Politecnico di Torino

Cybersecurity Master's Degree Course

Cryptography

Theory + Exercises & solutions

author: by1yr65J/pTKnX/a0fgQaDyNk8E1WStWhIkh+9Vqxwk=
email: JY/tLpu3EKzHfjkMdHoVaPMoc+2jiHlPdMJq6Oi0f+s=

Last modified on: March 13, 2024

About this paperwork

This paperwork has been produced by referring to the following sources:

- *Understanding Cryptography, A Textbook for Students and Practitioners* [C. Paar, J. Pelzl];
  - Some sections of this paperwork have been written by either copying or slightly modifying the corresponding sections of the book. Hence, I do not own the content of any of these sections and all rights are reserved to the authors of the book;
  - This paperwork only covers the main topics that have been included in the Cryptography exams at the Politecnico di Torino so far. Hence, this work doesn't give you a complete and detailed overview about Cryptography. If you want to learn something more about this amazing subject, I highly suggest you to buy the official release of the book;
- Wikipedia: mainly for Chinese Reminder Theorem (CRT) and quadratic residues;
- Overheads of the Cryptography course (2022-23);

Missing Topics

The following topics are not discussed in this paperwork, since the corresponding exercises are quite easy:

- Feistel networks and DES;

Please, refer to the "Holy Crypto Bible" for the solutions. For further explanations, refer to the book.

Adopted Notation

I adopted the same notation as the one used by the authors of

Understanding Cryptography, A Textbook for Students and Practitioners [C. Paar, J. Pelzl]

Hence, most of the letters and symbols used in the overheads of the Cryptography course (2022-23) differ from the ones used in this work.

For what concerns modular arithmetic and, mainly, modular equations, the following conventions have been adopted:

- the modulus $(\text{mod } m)$ of the equation is reported at the end of the equation itself, next to the result;
- the equality symbol $(=)$ is used when we're making calculations without applying the modulo operator, e.g., when we are re-writing an equation in a simpler (or just more elegant) way;
- the equivalence symbol $(\equiv)$ is used when we're making calculations by applying the modulo operator;

Warnings

This paperwork may contain errors and/or inaccuracies. Hence, by referring to this paperwork, you are taking into account the possibility of assimilating wrong/misleading concepts and information.

I DO NOT TAKE ANY RESPONSIBILITY FOR THE OUTCOME OF YOUR EXAM

Contents

1 Random Numbers

1.1 Random Number Generators

The security of stream ciphers hinges entirely on a "suitable" key stream $s_0, s_1, s_2, ...$ Since randomness plays a major role, we will learn about three types of random number generators (RNG) that are important for us.

1.1.1 True Random Number Generators (TRNG)

True random number generators (TRNGs) are characterized by the fact that their output cannot be reproduced. For instance, if we flip a coin 100 times and record the resulting sequence of 100 bits, it will be virtually impossible for anyone to generate the same 100 bit sequence (the chance of success is $1/2^{100}$). TRNGs are based on physical processes. Examples include coin flipping, rolling of dice, semiconductor noise, clock jitter in digital circuits and radioactive decay. In cryptography, TRNGs are often needed for generating session keys, which are then distributed, and for other purposes.

1.1.2 (Generalized) Pseudo Random Number Generators (PRNG)

Pseudorandom number generators (PRNGs) generate sequences which are *computed* from an initial seed value. Often they are computed recursively in the following way:

$$\begin{aligned} s_0 &= \text{seed} \\ s_{i+1} &= f(s_i), \quad i = 0, 1, ... \end{aligned}$$

A generalization of this are generators of the form $s_{i+1} = f(s_i, s_{i-1}, ..., s_{i-t})$, where t is a fixed integer.
A popular example is the **linear congruential generator**:

$$\boxed{\begin{aligned} s_0 &= \text{seed} \\ s_{i+1} &\equiv as_i + b \bmod m, \quad i = 0, 1, ... \end{aligned}}$$

where *a,b,m* are integer constants. Note that PRNGs are not random in a true sense because they can be computed and are thus completely deterministic. A widely used example is the `rand()` function used in ANSI C. It has the parameters:

$$\begin{aligned} s_0 &= 12345 \\ s_{i+1} &\equiv 1103515245s_i + 12345 \bmod 2^{31}, \quad i = 0, 1, ... \end{aligned}$$

A common requirement of PRNGs is that they process good statistical properties, meaning their output approximates a sequence of true random numbers. There are many mathematical tests which can verify the statistical behavior of PRNG sequences. Note that there are many applications for pseudorandom numbers outside cryptography. For instance, many types of simulations or testing need random data as input. That is the reason why a PRNG is included in the ANSI C specification.

1.1.3 Cryptographically Secure Pseudorandom Number Generators (CSPRNG)

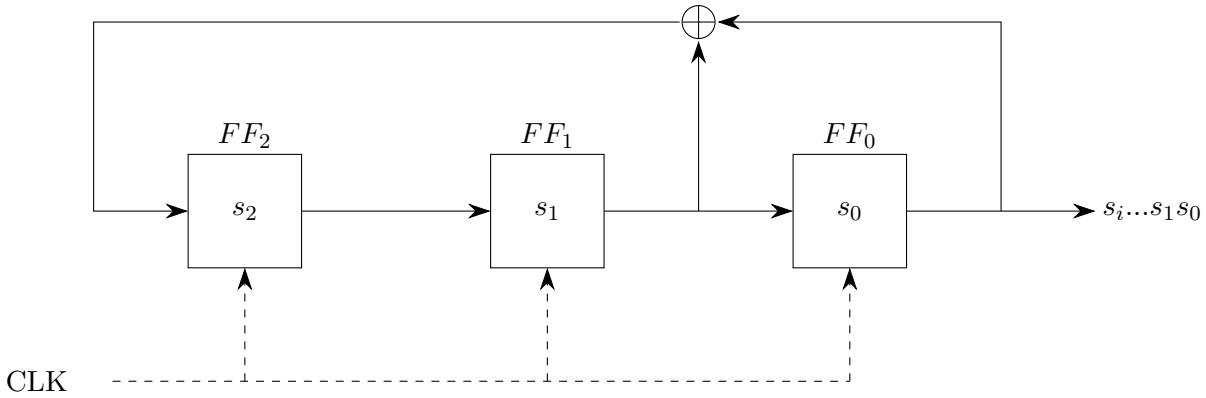
CSPRNGs are a special type of PRNGs which possess the following additional property: a CSPRNG is a PRNG which is **unpredictable**. Formally speaking, this means that given n output bits of the key stream $s_i, s_{i+1}, ..., s_{i+n-1}$, where n is some integer, it is computationally infeasible to compute the subsequent bits $s_{i+n}, s_{i+n+1}, ...$ A more exact definition is that given n consecutive bits of the key stream, there is no polynomial time algorithm that can predict the next bit s_{n+1} with better than 50% chance of success. Another property of CSPRNGs is that given the above sequence, it should be computationally infeasible to compute any preceding bits $s_{i-1}, s_{i-1}, ...$ Note that the need for unpredictability of CSPRNGs is unique to cryptography. Almost all PRNGs that were designed without the clear purpose of being stream ciphers are not CSPRNGs.

1.2 Linear Feedback Shift Registers (LSFR)

Practical stream ciphers use a stream of key bits $s_1, s_2, ...$ that are generated by the key stream generator. An elegant way of realizing long pseudorandom sequences is to use linear feedback shift registers (LFSRs). LFSRs are easily implemented in hardware and many, but certainly not all, stream ciphers make use of LFSRs. Combinations of LFSRs can make secure stream ciphers.

An LFSR consists of clocked storage elements (*flip-flops*) and a *feedback path.* The number of storage elements gives us the **degree** of the LFSR. In other words, an LFSR with m flip-flops is said to be of degree m. The feedback network computes the input for the last flip-flop as a XOR-sum of certain flip-flops in the shift register.

Example) We consider an LFSR of degree $m = 3$ with flip-flops FF_2, FF_1, FF_0, and a feedback path as shown in the following figure:



The internal state bits are denoted by s_i and are shifted by one to the right with each clock tick. The rightmost state bit is also the current output bit. The leftmost state bit is computed in the feedback path, which is the XOR sum of some of the flip-flop values in the previous clock period. Since the XOR is a linear operation, such circuits are called linear feedback shift registers. If we assume an initial state of $(s_2 = 1, s_1 = 0, s_0 = 0)$, the complete sequence of states of the LFSR is the following:

clk	FF_2	FF_1	$FF_0 = s_i$
0	1	0	0
1	0	1	0
2	1	0	1
3	1	1	0
4	1	1	1
5	0	1	1
6	0	0	1
7	1	0	0
8	0	1	0
...	...	...	...

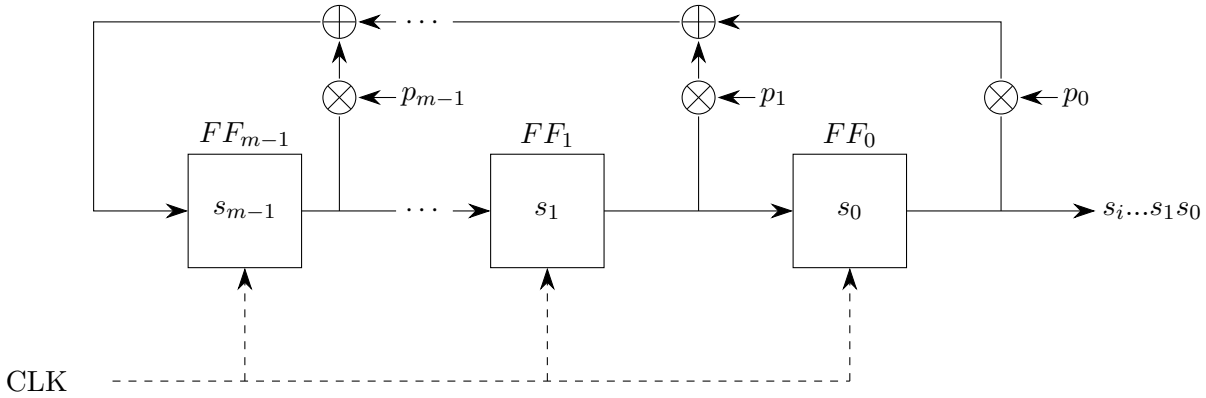
Note that the LSFR starts to repeat after clock cycle 6. This means the LSFR output has period of length 7 and has the form: 0010111 0010111 0010111 .... There is a simple formula which determines the functioning of this LFSR. Let's look at how the output bits s_i are computed, assuming the initial state bits s_0, s_1, s_2:

$$\begin{aligned} s_3 &\equiv s_1 + s_0 \bmod 2 \\ s_4 &\equiv s_2 + s_1 \bmod 2 \\ s_5 &\equiv s_3 + s_2 \bmod 2 \\ &\vdots \\ s_i &\equiv s_{i-2} + s_{i-3} \bmod 2 \end{aligned}$$

where $i = 3, 4, 5, ...$

1.2.1 A Mathematical Description of LFSRs: the Fibonacci LFSR

The general form of an LFSR of degree m is embodied by the **Fibonacci LFSR**, shown in the figure below:



It shows m flip-flops and m possible feedback locations, all combined by the XOR operation. Whether a feedback path is active or not, is defined by the *feedback coefficient* $p_0, p_1, ..., p_{m-1}$:

- if $p_i = 1$ (closed switch), the feedback is active;
- if $p_i = 0$ (open switch), the corresponding flip-flop output is not used for the feedback;

If we multiply the output of FF_i by its coefficient p_i, the result is either the output value if $p_i = 1$ or 0 if $p_i = 0$. The values of the feedback coefficients are crucial for the ouput sequence produced by the LFSR.

