_1, R_2 are fields, is $R_1 \times R_2$ a field?

14.B Let R be a ring and write R^* for the set of invertible elements. Show that R^* is a group (under the multiplication of the ring).

14.C Let R_1, R_2 be rings. Show that $(R_1 \times R_2)^* = R_1^* \times R_2^*$.

14.D Let R be a ring, let $a, b \neq 0$ be such that $ab = 0$. We call such a and b *zero-divisors*.

(a) Show that 2 is a zero-divisor in the ring $\mathbb{Z}/4\mathbb{Z}$. Show that 2 is a zero-divisor in the ring $\mathbb{Z}/6\mathbb{Z}$.

(b) Let n be a positive integer. Show that 2 is a zero-divisor in any ring $\mathbb{Z}/2n\mathbb{Z}$.

(c) Let n, m be positive integers. Show that whenever $\gcd(m, n) \neq 1$, that m is a zero-divisor in $\mathbb{Z}/n\mathbb{Z}$.

14.E Let R be a ring. Suppose that a is a zero-divisor (see previous exercise). Show that a is not invertible.

14.F Let k be a field. Show that k has no zero-divisors.

14.G Consider the ring $\mathbb{Z}/4\mathbb{Z}[X]$. Let f be the polynomial $2X+2$. So $\deg f = 1$. What is $\deg(f+f)$ and what is $\deg(f \cdot f)$? Why does this not contradict Lemma 14.2.4?

14.H Show that the following generalisation of Proposition 14.2.5 is true:

Let R be a ring and $f(X), g(X)$ polynomials in $R[X]$ such that the leading coefficient of g is invertible. Then there exist unique $q(X), r(X) \in R[X]$ with

$$f(X) = q(X)g(X) + r(X)$$

and $\deg r(X) < \deg g(X)$.

[Hint: The proof of uniqueness is identical.]

14.I Determine for all positive integers n the greatest common divisor of $n^2 - 4$ and $n^3 - 8$.

14.J Explicitly define $\mathbb{F}_{25}$. [Hint: Since $25 = 5^2$ you need to search for a monic quadratic polynomial. There are 10 different, but correct, answers.]

14.K Explicitly give an isomorphism between $\mathbb{F}_3[X]/(X^2+1)$ and $\mathbb{F}_3[X]/(X^2+2X+2)$. [Hint: Write the addition and multiplication tables and play around with them.]

14.L Determine all monic quadratic irreducible polynomials in $\mathbb{F}_2[X]$.

14.M Determine all monic quadratic irreducible polynomials in $\mathbb{F}_3[X]$.

14.N Determine all monic irreducible polynomials of degrees 3 and 4 in $\mathbb{F}_2[X]$.

Chapter 15

Probability Theory

This chapter is almost stand-alone and is devoted to establishing probability theory and combinatorics with as end-goal, the birthday paradox. Many proofs will be omitted as we expect familiarity with the material up to (but not including) the birthday paradox.

15.1 Basic probability theory

Suppose that we throw a six-sided die one hundred times and keep tally of the number of eyes each time:

⚀, ⚃, ⚀, ⚀, ⚃, ⚄, ⚃, ⚀, ⚄, ⚀, ⚃, ⚁, ⚁, ⚁, ⚁, ⚃, ⚃, ⚅, ⚁, ⚁, ⚃, ⚁, ⚂, ⚁, ⚁, ⚄, ⚂, ⚄, ⚅, ⚅, ⚅, ⚅, ⚁,

⚁, ⚃, ⚁, ⚃, ⚄, ⚀, ⚅, ⚂, ⚀, ⚂, ⚂, ⚂, ⚁, ⚁, ⚃, ⚄, ⚁, ⚀, ⚂, ⚁, ⚅, ⚂, ⚃, ⚃, ⚄, ⚀, ⚁, ⚃, ⚀, ⚀, ⚄, ⚄, ⚃,

⚀, ⚅, ⚃, ⚅, ⚂, ⚀, ⚃, ⚅, ⚅, ⚅, ⚃, ⚅, ⚃, ⚁, ⚃, ⚂, ⚃, ⚀, ⚅, ⚁, ⚃, ⚅, ⚃, ⚁, ⚁, ⚄, ⚂, ⚄, ⚁, ⚄, ⚀, ⚁, ⚂, ⚅

The results in a table:

⚀	~~////~~ ~~////~~ ~~////~~	15
⚁	~~////~~ ~~////~~ ~~////~~ ~~////~~ ///	23
⚂	~~////~~ ~~////~~ //	12
⚃	~~////~~ ~~////~~ ~~////~~ ~~////~~ //	22
⚄	~~////~~ ~~////~~ //	12
⚅	~~////~~ ~~////~~ ~~////~~ /	16

Then one might want to know the probability of the next roll coming up ⚀. Out of the previous one hundred tests, 15 have come up ⚀, so we might expect a probability of 15/100. (So in about 6 or 7 tries, we expect a ⚀. The amount of tries is $100/15 \approx 6.6667$.)