Let's assume the LSFR is initially loaded with the values $s_0, ..., s_{m-1}$. The next output bit of the LFSR s_m, which is also the input to the leftmost flip-flop, can be computed by the XOR-sum of the products of flip-flop outputs and corresponding feedback coefficients:

$$s_m \equiv s_{m-1}p_{m-1} + \cdots + s_1 p_1 + s_0 p_0 \mod 2$$

The next LFSR output can be computed as:

$$s_{m+1} \equiv s_m p_{m-1} + \cdots + s_2 p_1 + s_1 p_0 \mod 2$$

In general, the output sequence can be described as:

$$s_{m+i} \equiv \sum_{j=0}^{m-1} s_{j+i} \cdot p_j \mod 2; \qquad s_i, p_j \in \{0, 1\}; \quad i = 0, 1, 2, ...;$$

Clearly, the output values are given through a combination of some previous output values. LFSRs are sometimes referred to as *linear recurrences*.

Due to the finite number of recurring states, the output sequence of an LFSR repeats periodically. Moreover, an LFSR can produce output sequences of different lengths, depending on the feedback coefficients. The following theorem gives us the maximum length of an LFSR as function of its degree: *The maximum sequence length generated by an LFSR of degree m is* $2^m - 1$. Since an m-bit state vector $(p_0 p_1 ... p_{m-1})$ can only assume $2^m - 1$ nonzero states, the maximum sequence length before repetition is $2^m - 1$. Note that the all-zero state must be excluded. If an LFSR assumes this state, it will get "stuck" in it. Note also that only certain configurations of the state vector yield maximum length LFSRs.

An LFSR with a feedback coefficient vector $(p_{m-1}, ..., p_1, p_0)$ is represented by the polynomial:

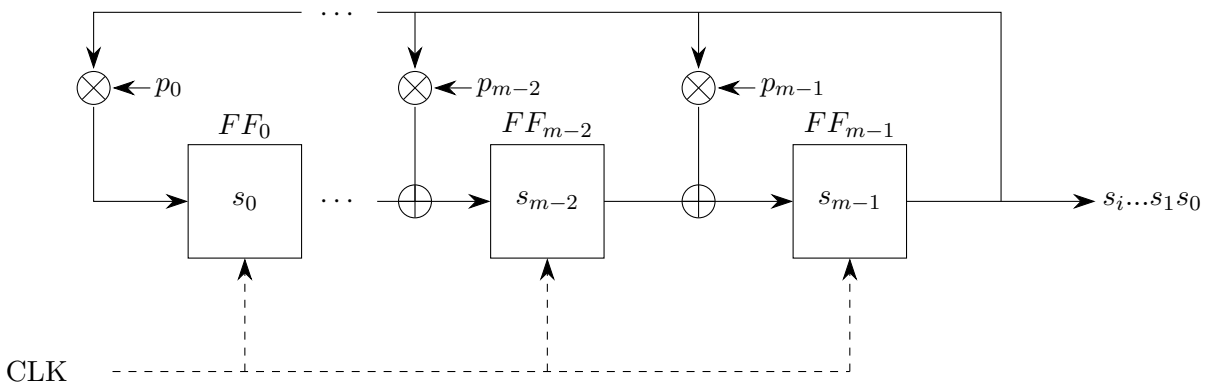
$$P(x) = x^m + p_{m-1}x^{m-1} + \cdots + p_1 x + p_0$$

This notation has several advantages! For instance, maximum-length LFSRs have what is called **primitive polynomials**. Primitive polynomials are a special type of irreducible polynomials which can relatively easily be computed. Hence, maximum-length LFSRs can easily be found. Note that there are many primitive polynomials for every given degree m. For instance, there exist 69.273.666 different primitive polynomials of degree $m = 31$.

1.2.2 Galois LFSRs

It has a structure that can generate the same output stream as a Fibonacci LFSR but offset in time. In the Galois configuration, when the system is clocked:

- if $p_i = 0$, s_{i-1} gets right shifted and stored unchanged into FF_i;
  - special case: for $i = 0$, FF_0 stores 0;
- if $p_i = 1$, s_{i-1} gets right shifted and XORed with the output bit before being stored by FF_i;
  - special case: for $i = 0$, FF_0 stores the output bit;



The computation of the transition state is quite straightforward:

$$\mathbf{s}' = \begin{cases} \text{shiftright}(\mathbf{s}) & \text{if } s_{m-1} = 0 \\ \text{shiftright}(\mathbf{s}) + \mathbf{p} & \text{if } s_{m-1} = 1 \end{cases}$$

where $\mathbf{s} = (s_0, s_1, ..., s_{m-1})$ and $\mathbf{p} = (p_0, p_1, ..., p_{m-1})$. Remember that, according to the previously adopted notation, the + symbol is a XOR-sum (addition modulo 2). Moreover, the state vector can be represented by the polynomial:

$$s(x) = s_{m-1}x^{m-1} + s_{m-2}x^{m-2} + \cdots + s_1 x + s_0$$

Example)

Assuming $s(x) = 1$ and $p(x) = x^2 + x + 1$, compute the next 3 transition states of the Galois LFSR:

$$\mathbf{s} = (s_0, s_1) = (1\ 0); \quad \mathbf{p} = (p_0, p_1) = (1\ 1);$$

Since $s_{m-1} = s_1 = 0$, the next transition state is computed as:

$$\mathbf{s}' = \text{shiftright}(\mathbf{s}) = (0\ 1)$$

Now, since $s_{m-1} = s_1 = 1$, the next transition state is computed as:

$$\mathbf{s}'' = \text{shiftright}(\mathbf{s}') + \mathbf{p} = (0\ 0) + (1\ 1) = (1\ 1)$$

Now, since $s_{m-1} = s_1 = 1$, the next transition state is computed as:

$$\mathbf{s}''' = \text{shiftright}(\mathbf{s}'') + \mathbf{p} = (0\ 1) + (1\ 1) = (1\ 0)$$

The transition state could also be computed by performing a multiplication in $GF(2^m)$ between the polynomials x and $s(x)$, using $p(x)$ to reduce the result of the multiplication (only if the latter's degree is greater than $m-1$) and taking the remainder of the division as the new $s(x)$. Let's consider the same example as before and let's adopt this different (more time consuming, in my honest opinion) approach:

Example) $s(x) = 1$ and $p(x) = x^2 + x + 1$

$$s'(x) = x \cdot s(x) = x \quad \longrightarrow \quad \mathbf{s}' = (s_0, s_1) = (0\ 1);$$

Since the result of the multiplication is a polynomial whose degree is $\leq m-1 = 1$ there's no need to perform the polynomial reduction by $p(x)$. Let's compute the next transition state then:

$$s''(x) = x \cdot s'(x) = x^2$$

Since the degree of the resulting polynomial is $> m-1 = 1$, let's apply the polynomial reduction by $p(x)$:

$+x^2$	$+0x$	$+0$	x^2+x+1
$+x^2$	$+x$	$+1$	1
//	$+x$	$+1$	

The remainder of the polynomial reduction is actually our next transition state:

$$s''(x) = x + 1 \quad \longrightarrow \quad \mathbf{s}'' = (s_0, s_1) = (1\ 1);$$

Let's finally compute our next and last transition state:

$$s'''(x) = x \cdot s''(x) = x^2 + x$$

We have to apply the polynomial reduction by $p(x)$ this time too:

$+x^2$	$+x$	$+0$	x^2+x+1
$+x^2$	$+x$	$+1$	1
//	//	$+1$	

In the end, the last transition state is:

$$s'''(x) = 1 \quad \longrightarrow \quad \mathbf{s}''' = (s_0, s_1) = (1\ 0);$$

1.3 Past exams exercises (RNG)

1.3.1 FacsimileCryptography 2020 ex.1

Compute the bits s_5, s_4, s_3, s_2 of the stream generated by the LFSR whose polynomial is:

$$\chi_{\mathrm{L}}(x) = x^2 + x + 1$$

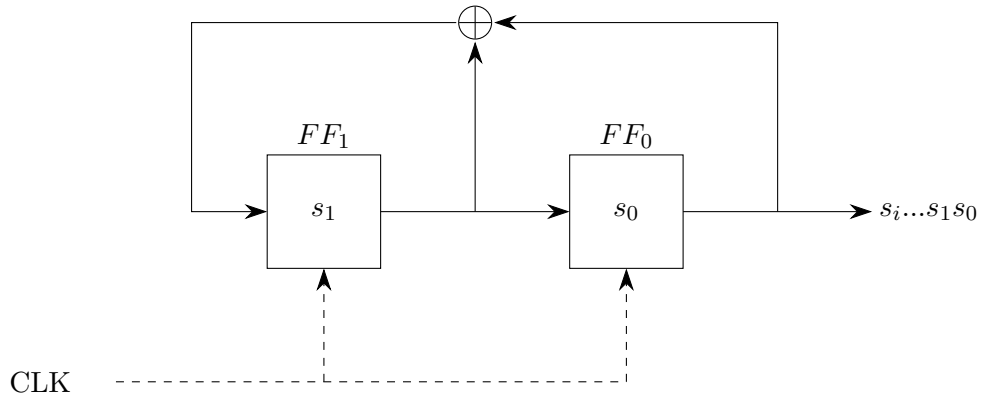
and the first bits are $s_0 = 1, s_1 = 0$.

Solution

First of all, let's "extract" the feedback coefficient vector from the LFSR polynomial:

$$(p_1, p_0) = (1, 1)$$

Since the polynomial degree is $m = 2$, the corresponding LFSR will have 2 flip-flops:



So, we can easily compute the output bits by applying the general formula:

$$s_2 = \sum_{j=0}^{1} s_j \cdot p_j \quad \equiv s_1 p_1 + s_0 p_0 \bmod 2 = 1;$$

$$s_3 = \sum_{j=0}^{1} s_{j+1} \cdot p_j \equiv s_2 p_1 + s_1 p_0 \bmod 2 = 1;$$

$$s_4 = \sum_{j=0}^{1} s_{j+2} \cdot p_j \equiv s_3 p_1 + s_2 p_0 \bmod 2 = 0;$$

$$s_5 = \sum_{j=0}^{1} s_{j+3} \cdot p_j \equiv s_4 p_1 + s_3 p_0 \bmod 2 = 1;$$

Considerations: since the degree of our LFSR is $m = 2$, the maximum length of the output sequence will be $2^m - 1 = 3$. As we've already discussed previously, not all configurations of the feedback coefficient vector $\mathbf{p}$ produce an output sequence with the maximum length. However, if $\mathbf{p}$ is an all-ones vector like in our case, we do obtain an output sequence with the maximum length! That means our output sequence will repeat itself after 3 iterations. As a matter of fact, it's quite useless to compute s_3, s_4, s_5 since we already know that our output sequence is $(s_0 s_1 s_2) = (101)$ and we can immediately conclude that $(s_3 s_4 s_5) = (s_0 s_1 s_2) = (101)$. This holds just for Fibonacci LFSRs.

Alternative solution

Instead of applying the general formula of the LFSRs, we can "build" the Fibonacci LFSR linear map as:

$$\mathrm{L}_{m\times m} = \begin{pmatrix} p_{m-1} & 1 & 0 & 0 & \cdots & 0 \\ p_{m-2} & 0 & 1 & 0 & \cdots & 0 \\ p_{m-3} & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ p_1 & 0 & 0 & 0 & \cdots & 1 \\ p_0 & 0 & 0 & 0 & \cdots & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

where:

- the red submatrix is always the Identity Matrix $\mathbb{1}_{(m-1)\times(m-1)}$;
- the blue subrow is always an all-zero row;

Then, we write the LFSR state vector as:

$$\mathbf{s} = (s_{m-1}, s_{m-2}, ..., s_1, s_0) = (s_1, s_0) = (0\ 1)$$

The transition state is given by the multiplication $\mathbf{s} \cdot \mathrm{L}$:

$$(s_2, s_1) = (s_1, s_0) \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = (0\ 1) \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = (1\ 0) \quad \longrightarrow \quad s_2 = 1$$

$$(s_3, s_2) = (s_2, s_1) \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = (1\ 0) \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = (1\ 1) \quad \longrightarrow \quad s_3 = 1$$

$$(s_4, s_3) = (s_3, s_2) \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = (1\ 1) \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = (0\ 1) \quad \longrightarrow \quad s_4 = 0$$

$$(s_5, s_4) = (s_4, s_3) \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = (0\ 1) \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = (1\ 0) \quad \longrightarrow \quad s_5 = 1$$

WATCH OUT: all computations are always done modulo 2.

Note that we could speed up the process by computing only $\mathbf{s} \cdot \mathbf{p}$:

$$s_2 = (s_1, s_0) \cdot \begin{pmatrix} p_1 \\ p_0 \end{pmatrix} = (0\ 1) \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 1$$

$$s_3 = (s_2, s_1) \cdot \begin{pmatrix} p_1 \\ p_0 \end{pmatrix} = (1\ 0) \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 1$$

$$s_4 = (s_3, s_2) \cdot \begin{pmatrix} p_1 \\ p_0 \end{pmatrix} = (1\ 1) \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 0$$

$$s_5 = (s_4, s_3) \cdot \begin{pmatrix} p_1 \\ p_0 \end{pmatrix} = (0\ 1) \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 1$$

We obviously must update the state vector $\mathbf{s}$ at each iteration of this procedure, in order to extract all the future output bits!

1.3.2 2020-07-21 ex.2 (linear PRNG)

Seed $s_0 = 2$.
$s_1, s_2...$ numbers generated by a linear PRNG with $a = 5, b = 1 \bmod 23$. What is s_2?

Solution

In order to compute s_2 we must compute s_1 first:

$$s_1 = 5s_0 + 1 \bmod 23 = 11$$

$$s_2 = 5s_1 + 1 \bmod 23 = 10$$

1.3.3 2021-07-16 ex.2 (Fibonacci LFSR)

A n-bit Fibonacci LFSR has n flip-flops and produces an output stream $...s_2s_1s_0$.
If a sequence like ...01100001001... is observed in the output stream, then:

a) $n = 4$;

b) $n = 2$;

c) $n = 5$;

d) $n = 3$;

Solution

Since the all-zero state must be excluded, we can say for sure that $n > 4$ because in the output stream there are 4 consecutive zeros. In other words, if we had 4 or less flip-flops, in that moment the LFSR would have assumed the prohibited all-zero state. Having said that, the only answer which satisfies our constraint is (c).

2 Galois Fields (GFs)

2.1 Introduction to groups and fields

Definition A **group** G is a set of elements which can be combined, two by two, by means of an operation which we generically refer to as $\circ$. A group has the following properties:

- The group operation $\circ$ is **closed**. That is, for all $a, b \in G$, it holds that:

$$a \circ b = c \in G$$

- The group operation $\circ$ is **associative**. That is, for all $a, b, c \in G$, it holds that:

$$a \circ (b \circ c) = (a \circ b) \circ c$$

- There is an element $\mathbf{1} \in G$, called the **neutral element**, such that, for all $a \in G$, it's true that:

$$a \circ \mathbf{1} = \mathbf{1} \circ a = a$$

- For each $a \in G$ there exists an element $a^{-1} \in G$, called the **inverse** of a, such that:

$$a \circ a^{-1} = a^{-1} \circ a = \mathbf{1}$$

- A group G is commutative if, furthermore, for all $a, b \in G$, it holds that:

$$a \circ b = b \circ a$$

Roughly speaking, a group is set with one operation and the corresponding inverse operation. If the operation is the addition(multiplication), the inverse operation is the subtraction(division).

Example) The set of integers $\mathbb{Z}_m = \{0, 1, ..., m-1\}$ and the operation "addition modulo m" form a group:

- with the neutral element 0;
- where every element a has an inverse $-a$ such that $a + (-a) = 0 \bmod m$;

Note that this set does not form a group with the operation "multiplication" because most elements a do not have an inverse such that $aa^{-1} = 1 \bmod m$.

Definition A **field** F is a set of elements with the following properties:

- All elements of F form an additive group with the group operation "+" and the neutral element 0;
- All elements of F, except 0, form a multiplicative group with the group operation "$\times$" and the neutral element 1;
- When two group operations are mixed, the distributivity law holds, i.e., for all $a, b, c \in F$:

$$a(b + c) = (ab) + (ac)$$

2.2 Introduction to Galois Fields

In cryptography, we are almost always interested in fields with a finite number of elements, which we call *Finite Fields* or ***Galois Fields***. The number of elements in the field is called the **order** (or cardinality) of the field. Of fundamental importance is the following theorem:

> A field with order m only exists if m is a prime power, i.e., $m = p^n$, for some positive integer n and a prime integer p. p is called the **characteristic** of the Galois Field.

This theorem implies that Galois Fields can only exist with a specific value of order. Some Galois Fields could have $81 = 3^4$ or even $256 = 2^8$ (since 3 and 2 are prime numbers) but there's no Galois Field with $12 = 2^2 \cdot 3$ elements.

2.3 Prime Fields

The most intuitive examples of Galois Fields are fields of prime order, i.e., fields with $n = 1$. Elements of the field $GF(p)$ can be represented by integers $\{0, 1, ..., p-1\}$. The two operations of the field are modular integer addition and integer multiplication modulo p.

Example) Let's consider the Galois Field $GF(5) = \{0, 1, 2, 3, 4\}$. The following tables describe all possible operations within the field itself:

addition

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

additive inverse

$-0 = 0$
$-1 = 4$
$-2 = 3$
$-3 = 2$
$-4 = 1$

multiplication

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

multiplicative inverse

0^{-1} does not exist
$1^{-1} = 1$
$2^{-1} = 3$
$3^{-1} = 2$
$4^{-1} = 4$

- addition: $a + b = c \bmod 5$;
- multiplication: $a \cdot b = c \bmod 5$;
- additive inverse: $a + (-a) = 0 \bmod 5$;
- multiplicative inverse: $a \cdot a^{-1} = 1 \bmod 5$;

WATCH OUT: $-a$ and a^{-1} are just symbols meant to identify, respectively, the additive inverse and the multiplicative inverse of a!

e.g. the additive inverse of 3 (-3) is equal to 2 because $3 + 2 = 0 \bmod 5$
e.g. the multiplicative inverse of 3 $\left(3^{-1}\right)$ is equal to 2 because $3 \cdot 2 = 1 \bmod 5$

A very important prime field is $GF(2)$, which is the smallest existing Galois Field. Let's have a look at the addition and multiplication tables for this field:

add

+	0	1
0	0	1
1	1	0

mult

×	0	1
0	0	0
1	0	1

As we can see, $GF(2)$ addition, i.e. modulo 2 addition, is equivalent to an XOR gate and $GF(2)$ multiplication is equivalent to the logical AND gate. The field $GF(2)$ is important for AES.

2.4 Extension Fields $GF(2^m)$

In AES the finite field contains 256 elements and is denoted as $GF(2^8)$. This field was chosen because each of its elements can be represented by one byte. However, since the order of this field (256) is not prime, the additions and multiplications cannot be performed as additions and multiplications of integers modulo 2^8 (like in Prime Fields). This class of fields $GF(2^m)$ with $m > 1$ are called *extension fields*.

In order to deal with them we need:

- a different notation for field elements;
- different rules for performing arithmetic with such elements;

In extension fields elements are not represented as integers but as polynomials with coefficients in $GF(2)$. The polynomials have a maximum degree of $m-1$, so that there are m coefficients in total for every element. In the field $GF(2^8)$ (used by AES), each element $A \in GF(2^8)$ is represented as:

$$A(x) = a_7x^7 + \cdots + a_1x + a_0, \quad a_i \in GF(2) = \{0, 1\}.$$

Note that there are exactly $256 = 2^8$ such polynomials.

Every polynomial can simply be stored in digital form as an 8-bit vector:

$$A = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$$

2.4.1 Addition and subtraction in $GF(2^m)$

We merely add or subtract coefficients with equal powers of x. The coefficient additions or subtractions are done in the underlying field $GF(2)$:

$$C(x) = A(x) + B(x) = \sum_{i=0}^{m-1} c_ix^i, \quad c_i \equiv a_i + b_i \bmod 2$$

$$C(x) = A(x) - B(x) = \sum_{i=0}^{m-1} c_ix^i, \quad c_i \equiv a_i - b_i \bmod 2$$

Note that addition and subtraction modulo 2 are the same operation! Moreover, addition (subtraction) is equal to bitwise XOR. Let's have a look at the following example...