Now for a true definition of probability,[1] we should regard an *experiment* $\mathcal{E}$ and the *universe of all possible outcomes of the experiment* $\mathcal{E}$, $\Omega_{\mathcal{E}}$. Any subset $A \subset \Omega_{\mathcal{E}}$ is called an *event*. If we assume that every outcome of the experiment has equal "chance" of happening, then the probability of A is given as

$$\Pr[A] = \frac{\#A}{\#\Omega}.$$

[1]This is just one example, there are many more, which we shall see later. This particular one is called the *vase model*.

Example 15.1.1. When rolling a six-sided die, which is assumed *fair*, the probability of rolling a ⚀ is $\frac{1}{6}$. This is due to the universe having six possible outcomes. Now consider the event $A = \{$⚀, ⚂, ⚄$\}$. This means, the outcome of the next roll is either a ⚀, a ⚂ or a ⚄. Then

$$\Pr[A] = \frac{\#A}{\#\Omega} = \frac{3}{6}.$$

♠

Example 15.1.2. A shopkeeper holds a lottery. He has a opaque vase in which are twenty balls. Five of them are red, the others white. The first customer can pull five balls out of the vase, and every next customer can pull out one ball. The customer who pulls out the fifth red ball wins a discount of 10% on his/her items. The probability that the first customer on his first pull gets a red ball is 5/20, on the second pull it is 5/19 if the first wasn't a red one, and 4/19 if the first was indeed a red ball. The probability that both the first and second balls are red is given by $5/20 \cdot 4/19 = 1/19$. ♠

The preceding example gives rise to many questions about probability, we give a few examples:

Question 15.1.3. *Is this lottery fair? (I.e., does every customer have an equal chance of winning?)*

Question 15.1.4. *What is the probability that the second customer wins the lottery?*

Question 15.1.5. *Which customer has the highest probability of winning?*

Question 15.1.6. *What is the probability that the kth customer wins the lottery when we hold this lottery with n balls?*

Some of these questions hint towards answers of others, and some of them even give exact answers to others. At the end of this chapter, not one of these questions should pose a problem.

15.2 Combinatorics

All we have right now for probability is the formula for the probability of an event to happen in a vase model. This in practice comes down to counting numbers of elements in sets. There are many things to learn about counting, to which this section is devoted.

Definition 15.2.1. We define the set $\underline{n}$ inductively as:

$$\underline{n} = \begin{cases} \emptyset & \text{if } n = 0; \\ \underline{n-1} \cup \{n-1\} & \text{else.} \end{cases}$$

In practice this comes down to:

$$\underline{0} = \emptyset, \underline{1} = \{0\}, \underline{2} = \{0, 1\}, \ldots, \underline{n} = \{0, \ldots, n-1\}.$$

Definition 15.2.2. A set A is *finite* if there exists a bijection $\mathrm{f}\colon A \to \underline{n}$ for some positive integer n. Then we say that A has n elements, written $\#A = n$. If no such bijection exists, we say that A is *infinite.*

The above were assumed intuitively clear in the preceding chapters concerning algebra and number theory, but right now, we wanted to make this more precise.

Proposition 15.2.3. *Let $A, B \subset \Omega$ be subsets of a finite set Ω. Then* *(a)* *A and B are finite;*

(b) *$A \cup B$ is finite and $\#(A \cup B) \leq \#A + \#B$;*

(c) *If $A \cap B = \emptyset$, then $\#(A \cup B) = \#A + \#B$;*

(d) *If $A \subset B$, then $\#A \leq \#B$;*

(e) *$\#(A \cup B) = \#A + \#B - \#(A \cap B)$;*

(f) *$\#(A \times B) = \#A \cdot \#B$;*

(g) *$\#(\Omega \setminus A) = \#X - \#A$;*

(h) *$\#(A \setminus B) = \#A - \#(A \cap B)$.*

The expression $\Omega \setminus A$ is called the *complement of A in Ω* and is often written A^c, but *only* when A is contained in some universe Ω. When talking about $B \setminus A$ we still call this the complement of A in B, but will not write A^c for this.