Example) Let $A(x) = x^7 + x^6 + x^4 + 1$ and $B(x) = x^4 + x^2 + 1$. Compute $C(x) = A(x) + B(x)$:

$$\begin{aligned} A(x) &= x^7 + x^6 + x^4 + \quad 1 \\ B(x) &= \qquad\quad x^4 + x^2 + 1 \\ C(x) &= x^7 + x^6 + \quad x^2 \end{aligned}$$

Note that if we computed the difference $A(x) - B(x)$ we would get the same result as for the sum.

2.4.2 Multiplication in $GF(2^m)$

Multiplying two elements of $GF(2^8)$ is a bit longer than performing addition/subtraction. There are two steps to follow:

1. Multiply the two elements using the standard polynomial multiplication rule:

$$\begin{aligned} A(x)\cdot B(x) &= (a_{m-1}x^{m-1} + \cdots + a_0) \cdot (b_{m-1}x^{m-1} + \cdots + b_0) \\ C'(x) &= c'_{2(m-1)}x^{2(m-1)} + \cdots + c'_0 \end{aligned}$$

   Note that all coefficients $a_i, b_i, c_i \in GF(2)$, and that coefficient arithmetic is performed in $GF(2)$;

2. In general, the product polynomial $C(x)$ will have degree higher than $m-1$ and has to be reduced. So, $C(x)$ is divided by a certain polynomial, and we consider only the reminder after the polynomial division. We need **irreducible polynomials** for the module reduction. Irreducible polynomials are roughly comparable to prime numbers, i.e., their only factors are 1 and the polynomial itself.

   Let $A(x), B(x) \in GF(2^m)$ and let

$$P(x) = \sum_{i=0}^{m} p_i x^i, \quad p_i \in GF(2)$$

   be an irreducible polynomial. Multiplication of the two elements $A(x), B(x)$ is performed as:

$$C(x) = A(x) \cdot B(x) \bmod P(x)$$

For AES, the irreducible polynomial

$$P(x) = x^8 + x^4 + x^3 + x + 1$$

is used. It's part of the AES specification.

Example) We want to multiply $A(x) = x^3 + x^2 + 1$ and $B(x) = x^2 + x$ in the field $GF(2^4)$.
The irreducible polynomial of this Galois Field is given as:

$$P(x) = x^4 + x + 1$$

The plain polynomial product is computed as:

$$\begin{aligned} C'(x) = A(x) \cdot B(x) &= (x^3 + x^2 + 1) \cdot (x^2 + x) \\ &= x^5 + x^4 + x^4 + x^3 + x^2 + x \\ &= x^5 + (2 \bmod 2)x^4 + x^3 + x^2 + x \\ &= x^5 + x^3 + x^2 + x \end{aligned}$$

We can now reduce $C'(x)$ using the polynomial division method. However, sometimes it is easier to reduce each of the leading terms individually. In our case, let's just reduce x^5:

$$x^5 \equiv x^2 + x \bmod P(x)$$

So, by inserting the reduced expression for x^5 into the intermediate result $C'(x)$, we obtain:

$$\begin{aligned} C(x) &\equiv (x^2 + x) + x^3 + x^2 + x = x^3 \\ A(x)\cdot B(x) &\equiv x^3 \end{aligned}$$

WATCH OUT: all coefficient additions so far have been performed in $GF(2)$.

2.4.3 Inversion in $GF(2^m)$

For a given Galois Field $GF(2^m)$ and the corresponding irreducible reduction polynomial $P(x)$, the inverse A^{-1} of a nonzero element $A \in GF(2^m)$ is defined as:

$$A(x) \cdot A^{-1}(x) = 1 \bmod P(x)$$

For small fields (usually with $\leq 2^{16}$ elements) lookup tables which contain the precomputed inverses of all field elements are often used. As an alternative to using lookup tables, the main algorithm for computing multiplicative inverses is the Extended Euclidean Algorithm (EEA).

2.5 Appendix: polynomial division

Let's suppose we have to compute the polynomial division of $A(x)$ by $B(x)$, having that $\deg_A \geq \deg_B$:

$$A(x) = x^4 + x^2 + 1$$

$$B(x) = x + 1$$

Here's how to proceed:

1. Complete $A(x)$ by adding the missing terms with coefficient 0:

$$A(x) = x^4 + 0 \cdot x^3 + x^2 + 0 \cdot x + 1$$

2. Divide the maximum degree term of $A(x)$ by the maximum degree term of $B(x)$:

$+x^4$	$+0x^3$	$+x^2$	$+0x$	$+1$	$x+1$
					x^3

3. Multiply the result you just obtained with $B(x)$ and write down the resulting polynomial by inverting the sign of all of its terms (no need to change sign in $GF(2^m)$ because of the modulo 2 arithmetic). After that, sum this polynomial with $A(x)$:

$+x^4$	$+0x^3$	$+x^2$	$+0x$	$+1$	$x+1$
$-x^4$	$-x^3$				x^3
//	$-x^3$	$+x^2$	$+0x$	$+1$	

4. Repeat from step 1:

$+x^4$	$+0x^3$	$+x^2$	$+0x$	$+1$	$x+1$
$-x^4$	$-x^3$				$x^3 - x^2 + 2x - 2$
//	$-x^3$	$+x^2$	$+0x$	$+1$	
	$+x^3$	$+x^2$			
	//	$+2x^2$	$+0x$	$+1$	
		$-2x^2$	$-2x$		
		//	$-2x$	$+1$	
			$+2x$	$+2$	
			//	$+3$	

5. Since we've obtained a polynomial $(+3)$ whose degree is less than the degree of $B(x)$ we can stop here the division. We can conclude that:

$$A(x) = Q(x) \cdot B(x) + R(x)$$
$$A(x) = (x^3 - x^2 + 2x - 2) \cdot (x + 1) + 3$$

where $Q(x) = x^3 - x^2 + 2x - 2$ is the quotient and $R(x) = 3$ is the remainder of the polynomial division.

3 Extended Euclidean Algorithm (EEA)

3.1 Euclidean Algorithm explanation

The (basic) Euclidean Algorithm itself is used for computing the greatest common divisor of two positive integers r_0 and r_1, denoted by $\gcd(r_0, r_1)$, that is the largest positive number that divides both r_0 and r_1. The algorithm is based on the following equivalence:

$$\gcd(r_0, r_1) = \gcd(r_0 - r_1, r_1)$$

where we assume that $r_0 > r_1$.

Example) Let $r_0 = 84$ and $r_1 = 30$. The gcd can be calculated as follows:

$$\begin{aligned} r_0 - r_1 &= 54 = 2 \cdot 3 \cdot 3 \cdot 3 \\ r_1 &= 30 = 2 \cdot 3 \cdot 5 \end{aligned}$$

The largest common factor between $r_0 - r_1$ and r_1 is $2 \cdot 3 = 6 = \gcd(54, 30) = \gcd(84, 30)$.

The equivalence above can be used iteratively:

$$\gcd(r_0, r_1) = \gcd(r_0 - r_1, r_1) = \gcd(r_0 - 2r_1, r_1) = \cdots = \gcd(r_0 - nr_1, r_1)$$

as long as $(r_0 - nr_1) > 0$. The algorithm uses the fewest number of steps if we choose the maximum value for n. This is the case if we compute:

$$\gcd(r_0, r_1) = \gcd(r_1, r_0 \bmod r_1)$$

where we've swapped the terms inside gcd() because $r_1 > (r_0 \bmod r_1)$.

Example) Let $r_0 = 973$ and $r_1 = 301$. Then:

$$\begin{aligned} \gcd(973, 301) &= \gcd(301, 70) & 973 &= 3 \cdot 301 + 70 \\ \gcd(301, 70) &= \gcd(70, 21) & 301 &= 4 \cdot 70 + 21 \\ \gcd(70, 21) &= \gcd(21, 7) & 70 &= 3 \cdot 21 + 7 \\ \gcd(21, 7) &= \gcd(7, 0) & 21 &= 3 \cdot 7 + 0 \end{aligned}$$

The equivalences in the 2^{nd} column have been written in the form:

$$a = q \cdot m + r \iff a \equiv r \bmod m$$

where $a \in \mathbb{Z}$, q is the quotient, m the modulus and r the remainder.

So, in the end, we obtain:

$$\gcd(937, 301) = \gcd(7, 0) = 7$$

Note that the algorithm stops at the i-th iteration as soon as $r_i = 0$.

3.2 EEA explanation

The main application of the EEA is finding the modular inverse of a number, that is the multiplicative inverse a^{-1} of $a \in \mathbb{Z}_m$:

$$a \cdot a^{-1} \equiv 1 \bmod m \iff a \cdot a^{-1} = q \cdot m + 1$$

The EEA applies the same steps of the Euclidean Algorithm to calculate $\gcd(r_0, r_1)$ but, in addition, it computes a linear combination of the form:

$$\gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1$$

where s and t are integer coefficients.

Let's consider the same example presented in the previous paragraph but, this time, let's calculate s and t too! Let's focus just on the right-side column of the previous example.

Example) Let $r_0 = 973$ and $r_1 = 301$. Then:

$$\begin{array}{llllll}
973 = 3 \cdot 301 + 70 & \longrightarrow & r_2 = 70 = 973 - 3 \cdot 301 & = [1]r_0 + [-3]r_1 \\
301 = 4 \cdot 70 + 21 & \longrightarrow & r_3 = 21 = r_1 - 4 \cdot r_2 & = r_1 - 4(r_0 - 3r_1) = [-4]r_0 + [13]r_1 \\
70 = 3 \cdot 21 + 7 & \longrightarrow & r_4 = 7 = r_2 - 3 \cdot r_3 & = (r_0 - 3r_1) - 3(-4r_0 + 13r_1) = [13]r_0 + [-42]r_1 \\
21 = 3 \cdot 7 + 0 & & &
\end{array}$$

We've successfully written $\gcd(r_0, r_1)$ as a linear combination of r_0 and r_1. The correctness can be verified by:

$$\gcd(973, 301) = [13]973 + [-42]301 = 12649 - 12642 = 7$$

Note that, at each iteration, we have:

$$\boxed{r_i = r_{i-2} - q_i \cdot r_{i-1}}$$

Why is all of this important? Well, let's assume we want to compute the inverse of $r_1 \bmod r_0$, where $r_1 < r_0$. The modular inverse only exists if $\gcd(r_0, r_1) = 1$. If we apply EEA, we obtain $\gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1 = 1$. By taking this equation modulo r_0, we obtain:

$$\begin{aligned}
s \cdot r_0 + t \cdot r_1 &= 1 \\
s \cdot 0 + t \cdot r_1 &\equiv 1 \bmod r_0 \\
r_1 \cdot t &\equiv 1 \bmod r_0
\end{aligned}$$

that is exactly the definition of the inverse of r_1. That means t itself is the inverse of r_1:

$$t = r_1^{-1} \bmod r_0$$

Note that t could also be a negative number! In that case, the modular inverse is:

$$r_1^{-1} = t + r_0$$

Example) Let's compute 12^{-1} mod 67 with the EEA:

$$\begin{aligned}
&\gcd(67,12) = \gcd(12,7) && 67 = 5\cdot 12 + 7\\
&\gcd(12,7) \;\; = \gcd(7,5) && 12 = 1\cdot 7 \;\; + 5\\
&\gcd(7,5) \;\; = \gcd(5,2) && 7 \;\; = 1\cdot 5 \;\; + 2\\
&\gcd(5,2) \;\; = \gcd(2,1) && 5 \;\; = 2\cdot 2 \;\; + 1\\
&\gcd(2,1) \;\; = 1
\end{aligned}$$

Since $\gcd(67,12) = 1$, we know for sure that the modular inverse of 12 mod 67 does exist!

Let's then determine t:

$$\begin{aligned}
r_2 &= 7 = r_0 - 5r_1\\
r_3 &= 5 = r_1 - r_2 \;\; = -r_0 + 6r_1\\
r_4 &= 2 = r_2 - r_3 \;\; = 2r_0 - 11r_1\\
r_5 &= 1 = r_3 - 2r_4 = [-5]r_0 + [28]r_1
\end{aligned}$$

Now that we know $t = 28$, let's verify that $12 \cdot 28 \equiv 1$ mod 67:

$$12 \cdot 28 = 336 = 5 \cdot 67 + 1$$

So, in the end, it's true that $28 = 12^{-1}$ mod 67 ✓

3.3 EEA application in Galois Fields

The EEA can be used completely analogously with polynomials instead of integers. If we want to compute an inverse in a Galois Field $GF(2^m)$, the inputs to the algorithm are the field element $A(x)$ and the irreducible polynomial $P(x)$. The EEA computes the auxiliary polynomials $s(x)$ and $t(x)$, as well as $\gcd(P(x), A(x))$ such that:

$$s(x)P(x) + t(x)A(x) = \gcd(P(x), A(x)) = 1$$

Note that since $P(x)$ is irreducible, the gcd is always equal to 1. If we take the equation above and reduce both sides modulo $P(x)$, it is straightforward to see that the auxiliary polynomial $t(x)$ is equal to the inverse of $A(x)$:

$$\begin{aligned}
s(x)0 + t(x)A(x) &\equiv 1 \bmod P(x)\\
t(x) \;\; &\equiv A^{-1}(x) \bmod P(x)
\end{aligned}$$

Example) Compute the inverse of $A(x) = x^2$ in $GF(2^3)$ with $P(x) = x^3 + x + 1$:

$$\begin{aligned}
x^3 + x + 1 &= [x]\cdot x^2 + [x+1] && r_2 = r_0 - q_2 r_1 = r_0 - xr_1\\
x^2 \qquad\quad &= [x-1](x+1) + [1] && r_3 = r_1 - q_3 r_2 = r_1 - (x-1)(r_0 - xr_1) = [1-x]r_0 + [x^2 - x + 1]r_1\\
x + 1 \qquad &= [x+1]\cdot 1 + [0] && \text{Termination since } r_4 = 0
\end{aligned}$$

where, by using the same notation we've been using until now, we have that:

- $r_0 = P(x)$ and $r_1 = A(x)$;
- $q_2 = x$ and $r_2 = x + 1$;
- $q_3 = x - 1$ and $r_3 = 1$

So, we obtain $t(x) = x^2 - x + 1 \equiv x^2 + x + 1 \equiv A^{-1}(x)$ mod $P(x)$. Note that polynomial coefficients are computed in $GF(2)$, and since addition and subtraction are the same operations, we can always replace a negative coefficient (such as $-x$) with a positive one.

3.4 EEA by means of Gauss-Jordan method

The Gauss-Jordan method is used to compute the inverse of a matrix A. However, it turns out to be a great method to compute the inverse of a number modulo m! Let's recall how the method works:

- In order to compute A^{-1} we must transform A into an identity matrix $\mathbb{1}$ with the same dimensions of A. To do so, we're allowed to perform the following operations on A:
  - Row swapping (not useful in our case);
  - Replacement of a row with the sum of itself and the multiple of another row;
  - Multiplication of a row by a non-zero constant (not useful in our case);

So, let's see how the Gauss-Jordan method can be applied to our problem. We have to compute the inverse of x modulo m, that is a number x^{-1} such that:

$$x \cdot x^{-1} \equiv 1 \bmod m$$

We define the matrix A and the identity matrix as:

$$A = \begin{pmatrix} x \\ m \end{pmatrix} \qquad\qquad \mathbb{1} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Our goal is to transform A into $\mathbb{1}$ by means of the operations listed above. Let's have a look at an example:

Example) Compute the inverse of 29 modulo 45.

First of all, let's put A and $\mathbb{1}$ next to each other:

$$\left(\begin{array}{c|c} 29 & 1 \\ 45 & 0 \end{array}\right)$$

Now, let's apply a series of operations to the rows of the matrix so that the first half of it becomes the identity matrix $\mathbb{1}$. We use the notation r_i for the i-th row of the matrix.

$$\left(\begin{array}{c|c} 29 & 1 \\ 45 & 0 \end{array}\right) \xrightarrow{r_2 = r_2 - r_1} \left(\begin{array}{c|c} 29 & 1 \\ 16 & -1 \end{array}\right) \xrightarrow{r_1 = r_1 - r_2} \left(\begin{array}{c|c} 13 & 2 \\ 16 & -1 \end{array}\right) \xrightarrow{r_2 = r_2 - r_1}$$

$$\xrightarrow{r_2 = r_2 - r_1} \left(\begin{array}{c|c} 13 & 2 \\ 3 & -3 \end{array}\right) \xrightarrow{r_1 = r_1 - 4r_2} \left(\begin{array}{c|c} 1 & 14 \\ 3 & -3 \end{array}\right) \xrightarrow{r_2 = r_2 - 3r_1} \left(\begin{array}{c|c} 1 & 14 \\ 0 & -45 \end{array}\right)$$

The one in red is the result we're looking for:

$$29^{-1} \equiv 14 \bmod 45$$

3.5 Past exams exercises (EEA)

3.5.1 2020-07-21 ex.3

Find $x \in \mathbb{Z}_{401}$ such that:

$$\begin{cases} x \cdot 262 = 1 \bmod 401 \\ 5 \cdot x = 375 \bmod 401 \end{cases}$$

Solution

The 1st equation can only be solved for $x = 262^{-1} \bmod 401$. Let's apply EEA ($r_0 = 401, r_1 = 262$):

$$\begin{array}{lll} \gcd(401, 262) = \gcd(262, 139) & & 401 = 1 \cdot 262 + 139 \\ \gcd(262, 139) = \gcd(139, 123) & & 262 = 1 \cdot 139 + 123 \\ \gcd(139, 123) = \gcd(123, 16) & & 139 = 1 \cdot 123 + 16 \\ \gcd(123, 16) = \gcd(16, 11) & & 123 = 7 \cdot 16 + 11 \\ \gcd(16, 11) = \gcd(11, 5) & & 16 = 1 \cdot 11 + 5 \\ \gcd(11, 5) = \gcd(5, 1) & & 11 = 2 \cdot 5 + 1 \\ \gcd(5, 1) = 1 & & \end{array}$$

The modular inverse does exist:

$$\begin{array}{lll} r_2 = 139 = r_0 - r_1 & & \\ r_3 = 123 = r_1 - r_2 & = r_0 + 2r_1 & \\ r_4 = 16 = r_2 - r_3 & = -3r_1 & \\ r_5 = 11 = r_3 - 7r_4 & = r_0 + 23r_1 & \\ r_6 = 5 = r_4 - r_5 & = -r_0 - 26r_1 & \\ r_7 = 1 = r_5 - 2r_6 & = 3r_0 + [75]r_1 & \end{array}$$

and it's $t = 75$:

$$75 \cdot 262 = 19650 = 49 \cdot 401 + 1 \checkmark$$

As we can see, the 2nd equation is satisfied too:

$$5 \cdot 75 = 375 = 0 \cdot 401 + 375 \checkmark$$

Solution with Gauss-Jordan method

We just show the inverse computation:

$$\left(\begin{array}{c|c} 262 & 1 \\ 401 & 0 \end{array}\right) \xrightarrow{r_2 = r_2 - r_1} \left(\begin{array}{c|c} 262 & 1 \\ 139 & -1 \end{array}\right) \xrightarrow{r_1 = r_1 - r_2} \left(\begin{array}{c|c} 123 & 2 \\ 139 & -1 \end{array}\right) \xrightarrow{r_2 = r_2 - r_1}$$

$$\xrightarrow{r_2 = r_2 - r_1} \left(\begin{array}{c|c} 123 & 2 \\ 16 & -3 \end{array}\right) \xrightarrow{r_1 = r_1 - 7r_2} \left(\begin{array}{c|c} 11 & 23 \\ 16 & -3 \end{array}\right) \xrightarrow{r_2 = r_2 - r_1} \left(\begin{array}{c|c} 11 & 23 \\ 5 & -26 \end{array}\right) \xrightarrow{r_1 = r_1 - 2r_2}$$

$$\xrightarrow{r_1 = r_1 - 2r_2} \left(\begin{array}{c|c} 1 & 75 \\ 5 & -26 \end{array}\right) \xrightarrow{r_2 = r_2 - 5r_1} \left(\begin{array}{c|c} 1 & 75 \\ 0 & -401 \end{array}\right)$$

3.5.2 2021-07-02 ex.3 v.1

Find $x \in \mathbb{Z}_{401}$ such that:

$$\begin{cases} x \cdot 56 = 1 \bmod 401 \\ 5 \cdot x = 308 \bmod 401 \end{cases}$$

Solution

The 1st equation can only be solved for $x = 56^{-1} \bmod 401$. Let's apply EEA ($r_0 = 401, r_1 = 56$):

$$\begin{array}{lll} \gcd(401, 56) = \gcd(56, 9) & & 401 = 7 \cdot 56 + 9 \\ \gcd(56, 9) = \gcd(9, 2) & & 56 = 6 \cdot 9 + 2 \\ \gcd(9, 2) = \gcd(2, 1) & & 9 = 4 \cdot 2 + 1 \\ \gcd(2, 1) = 1 & & \end{array}$$