These rules should be clear intuitively, a proof will hence be omitted. The claim (e) can be generalized to arbitrary unions. As an example, we first give a union of three sets.

Example 15.2.4. Let $A, B, C \subset \Omega$ be subsets of a finite set Ω. Then

$$\#(A \cup B \cup C) = \#A + \#B + \#C - \#(A \cap B) - \#(A \cap C) - \#(B \cap C) + \#(A \cap B \cap C).$$

Write $D = B \cup C$. Then by (e) of the previous proposition, we have

$$\begin{aligned}
\#(A \cup B \cup C) =& \#(A \cup D) \\
=& \#A + \#D - \#(A \cap D) \\
=& \#A + \#(B \cup C) - \#(A \cap (B \cup C)) \\
=& \#A + \#B + \#C - \#(B \cap C) - \#((A \cap B) \cup (A \cap C)) \\
=& \#A + \#B + \#C - \#(B \cap C) - \#(A \cap B) - \#(A \cap C) + \#(A \cap B \cap A \cap C) \\
=& \#A + \#B + \#C - \#(B \cap C) - \#(A \cap B) - \#(A \cap C) + \#(A \cap B \cap C)
\end{aligned}$$

Note that we have used (e) three times (where?). ♠

The above example is an example of the Principle of Inclusion/Exclusion.

Theorem 15.2.5 (Principle of Inclusion/Exclusion)**.** *Let $A_1, \ldots, A_k \subset \Omega$ be subset of a finite set Ω. Then*

$$\#\left(\bigcup_{i=1}^{k} A_i\right) = \sum_{J \subset \{1,\ldots,k\}} (-1)^{\#J-1} \#\left(\bigcap_{j \in J} A_j\right).$$

Given k different numbers, letters, symbols, etc. how many ways are there to sort them? For $k = 1$ and $k = 2$ it is obvious, 1 respectively 2 ways. But with three letters A, B, C we have six options:

$$ABC, ACB, BAC, BCA, CAB, CBA.$$

With four symbols, we have 24 options. One notices the following pattern:

$$1, 2 \cdot 1, 3 \cdot 2 \cdot 1, 4 \cdot 3 \cdot 2 \cdot 1, \ldots$$

This is indeed what happens. Given k different letters and to order them, for the first letter, there are k possibilities. Then for the second letter, there are $k - 1$ possibilities left, etcetera until we have only one possibility for the last letter. This yields

$$k! := k \cdot (k-1) \cdot (k-2) \cdots 3 \cdot 2 \cdot 1.$$

Now, given k different letters, how many ways are there to choose two out of them and put them in one basket and discarding the rest? For the first option, we have k possibilities, while for the second, we have only $k-1$ possibilities remaining. However, choosing first A and then B yields the same result as first B and then A. Hence we need to divide by 2. The result is $\frac{k(k-1)}{2}$.

In general, given n different letters, how many ways are there to choose k out of them and putting them in one basket and discarding the rest? To do this, we note that there are $n!$ ways to order the letters. Then we take the first k to put them in the basket. Note that these k can be in any order, so we have to divide by $k!$. Also, the remaining $n-k$ can be in any order, so we have to divide by $(n-k)!$. This yields the formula

$$\frac{n!}{k!(n-k)!}$$

which will shorthand be written as $\binom{n}{k}$.

There are many equalities that hold for $\binom{n}{k}$, which we will last later. First we will state Newton's binomial formula and Pascal's Triangle.

Theorem 15.2.6 (Newton's binomial theorem)**.** *Let $a, b \in \mathbb{Z}$ and n a positive integer. Then*

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k.$$

Proof. The proof is quite easy by induction.

For $n = 0$, the LHS equals 1, while the RHS is a sum from 0 to 0 so yields $\binom{0}{0} a^0 b^0 = 1$.

For $n = 1$, the LHS equals $a + b$, while the RHS is a sum from 0 to 1 so yields $\binom{1}{0} a^1 b^0 + \binom{1}{1} a^0 b^1 = 1 \cdot a \cdot 1 + 1 \cdot 1 \cdot b = a + b$.

For $n = 2$, the LHS equals $a^2 + ab + ba + b^2 = a^2 + 2ab + b^2$. The RHS is a sum from 0 to 2 so yields

$$\binom{2}{0} a^2 b^0 + \binom{2}{1} a^1 b^1 + \binom{2}{2} a^0 b^2 = a^2 + 2ab + b^2.$$

Now let's assume that it holds for some n. Then

$$\begin{aligned}
(a+b)^{n+1} =& (a+b)(a+b)^n \\
=& (a+b) \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k \\
=& \sum_{k=0}^{n} \binom{n}{k} a^{n-k+1} b^k + \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^{k+1} \\
=& \sum_{k=0}^{n+1} \binom{n}{k} a^{n-k+1} b^k + \sum_{j=1}^{n+1} \binom{n}{j-1} a^{n-(j-1)} b^j
\end{aligned}$$

The first equality should be clear, while the second is applying the induction hypothesis. The third equality should again be clear. Then in the first sum, we note that $\binom{n}{n+1} = 0$, so increasing the range from n to $n+1$ does not change the value of the sum. In the second sum we substitute j for $k+1$, (or $j-1$ for k) to also obtain the high-end of the range at $n+1$. We continue:

$$(a+b)^{n+1} = \sum_{k=0}^{n+1} \binom{n}{k} a^{n-k+1} b^k + \sum_{k=1}^{n+1} \binom{n}{k-1} a^{n-(k-1)} b^k$$

$$= \sum_{k=0}^{n+1} \binom{n}{k} a^{n-k+1} b^k + \sum_{k=0}^{n+1} \binom{n}{k-1} a^{n-k+1} b^k$$
$$= \sum_{k=0}^{n+1} \left(\binom{n}{k} + \binom{n}{k-1} \right) a^{n-k+1} b^k$$

In the first equality we just replace all j's with k's, then we add 0 to the low-end of the range in the second sum, since $\binom{n}{-1} = 0$ anyway. Then in the last equality we just take the sums together. What is left to prove is the equality

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}.$$

This can be easily done by writing out the formulas for $\binom{n}{k}$ (see Exercise 15.H, or viewed in Pascal's Triangle (see below). Here we present a different proof using combinatorics.

Write $x_1, \ldots, x_n, x_{n+1}$ for the letters from which we want to choose k. We can divide the number of ways to choose those into two categories:

- The ones that do not contain x_{n+1}, of those there are $\binom{n}{k}$.
- The ones that do indeed contain x_{n+1}, of those there are $\binom{n}{k-1}$.

So indeed $\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$. □

Below we present Pascal's Triangle, which easily shows the binomial coefficients for some small n. Remember that it starts with $n = 0$ (as the first row) and $k = 0$ (in each first column). In particular, the top is $\binom{0}{0}$.

```
                              1
                          1       1
                      1       2       1
                  1       3       3       1
              1       4       6       4       1
          1       5      10      10       5       1
      1       6      15      20      15       6       1
  1       7      21      35      35      21       7       1
```

Figure 15.1: Pascal's Triangle, the rows represent n, the columns represent k. In particular $\binom{6}{4} = 15$. (Remember that both the rows and columns start from 0.

Lemma 15.2.7. *Let n be a positive non-negative integer, then we have*

(a) $\binom{n}{0} = 1$;

(b) $\binom{n}{n} = 1$;

(c) $\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$;

(d) $\binom{n}{k} = \binom{n}{n-k}$.

Proof. The first two items are trivial to verify. The third item is proved in the proof of the Binomial Theorem. For the last one, note that choosing k of the n letters to keep is of course the same as choosing $n-k$ of the n letters to discard. (Or observe that Pascal's Triangle is symmetric.) □

By Newton's binomial theorem, we call $\binom{n}{k}$ binomial coefficients. One can also say that $\binom{n}{k}$ is the number of subsets with k elements of a subset of n elements. Then the following rule is easy:

Proposition 15.2.8. *Let n be a non-negative integer. Then*

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n.$$

Proof. We know that the number of subsets of a set with n elements is 2^n. Since we can also group them by the number of elements they contain and then add those numbers, the result follows. □

If the order of the k chosen elements does matter, we only need to omit the factor $k!$ in the denominator. So we get

$$\frac{n!}{(n-k)!}$$

for the number of ordered picks from a vase.

15.3 More probability theory

Next to the vase model, there are of course other models for which we have probabilities. For example, let's consider an unfair six-sided die. It will roll a ⚅ with probability $\frac{1}{3}$, a ⚀ or ⚁ with probability $\frac{1}{12}$ and ⚂,⚃,⚄ each with probability $\frac{1}{6}$. Then this doesn't clearly satisfy our previous model.

For this we will assign a variable to the undetermined outcome of an experiment. We call this a *stochastic variable.* So in the preceding example, we introduce the stochastic variable X for the outcome of one roll of the die.