The modular inverse does exist:

$$\begin{aligned} r_2 &= 9 = r_0 - 7r_1 \\ r_3 &= 2 = r_1 - 6r_2 = -6r_0 + 43r_1 \\ r_4 &= 1 = r_2 - 4r_3 = 25r_0 + [-179]r_1 \end{aligned}$$

Since $t = -179 < 0$, the modular inverse is $56^{-1} = t + r_0 = -179 + 401 = 222$:

$$222 \cdot 56 = 12432 = 31 \cdot 401 + 1 \checkmark$$

As we can see, the 2nd equation is satisfied too:

$$5 \cdot 222 = 1110 = 2 \cdot 401 + 308 \checkmark$$

Solution with Gauss-Jordan method

We just show the inverse computation:

$$\left(\begin{array}{c|c} 56 & 1 \\ 401 & 0 \end{array}\right) \xrightarrow{r_2 = r_2 - 7r_1} \left(\begin{array}{c|c} 56 & 1 \\ 9 & -7 \end{array}\right) \xrightarrow{r_1 = r_1 - 6r_2} \left(\begin{array}{c|c} 2 & 43 \\ 9 & -7 \end{array}\right) \xrightarrow{r_2 = r_2 - 4r_1}$$

$$\xrightarrow{r_2 = r_2 - 4r_1} \left(\begin{array}{c|c} 2 & 43 \\ 1 & -179 \end{array}\right) \xrightarrow{r_1 = r_1 - r_2} \left(\begin{array}{c|c} 1 & 222 \\ 1 & -179 \end{array}\right) \xrightarrow{r_2 = r_2 - r_1} \left(\begin{array}{c|c} 1 & 222 \\ 0 & -401 \end{array}\right)$$

3.5.3 2021-07-02 ex.3 v.2

Find $x \in \mathbb{Z}_{401}$ such that:

$$\begin{cases} x \cdot 29 = 1 \bmod 401 \\ 5 \cdot x = 14 \bmod 401 \end{cases}$$

Solution

The 1st equation can only be solved for $x = 29^{-1} \bmod 401$. Let's apply EEA ($r_0 = 401, r_1 = 29$):

$$\begin{array}{lll} \gcd(401, 29) = \gcd(29, 24) & & 401 = 13 \cdot 29 + 24 \\ \gcd(29, 24) = \gcd(24, 5) & & 29 = 1 \cdot 24 + 5 \\ \gcd(24, 5) = \gcd(5, 4) & & 24 = 4 \cdot 5 + 4 \\ \gcd(5, 4) = \gcd(4, 1) & & 5 = 1 \cdot 4 + 1 \\ \gcd(4, 1) = 1 & & \end{array}$$

The modular inverse does exist:

$$\begin{array}{llll} r_2 = 24 & = r_0 - 13r_1 & & \\ r_3 = 5 & = r_1 - r_2 & = -r_0 + 14r_1 & \\ r_4 = 4 & = r_2 - 4r_3 & = 5r_0 - 69r_1 & \\ r_5 = 1 & = r_3 - r_4 & = -6r_0 + [83]r_1 & \end{array}$$

and it's $t = 83$:

$$83 \cdot 29 = 2407 = 6 \cdot 401 + 1 \checkmark$$

As we can see, the 2nd equation is satisfied too:

$$5 \cdot 83 = 415 = 1 \cdot 401 + 14 \checkmark$$

Solution with Gauss-Jordan method

We just show the inverse computation:

$$\left(\begin{array}{c|c} 29 & 1 \\ 401 & 0 \end{array}\right) \xrightarrow{r_2 = r_2 - 13r_1} \left(\begin{array}{c|c} 29 & 1 \\ 24 & \text{-}13 \end{array}\right) \xrightarrow{r_1 = r_1 - r_2} \left(\begin{array}{c|c} 5 & 14 \\ 24 & \text{-}13 \end{array}\right) \xrightarrow{r_2 = r_2 - 4r_1}$$

$$\xrightarrow{r_2 = r_2 - 4r_1} \left(\begin{array}{c|c} 5 & 14 \\ 4 & \text{-}69 \end{array}\right) \xrightarrow{r_1 = r_1 - r_2} \left(\begin{array}{c|c} 1 & 83 \\ 4 & \text{-}69 \end{array}\right) \xrightarrow{r_2 = r_2 - 4r_1} \left(\begin{array}{c|c} 1 & 83 \\ 0 & \text{-}401 \end{array}\right)$$

3.6 Exercises from slides (EEA)

3.6.1 Exercise 6.2.2

Set $N = 16^{30} - 1$. Compute x such that $2x \equiv 1 \bmod N$.

Solution with Gauss-Jordan method

First of all, let's express N in a different way:

$$N = 2^{120} - 1$$

Now, let's apply the Gauss-Jordan method:

$$\left(\begin{array}{c|c} 2 & 1 \\ 2^{120}-1 & 0 \end{array}\right) \xrightarrow{r_2=r_2-2^{119}r_1} \left(\begin{array}{c|c} 2 & 1 \\ -1 & -2^{119} \end{array}\right) \xrightarrow{r_1=r_1+r_2}$$

$$\xrightarrow{r_1=r_1+r_2} \left(\begin{array}{c|c} 1 & 1-2^{119} \\ -1 & -2^{119} \end{array}\right) \xrightarrow{r_2=r_2+r_1} \left(\begin{array}{c|c} 1 & 1-2^{119} \\ 0 & 1-2^{120} \end{array}\right)$$

Since we've obtained a negative result, let's just add N to it as many times as necessary in order to make it positive (just once, in this case):

$$res = 1 - 2^{119} + N = 1 - 2^{119} + 2^{120} - 1 = 2^{120} - 2^{119} = 2 \cdot 2^{119} - 2^{119} = 2^{119}$$

In the end, we obtain:

$$x = 2^{-1} \equiv 2^{119} \bmod N$$

3.7 Past exams exercises (EEA in GFs)

3.7.1 2021-07-02 ex.2 v.1

Let $GF(8)$ be the Galois Field defined by the polynomial $G(x) = x^3 + x + 1 \in \mathbb{Z}_2[x]$. Let $a(x) \in GF(8)$ be $a(x) = x + 1$. What is the multiplicative inverse of $a(x)$?

Solution

We start dividing $G(x)$ by $a(x)$ (1st iteration of the EEA):

$+x^3$	$+0x^2$	$+x$	$+1$	$x+1$
$-x^3$	$-x^2$			$x^2 - x + 2$
//	$-x^2$	$+x$	$+1$	
	$+x^2$	$+x$		
	//	$+2x$	$+1$	
		$-2x$	-2	
		//	-1	

or, in short:

$$x^3 + x + 1 = [x^2 - x + 2] \cdot (x + 1) + [-1]$$

where $q_2 = x^2 - x + 2$ and $r_2 = -1$.

The 2nd iteration of the EEA already returns a reminder equal to 0:

$$x + 1 = [-x - 1] \cdot (-1) + [0]$$

So, we've already obtained $t(x)$ from the 1st iteration of the algorithm:

$$r_2 = r_0 - q_2 r_1 = G(x) - [x^2 - x + 2]a(x)$$

Since all polynomial coefficients are calculated in $GF(2)$, $t(x)$ becomes:

$$t(x) = x^2 + x \equiv a^{-1}(x) \bmod G(x)$$

Faster Solution

The trick consists in computing polynomial coefficients in $GF(2)$ while performing the polynomial division! Since we're doing that, we can avoid inverting the sign of the coefficients which we obtain by multiplying $a(x)$ with the quotient addends (that was what we were doing when we recalled polynomial divisions):

$+x^3$	$+0x^2$	$+x$	$+1$	$x+1$
$+x^3$	$+x^2$			$x^2 + x$
//	$+x^2$	$+x$	$+1$	
	$+x^2$	$+x$		
	//	//	$+1$	

Whenever the remainder of the polynomial division is either 1 or -1 we can stop the algorithm, since the next iteration will return a remainder equal to 0 for sure. Moreover, if we already obtain a remainder equal to 1 (-1) during the 1st iteration, we can conclude that the quotient of the division is the inverse of $a(x)$:

$$r_2 = 1 = r_0 - [x^2 + x]r_1 = G(x) - [x^2 + x]a(x)$$

3.7.2 2021-07-02 ex.2 v.2

Let $GF(8)$ be the Galois Field defined by the polynomial $G(x) = x^3 + x + 1 \in \mathbb{Z}_2[x]$. Let $a(x) \in GF(8)$ be $a(x) = x^2 + x$. What is the multiplicative inverse of $a(x)$?

Solution

Just like in the previous exercise:

$$\begin{array}{cccc|c} +x^3 & +0x^2 & +x & +1 & x^2+x \\ +x^3 & +x^2 & & & \hline x+1 \\ \hline // & +x^2 & +x & +1 & \\ & +x^2 & +x & & \\ \hline & // & // & +1 & \end{array}$$

$$r_2 = 1 = r_0 - [x+1]r_1 = G(x) - [x+1]a(x)$$

4 Chinese Reminder Theorem (CRT)

4.1 CRT explanation

Let N be an integer such that:

$$N = \prod_{i=1}^{k} n_i \quad , \quad n_i > 1$$

where $n_1, ..., n_k$ are pairwise coprime integers, that is $\gcd(n_i, n_j) = 1$ for $1 \leq i < k$, $1 \leq j < k$ and $i \neq j$. In other words, $n_i, ..., n_k$ are pairwise coprime divisors of N and they're called **moduli**.

Then, if $a_1, ..., a_k$ are integers such that $0 \leq a_i < n_i \ \forall i$, there is one and only one integer x, such that $0 \leq x < N$ and the remainder of the Euclidean division of x by n_i is a_i for every i.

So, the system of equations

$$\begin{cases} x \equiv a_1 \bmod n_1 \\ \quad \vdots \\ x \equiv a_i \bmod n_i \end{cases}$$

has a solution, and any pair of solutions x_i, x_j are equivalent modulo N, that is, $x_i \equiv x_j \bmod N$.

Example) Suppose $N = 24 = 3 \cdot 8$, where $n_1 = 3,\ n_2 = 8$. By choosing all possible values of $0 \leq x < N$, that are basically the elements of the ring $\mathbb{Z}_N$ $\{0, ..., 23\}$, we can find all the corresponding tuples (a_1, a_2) with $0 \leq a_1 < n_1$ and $0 \leq a_2 < n_2$ thanks to the formula:

$$x \equiv (a_1, a_2) = (x \bmod n_1,\ x \bmod n_2)$$

So, we can easily obtain the following map:

0	(0, 0)
1	(1, 1)
2	(2, 2)
3	(0, 3)
4	(1, 4)
5	(2, 5)
6	(0, 6)
7	(1, 7)
8	(2, 0)
9	(0, 1)
10	(1, 2)
11	(2, 3)
12	(0, 4)
13	(1, 5)
14	(2, 6)
15	(0, 7)
16	(1, 0)
17	(2, 1)
18	(0, 2)
19	(1, 3)
20	(2, 4)
21	(0, 5)
22	(1, 6)
23	(2, 7)

The mapping $x \leftrightarrow (a_1, ..., a_k)$ is bijective because it's possible to perform the inverse map operation starting from the a_i tuples (by knowing $n_1, ..., n_k$) even if that's not so straightforward.

Another formulation of the CRT is actually more interesting for us: if the n_i are pairwise coprime, the map

$$x \bmod N \mapsto (x \bmod n_1, ..., x \bmod n_k)$$

defines a **ring isomorphism**

$$\mathbb{Z}_N \approx \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k}$$

between the ring of integers modulo N and the **direct product** of the rings of integers modulo the n_i. This means that for doing a sequence of arithmetic operations in $\mathbb{Z}_N$ one may do the same computation independently in each $\mathbb{Z}_{n_i}$ and then get the result by applying the isomorphism (from the right to the left). This may be much faster than the direct computation if N and the number of operations are large.

In order to understand this statement, let's introduce the following concepts:

- A **direct product of rings** is a ring that is formed by the Cartesian product of the underlying sets of several rings (possibly an infinity), equipped with componentwise operations:
  - if we think of $\mathbb{R}$ as the ring of real numbers, then the direct product $\mathbb{R} \times \mathbb{R}$ has $\{(x, y) : x, y \in \mathbb{R}\}$ as its underlying set. The ring structure consists of addition by $(a, b) + (c, d) = (a + c, b + d)$ and multiplication defined by $(a, b)(c, d) = (ac, bd)$;

  Since direct products are defined up to an isomorphism, one says colloquially that a ring is the product of some rings if it is isomorphic to the direct product of these rings:

  - an important example is the ring of integers modulo N, $\mathbb{Z}_N$. If N is written as a product of prime powers $N = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$, where p_i are distinct primes, then $\mathbb{Z}_N$ is naturally isomorphic to the product

    $$\mathbb{Z}_{p_1^{n_1}} \times \mathbb{Z}_{p_2^{n_2}} \times \cdots \times \mathbb{Z}_{p_k^{n_k}}.$$

    This follows from the CTR;

  When a ring is isomorphic to the direct product of some other rings, it means that it's equivalent to (so it behaves like) that direct product!

- In ring theory, a **ring homomorphism** is a structure-preserving function between two rings. More explicitly, if R and S are rings, then a ring homomorphism is a function $f : R \to S$ such that f is:
  - addition preserving:

    $$f(a + b) = f(a) + f(b) \ \forall a, b \in R$$

  - multiplication preserving:

    $$f(ab) = f(a)f(b) \ \forall a, b \in R$$

  - unit (multiplicative identity) preserving:

    $$f(\mathbf{1}_R) = \mathbf{1}_S$$

  Additive inverses and the additive identity are part of the structure too, but it is not necessary to require exactly that they too are respected, because these conditions are consequences of the three conditions above.

  If in addition f is a bijection, then its inverse f^{-1} is also a ring homomorphism. In this case, f is called a **ring isomorphism**, and the rings R and S are called *isomorphic*. From the standpoint of ring theory, isomorphic rings cannot be distinguished.

Example) Supposing that $(a_1, a_2) = (2, 7)$ and $(n_1, n_2) = (3, 8)$, x can be calculated by solving the following system of equations:

$$\begin{cases} x \equiv 2 \bmod 3 \\ x \equiv 7 \bmod 8 \end{cases}$$

First of all, since 3 and 8 are coprime, we know for sure that the system has a solution. In order to find the smallest possible solution we must search within the ring $\mathbb{Z}_{3\times 8} = \mathbb{Z}_{24} = \{0, ..., 23\}$.

Strategy 1: Systematic search (simple but could be time consuming)

In order to do that, we can simply follow this strategy:

1. Choose the modular equation with the greatest modulo `n`: $x \equiv 7 \bmod 8$;
2. Starting from the reminder of that equation `a`, try to verify all the other equations of the system by supposing `x=a` ($x = 7$). At each "iteration":
   - if the current value of `x` solves the system $\rightarrow$ STOP;
   - else `x+=n`;
3. The first value which verifies all the equations is the system solution: $x = 23$;

In our case we would have to try just three different values of x before finding the system solution: $\{7, 15, 23\}$. However, this could become much more time consuming depending from the number of the equations to be verified and the order (cardinality) of the ring $\mathbb{Z}_N = \{0, ..., N - 1\}$.

For example, let's suppose this time that our starting point is the following system of equations:

$$\begin{cases} x \equiv 2 \bmod 3 \\ x \equiv 7 \bmod 8 \\ x \equiv 4 \bmod 5 \end{cases}$$

Since all the possible pairs (n_i, n_j) with $i \neq j$ are coprime, a solution does exist. Let's try to apply our previous strategy:

$$N = 120 \quad \longrightarrow \quad \mathbb{Z}_N = \{0, ..., 119\}$$

Let's try already with the previous system solution (3 iterations), that is $x = 23$. However, since $23 \equiv 3 \bmod 5$, 23 is not a solution of the system. Let's keep trying with other possible values:

$$x = \{7, 15, 23, 31, 39, 47, 55, 63, 71, 79, 87, 95, 103, 111, 119\}$$

We obtain that $x = 119$ after 15 iterations. It is clear that doing this computations by hand is really time consuming! If we add also the equation $x \equiv 6 \bmod 7$, we obtain $x = 839$ after 105 iterations...

Even though we've always considered the worst cases, where $a_i = n_i - 1$ and the algorithm requires the maximum number of iterations, this strategy on average takes too much time to be applied. As a matter of fact, the time complexity of this algorithm is exponential!

A little trick

Assuming that the CRT hypothesis are verified:

- if $a_i = 0 \ \forall i \rightarrow x = 0$;
- if $a_i = b \ \forall i \rightarrow x = b$;
- if $a_i = n_i - 1 \ \forall i \rightarrow x = N - 1$;

Strategy 2: Using the ring isomorphism (faster thanks to EEA)

This strategy takes advantage of the isomorphism:

$$\mathbb{Z}_N \approx \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k}$$

in order to operate on the single rings $\mathbb{Z}_{n_i}$ (which is generally easier, especially when N is big) before reverting the transformation to obtain the result in $\mathbb{Z}_N$.

Let's assume $N = 24$, so $\mathbb{Z}_{24} \approx \mathbb{Z}_3 \times \mathbb{Z}_8$ and the mapping

$$x \bmod N \mapsto (x \bmod 3, x \bmod 8) = (a, b)$$

is the function f which defines the isomorphism between $\mathbb{Z}_{24}$ and the direct product of the rings $\mathbb{Z}_3$ and $\mathbb{Z}_8$.

Starting from the general tuple (a, b) we can make the following considerations thanks to the properties of the isomorphism:

$$f(a,b) = f\left(a(1,0) + b(0,1)\right) = f(a(1,0)) + f(b(0,1)) = a \cdot f(1,0) + b \cdot f(0,1)$$

Let's then compute $f(1,0)$ and $f(0,1)$. We introduce some "support variables" just to use a simpler notation:

$$y = f(1,0) \longrightarrow \begin{cases} y \equiv 1 \bmod 3 \\ y \equiv 0 \bmod 8 \end{cases} \longrightarrow \begin{cases} 8q \equiv 1 \bmod 3 \\ y = 8q \end{cases}$$

In order to compute y we have to compute the inverse of 8 modulo 3, that is q. To do that easily, we can generally use the Extended Euclidean Algorithm but in this case it's quite straightforward that $q = 2$ and so $f(1,0) = 8q = 16$. We do the same for $f(0,1)$:

$$y = f(0,1) \longrightarrow \begin{cases} y \equiv 0 \bmod 3 \\ y \equiv 1 \bmod 8 \end{cases} \longrightarrow \begin{cases} y = 3q \\ 3q \equiv 1 \bmod 8 \end{cases}$$

Even in this case it's quite straightforward that $q = 3$ and so $f(0,1) = 3q = 9$.

We can now compute the final result as:

$$f(a,b) = a \cdot f(1,0) + b \cdot f(0,1) = (16a + 9b) \bmod N$$

We've obtained a formula to reverse the mapping defined by the function f. Now, starting from whatever tuple (a, b) such that a and b satisfy the CTR hypothesis ($0 \leq a < 3$ and $0 \leq b < 8$) we can easily calculate the corresponding value of x. For example, if we choose the tuple (1,3) we obtain:

$$x = f(1,3) = (16 + 27) \bmod N = 43 \bmod 24 = 19$$

Let's consider again the case with $N = 840 = 3 \cdot 5 \cdot 7 \cdot 8$ which was really time consuming with the systematic search algorithm. In this case, let's compute:

$$f(a,b,c,d) = a \cdot f(1,0,0,0) + b \cdot f(0,1,0,0) + c \cdot f(0,0,1,0) + d \cdot f(0,0,0,1)$$

$$y = f(1,0,0,0) \longrightarrow \begin{cases} y \equiv 1 \bmod 3 \\ y \equiv 0 \bmod 5 \\ y \equiv 0 \bmod 7 \\ y \equiv 0 \bmod 8 \end{cases} \longrightarrow \begin{cases} y \equiv 1 \bmod 3 \\ y = 5q_1 \\ y = 7q_2 \\ y = 8q_3 \end{cases} \longrightarrow y = 280$$

The only possible value(s) of y which satisfies the last three equations is the m.c.m. (minimum common multiple) of the coprime factors 5,7,8 (and all its multiples) which is easily calculated by computing their product. So, we obtain $y = 280$ which luckily verifies also the first equation (since $280 = 3 \cdot 93 + 1$).