We write:

$$\Pr[X=1] = \Pr[X=2] = \frac{1}{12}, \Pr[X=3] = \Pr[X=4] = \Pr[X=5] = \frac{1}{6}, \Pr[X=6] = \frac{1}{3}.$$

Now we can also write more interesting stuff like:

$$\Pr[4 \leq X] = \Pr[X=4] + \Pr[X=5] + \Pr[X=6] = \frac{2}{3}.$$

Or

$$\Pr[X \neq 1] = \Pr[X > 1] = \frac{11}{12}.$$

The preceding example can be modelled as a vase model. (See Exercise 15.J.)

Question 15.3.1. *Consider a fair six-sided die. What is the expected number of throws before we have the first ⚅? In particular, there an infinite amount of possible outcomes to this experiment.*

So indeed, given an experiment $\mathcal{E}$, we still have a universe of outcomes pertaining to this experiment $\Omega_{\mathcal{E}}$ which no longer has to be finite.

Definition 15.3.2. Given an experiment $\mathcal{E}$ and events A, B within the universe of outcomes, we have the following terminology. Write ω for the result of the experiment.

- $\omega \in A$ is called "A occurs";
- $\omega \in A \cap B$ is called "both A and B occur";

- $\omega \in A \cup B$ is called "A or B occurs";
- $\omega \in (A \cup B) \setminus (A \cap B)$ is called "either A or B occurs (but not both)";
- $\omega \in A^c$ is called "A does not occur";
- $A \cap B = \emptyset$ is called "A and B are mutually exclusive";
- $A \subset B$ is called "if A occurs, then certainly B occurs".

For example, let $\mathcal{E}$ be the experiment of throwing two (fair) six-sided dice and computing the sum of the rolls. Let A be the event: the sum of both values is even and let B be the event: the sum of both values is six. Then clearly we have

$$A = \{2, 4, 6, 8, 10, 12\} \qquad B = \{6\} \qquad \Omega_{\mathcal{E}} = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}.$$

In particular, when B occurs, then A occurs.

Given an experiment, we want to define a *probability function* Pr, that assigns to any event a value between 0 and 1 that satisfies all rules that are intuitively clear. For example $\Pr[\Omega] = 1$ should hold and $\Pr[\emptyset] = 0$ as well.

Definition 15.3.3. Given an experiment $\mathcal{E}$ and its corresponding universe of outcomes $\Omega_{\mathcal{E}}$, then we define a *probability function* as any function $\Pr\colon \mathcal{P}(\Omega_{\mathcal{E}}) \to [0, 1]$ for which the following three properties hold:

1. $\Pr[A] \geq 0$ for all $A \subset \Omega_{\mathcal{E}}$;
2. $\Pr[\Omega_{\mathcal{E}}] = 1$;
3. For any sequence (A_k) of mutually exclusive events we have
$$\Pr[\bigcup A_k] = \sum \Pr[A_k].$$

Note that in the third item it does not matter whether the sequence is finite or infinite, and that the first property is already given in the fact that the co-domain of Pr is $[0, 1]$.

From these three properties, we can deduce the following properties:

Proposition 15.3.4. *Let $A, B, A_1, A_2, \ldots$ be events in a universe of outcomes Ω. Then*

1. $\Pr[\emptyset] = 0$;
2. $\Pr[A \setminus B] = \Pr[A] - \Pr[A \cap B]$;
3. $\Pr[A \cup B] = \Pr[A] + \Pr[B] - \Pr[A \cap B]$;
4. $\Pr[A^c] = 1 - \Pr[A]$;
5. *If* $A \subset B$, *then* $\Pr[A] \leq \Pr[B]$.

Proof.
1. Note that $\Pr[\emptyset] = \Pr[\emptyset \cup \emptyset] = \Pr[\emptyset] + \Pr[\emptyset]$. Hence $\Pr[\emptyset] = 0$.
2. Note that $A = A \setminus B \cup (A \cap B)$ and those are disjoint. Hence $\Pr[A] = \Pr[A \setminus B] + \Pr[A \cap B]$. The result follows.
3. Note that $A \cup B = A \cup B \setminus A$, and that $A \cap (B \setminus A) = \emptyset$. So $\Pr[A \cup B] = \Pr[A] + \Pr[B \setminus A]$. By the previous item: $\Pr[B \setminus A] = \Pr[B] - \Pr[A \cap B]$. So altogether
$$\Pr[A \cup B] = \Pr[A] + \Pr[B \setminus A] = \Pr[A] + \Pr[B] - \Pr[A \cap B].$$

4. We have $\Omega = A^c \cup A$ and $A^c \cap A = \emptyset$. So $1 = \Pr[\Omega] = \Pr[A^c \cup A] = \Pr[A^c] + \Pr[A]$. The result follows.

5. Note that $A \cap B = A$, so $\Pr[B \setminus A] = \Pr[B] - \Pr[A]$. Since $\Pr[B \setminus A] \geq 0$, we must have $\Pr[B] - \Pr[A] \geq 0$, hence $\Pr[A] \leq \Pr[B]$.

$\square$

Now we can make a model for the case in Question 15.3.1. Note that we can calculate for each n the probability that the nth roll is the first ⚅.

n	$\Pr[n\text{th roll is first ⚅}]$	$\approx$
1	$\frac{1}{6}$	0.1666667
2	$\frac{5}{6} \cdot \frac{1}{6}$	0.1388888
3	$(\frac{5}{6})^2 \cdot \frac{1}{6}$	0.1157407
4	$(\frac{5}{6})^3 \cdot \frac{1}{6}$	0.0964506
5	$(\frac{5}{6})^4 \cdot \frac{1}{6}$	0.0803755
6	$(\frac{5}{6})^5 \cdot \frac{1}{6}$	0.0669795
7	$(\frac{5}{6})^6 \cdot \frac{1}{6}$	0.0558163
8	$(\frac{5}{6})^7 \cdot \frac{1}{6}$	0.0465136
9	$(\frac{5}{6})^8 \cdot \frac{1}{6}$	0.0387613

Example 15.3.5. Consider a deck of 52 playing cards, with suits $\diamondsuit, \spadesuit, \heartsuit, \clubsuit$ and ranks $2, 3, \ldots, 9, 10, J, Q, K, A$. The probability of drawing any specific card from a well-shuffled deck is of course $\frac{1}{52}$. Also, the probability of drawing a $\clubsuit$ is $\frac{1}{4}$. The probability of drawing a $\heartsuit$ that has as rank a number is the event:

$$\{\heartsuit\} \setminus \{J, Q, K, A\}$$

Note that the probability then becomes

$$\Pr[\heartsuit] - \Pr[\heartsuit \cap \{J, Q, K, A\}] = \frac{1}{4} - \frac{4}{52} = \frac{13}{52} - \frac{4}{52} = \frac{9}{52}$$

and not $\Pr[\heartsuit] - \Pr[\{J, Q, K, A\}] = \frac{1}{4} - \frac{4}{13} = -\frac{3}{52}$! $\spadesuit$

15.4 Conditional probabilities

When returning to the example of a vase, getting information on the first ball pulled from the vase influences the probability of pulling a red ball on the second try. For example, consider an opaque vase with 20 balls of which 6 are red. The probability that the second ball is red is given by:

$$\frac{6}{20}\frac{5}{19} + \frac{14}{20}\frac{6}{19} = \frac{3}{10}$$

Suppose that we already know that the first ball is red, then the probability becomes

$$\frac{5}{19} \approx 0.26315,$$

while if the first ball is white, we get

$$\frac{6}{19} \approx 0.31578.$$

Definition 15.4.1. Suppose that an experiment has an event A with $\Pr[A] \neq 0$. Then the probability that an event B happens provided that A has happened, written $\Pr[B \mid A]$ is given by

$$\Pr[B \mid A] = \frac{\Pr[A \cap B]}{\Pr[A]}.$$

This is called a *conditional probability*.

With the example above, say that the event A is that the first ball is red, and say that the event B is the second ball is red. Then we have

$$\Pr[B \mid A] = \frac{\Pr[A \cap B]}{\Pr[A]} = \frac{\frac{6}{20}\frac{5}{19}}{\frac{6}{20}}$$

which is exactly what we've expected.

Theorem 15.4.2 (Bayes' Theorem)**.** *Let A and B be events with $\Pr[A] \neq 0$. Then we have*

$$\Pr[B \mid A] = \frac{\Pr[A \mid B] \Pr[B]}{\Pr[A]}.$$

Example 15.4.3. In athletics tournaments the participants are checked for using doping. A doctor has procured a test that gives a positive result in 95% of cases when a subject has used doping. When a subject has not used doping there is a 98% chance that the test gives a negative result. Assuming that 1% of the athletes indeed uses doping, we want to calculate the probability that a test-subject with a positive test result is indeed a doping user.

Write A for the event that an athlete tests positive, and B for the event that an athlete uses doping. Then we get to calculate $\Pr[B \mid A]$. We use the formula:

$$\begin{aligned} \Pr[B \mid A] =& \frac{\Pr[A \mid B] \Pr[B]}{\Pr[A]} \\ =& \frac{0.95 \cdot 0.01}{\Pr[A]} \\ =& \frac{0.95 \cdot 0.01}{0.95 \cdot 0.01 + 0.02 \cdot 0.99} \\ =& \frac{0.0095}{0.0293} \\ \approx& 0.32423 \end{aligned}$$

Indeed, more athletes will be tested positive than are actually using doping. This is due to the fact that the group of non-users is much larger than the group of users. ♠

Example 15.4.4. When repeatedly performing the same experiment, where only two different outcomes get considered, one gets the so-called *binomial distribution.* Say that we throw a (fair) six-sided die and are only considered with the fact whether the result is ⚅ or not. The probability for ⚅ is $\frac{1}{6}$. Say we perform this experiment n times, then the probability that we have exactly k times ⚅ is given by

$$\binom{n}{k}(\frac{1}{6})^k(\frac{5}{6})^{n-k}.$$

One can easily verify this formula, since every roll of the die is independent of the previous rolls. ♠

In probability theory, independence is an important feat. In the previous example, we saw that when rolling two dice, the results of the dice are independent of each other. Now we can multiply the probabilities:

$$\Pr[⚅, ⚅] = \Pr[⚅] \cdot \Pr[⚅].$$

Definition 15.4.5. Let A and B be events, then we call A and B *independent* of each other if we have

$$\Pr[A \cap B] = \Pr[A] \cdot \Pr[B].$$

If A and B are not independent, we call them *dependent.*

In particular, when A and B are independent, we have $\Pr[A \mid B] = \Pr[A]$. (Verify this!)

Example 15.4.6. Consider a deck of 52 playing cards. Each card has a suit and a rank. What is $\Pr[K \mid \spadesuit]$? We have $\Pr[K] = 1/13$ and $\Pr[\spadesuit] = 1/4$. Also, there is only one card $K\spadesuit$ ("King of Spades[2]"), hence $\Pr[K \cap \spadesuit] = 1/52$. Hence K and $\spadesuit$ are independent and $\Pr[K \mid \spadesuit] = \Pr[K] = 1/13$. $\spadesuit$

What happens if we have multiple events? When are they *independent*?

Definition 15.4.7. Let $A_1, \ldots, A_n$ be events. Then we say that they are independent if for any subset $\{i_1, \ldots, i_k\}$ of $\{1, \ldots, n\}$ we have

$$\Pr[A_{i_1} \cap \ldots \cap Pr[A_{i_k}] = \Pr[A_{i_1}] \cdots \Pr[A_{i_k}].$$

If we only have $\Pr[A_i \cap A_j] = \Pr[A_i] \cdot \Pr[A_j]$ for all $i \neq j$, we call $A_1, \ldots, A_n$ *pairwise independent.*

See Exercises 15.D and 15.E for detailed examples of what can go wrong with multiple events.

15.5 Birthday paradox

This section is dedicated to one specific problem in probability theory. Under certain assumptions, the probability that in a group of 23 individuals, there are two who share a birthday is larger than 50%. We first state the assumptions and then show the validity of this argument. Afterwards, we will pose a generalization of this problem.

Assume that every year has 365 days, hence the 29th of February is left out in our considerations. Furthermore, assume that there are no twins in the group, as twins typically have the same birthday. Also assume that every day has an equal probability for an individual to have his/her birthday. (In reality, not every date is equally popular, see e.g. http://thedailyviz.com/2016/09/17/how-common-is-your-birthday-dailyviz/.)

Under these assumptions, the probability that at least two individuals share a birthday is of course equal to 1 minus the probability that no individuals share a birthday. So we get:

$$1 - \frac{365}{365}\frac{364}{365}\cdots\frac{343}{365} = 1 - \left(\frac{1}{365}\right)^{23}(365 \cdot 364 \cdots 343) \approx 0.507297234.$$

One can easily verify this computation.

The reason for calling this the "Birthday Paradox", is that 23 seems as such a low number. In particular, to get a 99.9% probability, we only need 70 individuals.

Now suppose that we have a year with 1000 days, and each day equally probable to have as birthday. How many people do we need to have in a group to achieve a 50% probability of at least one shared birthday? The calculation is similar:

$$1 - \left(\frac{1}{1000}\right)^{k}\frac{1000!}{(1000-k)!} > 50\%.$$

Solving this for k gives $k \geq 38$.

With 100000 days or even 100000 days, these numbers will be ≥ 119, ≥ 373 respectively. Calculation by hand of these numbers is hard, but a computer algebra system can do this. Even calendars with 1000000 days can be calculated with the most direct (i.e., not very efficient) code in 25 minutes. Yielding a result $k \geq 1178$.

Note that although computer algebra systems can work with such large numbers, this takes a long computational time and cannot be verified by hand using at most a basic calculator. For this we have the following exposition.

[2] Which US army-man had the same nickname?

The Taylor expansion[3] of the exponential function e^x is given by

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \ldots$$

From this expansion, we remark that whenever $x \ll 1$ (i.e., very small, but still positive) a good approximation of e^x is

$$e^x \approx 1 + x.$$

Reminding

$$1 - \frac{365}{365}\frac{364}{365}\cdots\frac{343}{365}$$

we note that we can write this as

$$1 - 1 \cdot (1 - \frac{1}{365})(1 - \frac{2}{365})\cdots(1 - \frac{22}{365}).$$

Using the above approximation, this will turn out to be approximately equal to

$$1 - 1 \cdot e^{-\frac{1}{365}} \cdot e^{-\frac{2}{265}} \cdots e^{-\frac{22}{365}}.$$

With some elementary calculus we can conclude:

$$\begin{aligned} 1 - 1 \cdot e^{-\frac{1}{365}} \cdot e^{-\frac{2}{265}} \cdots e^{-\frac{22}{365}} &= 1 - 1 \cdot e^{-\frac{1+2+\ldots+22}{365}} \\ &= 1 - 1 \cdot e^{-\frac{11\cdot 23}{365}} \\ &= 1 - e^{-\frac{253}{365}} \end{aligned}$$

This last expression equals

$$1 - e^{-\frac{253}{365}} \approx 0.5000017522.$$

Since this last calculation can easily be made on a small calculator, this is a very useful approximation indeed.

15.6 Exercises

Note that most exercises here have in some for been taken from the (Dutch) lecture notes Inleiding Kansrekening by H. Maassen used for many years in educating Probability Theory to Mathematics, Physics and Computing Science students in Nijmegen.

15.A Which has highest probability of success, throwing at least one ⚅ in four rolls with one six-sided die, or throwing at least one (⚅,⚅) in twenty-four rolls with two six-sided dice?

15.B When throwing three six-sided dice and adding the number of eyes on the dice, the result 10 can be gotten in six ways: $631, 622, 541, 532, 442, 433$. The result 9 can also be gotten in six ways: $621, 531, 522, 441, 432, 333$. But when trying this, the number 10 is gotten more often than the number 9. What explanation can you give for this phenomenon?

15.C Suppose that n knights are to be seated at a table. Among them are Sir Lancelot the Brave and Sir Robin the Not-Quite-So-Brave-as-Sir-Lancelot. The assignment of seats is arbitrary/random. What is the probability that:

(a) Sir Lancelot and Sir Robin sit next to each other when all knights are seated at one side of the table;

(b) Sir Lancelot and Sir Robin sit next to each other when the table is round;

[3] See any text on calculus.

(c) Sir Lancelot and Sir Robin sit next to each other when n is even and at each of the long sides of the table $\frac{1}{2}n$ knights are seated;

(d) Sir Lancelot and Sir Robin sit opposite to each other in the situation of (c)?

15.D Let A, B, C be events. Show that is not enough to have $\Pr[A \cap B \cap C] = \Pr[A]\Pr[B]\Pr[C]$ for A, B and C to be independent. [Hint: Consider the experiment of tossing three coins, where A is the event that the first toss yields "heads", B is the event where in total there are more "heads" than "tails" and C the event that the last two tosses yield the same result.]

15.E In many houses a lamp near stairs can be switched on or off at the bottom of the stairs as well as at the top of the stairs. Each of the switches can be "up" ("1") or "down"("0").

Assume that the light is on whenever both switches are in the same position. Write X for the position of the downstairs switch and Y for the position of the switch upstairs.

Let events A, B, C be given as $A = [X = 1], B = [Y = 1]$ and $C =$ [the lamp is turned on]. Show that A, B, C are pairwise independent, but not independent.

15.F Suppose that A and B are independent events. Show that A and B^c are independent events. (Then A^c and B are also independent events, as are A^c and B^c.)

15.G Show that in a group of 70 individuals the probability that there are at least two who share a birthday is 99.9%.

15.H Prove $\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$ using the formula for $\binom{n}{k}$.

15.I Explain that $\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$ using Pascal's Triangle.

15.J Show how the example with the unfair die in the beginning of section 15.3 can be seen as an instance of the vase model.