Let's do the same for the other values:

$$y = f(0,1,0,0) \longrightarrow \begin{cases} y \equiv 0 \bmod 3 \\ y \equiv 1 \bmod 5 \\ y \equiv 0 \bmod 7 \\ y \equiv 0 \bmod 8 \end{cases} \longrightarrow \begin{cases} y = 3q_1 \\ y \equiv 1 \bmod 5 \\ y = 7q_2 \\ y = 8q_3 \end{cases} \longrightarrow y = 168 \cdot 2 = 336$$

$$y = f(0,0,1,0) \longrightarrow \begin{cases} y \equiv 0 \bmod 3 \\ y \equiv 0 \bmod 5 \\ y \equiv 1 \bmod 7 \\ y \equiv 0 \bmod 8 \end{cases} \longrightarrow \begin{cases} y = 3q_1 \\ y = 5q_2 \\ y \equiv 1 \bmod 7 \\ y = 8q_3 \end{cases} \longrightarrow y = 120$$

$$y = f(0,0,0,1) \longrightarrow \begin{cases} y \equiv 0 \bmod 3 \\ y \equiv 0 \bmod 5 \\ y \equiv 0 \bmod 7 \\ y \equiv 1 \bmod 8 \end{cases} \longrightarrow \begin{cases} y = 3q_1 \\ y = 5q_2 \\ y = 7q_3 \\ y \equiv 1 \bmod 8 \end{cases} \longrightarrow y = 105$$

In the end we obtain:

$$f(a,b,c,d) = (280a + 336b + 120c + 105d) \bmod N$$

$$f(2,4,6,7) = (560 + 1344 + 720 + 735) \bmod 840 = 3359 \bmod 840 = 839$$

4.2 Past exams exercises (isomorphisms)

4.2.1 2020-07-03 ex.2 v.1/2022-06-27 ex.2

Let $f : \mathbb{Z}_3 \times \mathbb{Z}_5 \mapsto \mathbb{Z}_{15}$ be the isomorphism of CRT. Calculate $f(a, b)$.

Solution

As we've already seen in the previous examples, by using the isomorphisms properties, we obtain:

$$f(a, b) = a \cdot f(1, 0) + b \cdot f(0, 1)$$

$$y = f(1, 0) \longrightarrow \begin{cases} y \equiv 1 \bmod 3 \\ y \equiv 0 \bmod 5 \end{cases} \longrightarrow \begin{cases} 5q \equiv 1 \bmod 3 \\ y = 5q \end{cases} \longrightarrow q = 2 \longrightarrow y = 10$$

$$y = f(0, 1) \longrightarrow \begin{cases} y \equiv 0 \bmod 3 \\ y \equiv 1 \bmod 5 \end{cases} \longrightarrow \begin{cases} y = 3q \\ 3q \equiv 1 \bmod 5 \end{cases} \longrightarrow q = 2 \longrightarrow y = 6$$

So, we obtain:

$$f(a, b) = 10a + 6b$$

4.2.2 2020-07-03 ex.2 v.2

Let $f : \mathbb{Z}_5 \times \mathbb{Z}_7 \mapsto \mathbb{Z}_{35}$ be the isomorphism pf CRT. Calculate $f(a, b)$.

Solution

Let's compute $f(1, 0)$ and $f(0, 1)$:

$$y = f(1, 0) \longrightarrow \begin{cases} y \equiv 1 \bmod 5 \\ y \equiv 0 \bmod 7 \end{cases} \longrightarrow \begin{cases} 7q \equiv 1 \bmod 5 \\ y = 7q \end{cases} \longrightarrow q = 3 \longrightarrow y = 21$$

$$y = f(0, 1) \longrightarrow \begin{cases} y \equiv 0 \bmod 5 \\ y \equiv 1 \bmod 7 \end{cases} \longrightarrow \begin{cases} y = 5q \\ 5q \equiv 1 \bmod 7 \end{cases} \longrightarrow q = 3 \longrightarrow y = 15$$

So, we obtain:

$$f(a, b) = 21a + 15b$$

4.2.3 2021-07-16 ex.1

Let $f : \mathbb{Z}_4 \times \mathbb{Z}_5 \mapsto \mathbb{Z}_{20}$ be the isomorphism pf CRT. Calculate $f(a, b)$.

Solution

Let's compute $f(1, 0)$ and $f(0, 1)$:

$$y = f(1, 0) \longrightarrow \begin{cases} y \equiv 1 \bmod 4 \\ y \equiv 0 \bmod 5 \end{cases} \longrightarrow \begin{cases} 5q \equiv 1 \bmod 4 \\ y = 5q \end{cases} \longrightarrow q = 1 \longrightarrow y = 5$$

$$y = f(0, 1) \longrightarrow \begin{cases} y \equiv 0 \bmod 4 \\ y \equiv 1 \bmod 5 \end{cases} \longrightarrow \begin{cases} y = 4q \\ 4q \equiv 1 \bmod 5 \end{cases} \longrightarrow q = 4 \longrightarrow y = 16$$

So, we obtain:

$$f(a, b) = 5a + 16b$$

4.2.4 random.pdf ex.1 (CRT system)

Solve the following system:

$$\begin{cases} x \equiv 4 \bmod 11 \\ x \equiv 3 \bmod 17 \\ x \equiv 6 \bmod 18 \end{cases}$$

Solution

From the system we understand that $(a_1, a_2, a_3) = (4, 3, 6)$ and $N = 11 \cdot 17 \cdot 8 = 3366$. We can then express the mapping function f as:

$$f(a_1, a_2, a_3) = a_1 \cdot f(1, 0, 0) + a_2 \cdot f(0, 1, 0) + a_3 \cdot f(0, 0, 1)$$

Let's compute $f(1, 0, 0)$, $f(0, 1, 0)$ and $f(0, 0, 1)$:

- $f(1, 0, 0)$

  The last two equations of the system are solved for all multiples of $mcm(17, 18) = 306$. Note that 17 and 18 are coprime, so their minimum common multiple is just the product $17 \cdot 18 = 306$.

  The 1^{st} equation of the system can be solved if and only if the inverse $q = 306^{-1} \bmod 11$ does exist. Since $gcd(306, 11) = 1$, i.e., 306 and 11 are coprime, we know for sure that the inverse does exist. We can easily compute that inverse with the EEA/Gauss-Jordan method.

$$y = f(1, 0, 0) \longrightarrow \begin{cases} y \equiv 1 \bmod 11 \\ y \equiv 0 \bmod 17 \\ y \equiv 0 \bmod 18 \end{cases} \longrightarrow 306 \cdot q \equiv 1 \bmod 11 \longrightarrow y = 5 \cdot 306 = 1530$$

- $f(0, 1, 0)$

  In this case we have that $mcm(11, 18) = 198$ and $q = 198^{-1} \bmod 17 = 14$.

$$y = f(0, 1, 0) \longrightarrow \begin{cases} y \equiv 0 \bmod 11 \\ y \equiv 1 \bmod 17 \\ y \equiv 0 \bmod 18 \end{cases} \longrightarrow y = 14 \cdot 198 = 2772$$

- $f(0, 0, 1)$

  In this case we have that $mcm(11, 17) = 187$ and $q = 187^{-1} \bmod 18 = 13$.

$$y = f(0, 0, 1) \longrightarrow \begin{cases} y \equiv 0 \bmod 11 \\ y \equiv 0 \bmod 17 \\ y \equiv 1 \bmod 18 \end{cases} \longrightarrow y = 13 \cdot 187 = 2431$$

In the end, we obtain:

$$f(a_1, a_2, a_3) = a_1 \cdot 1530 + a_2 \cdot 2772 + a_3 \cdot 2431$$

So, by considering the input data $(a_1, a_2, a_3) = (4, 3, 6)$, we obtain:

$$x = f(4, 3, 6) = 4 \cdot 1530 + 3 \cdot 2772 + 6 \cdot 2431 = 29022 \equiv 2094 \bmod 3366$$

5 Quadratic residues: solving $x^2 \equiv r \bmod n$

5.1 Euler's Criterion

Euler's criterion is a formula for determining whether an integer is a quadratic residue modulo a prime. Precisely:

Euler's Criterion Let p be an **odd** prime and r be an integer **coprime** to p. Then:

$$r^{\frac{p-1}{2}} \equiv \begin{cases} 1 \bmod p & \text{if there is an integer } x \text{ such that } r = x^2 \bmod p, \\ -1 \bmod p & \text{if there is no such integer.} \end{cases}$$

Example) Finding primes for which r is a residue:

Let $r = 17$. For which primes p is 17 a quadratic residue?

We can test some primes manually by using Euler's Criterion:

- $p = 3$

$$17^{\frac{3-1}{2}} = 17 \equiv 2 \bmod 3 \equiv -1 \bmod 3$$

17 is not a quadratic residue of 3;

- $p = 13$

$$17^{\frac{13-1}{2}} = 17^6 \equiv 4^6 \bmod 13 = (16 \cdot 16 \cdot 16) \bmod 13 \equiv (3 \cdot 3 \cdot 3) \bmod 13 \equiv 1 \bmod 13$$

17 is a quadratic residue of 13;

If we keep calculating the values, we find that 17 is a quadratic residue of $\{13, 19, ...\}$ but is not a quadratic residue of $\{3, 5, 7, 11, 23, ...\}$.

Example) Finding residues given a prime modulus p:

Which numbers are quadratic residues modulo 17? We can manually calculate them as:

$$\begin{aligned} 1^2 &= 1 \\ 2^2 &= 4 \\ 3^2 &= 9 \\ 4^2 &= 16 \\ 5^2 &= 25 \equiv 8 \bmod 17 \\ 6^2 &= 36 \equiv 2 \bmod 17 \\ 7^2 &= 49 \equiv 15 \bmod 17 \\ 8^2 &= 64 \equiv 13 \bmod 17 \end{aligned}$$

So, the set of the quadratic residues modulo 17 is $\{1, 2, 4, 8, 9, 13, 15, 16\}$. Note that we did not need to calculate squares for the values 9 through 16, as they are all negatives of the previously squared values. As a matter of fact, the group of quadratic residues modulo p is:

$$G = \left\{ 1^2, 2^2, 3^2, ..., \left(\frac{p-1}{2}\right)^2 \right\} \pmod p$$

5.2 Strategies to find quadratic residues

5.2.1 Using CTR to find quadratic residues

Let's consider the isomorphic ring $\mathbb{Z}_{24} \approx \mathbb{Z}_3 \times \mathbb{Z}_8$. As we've already seen, each element x of $\mathbb{Z}_{24}$ is mapped to a tuple (a, b). So, solving the equation

$$x^2 \equiv 1 \bmod 24$$

is the same thing as solving separately the following equations:

$$\begin{cases} a^2 \equiv 1 \bmod 3 \\ b^2 \equiv 1 \bmod 8 \end{cases}$$

This is much more straightforward (if you already know the map $x \leftrightarrow (a, b)$):

$$a_1 = 1,\ a_2 = 2,\ b_1 = 1,\ b_2 = 7$$

and we obtain 4 possible solutions to our equation:

$$\begin{aligned} (a_1, b_1) = (1, 1) &\mapsto x = 1 \\ (a_1, b_2) = (1, 7) &\mapsto x = 7 \\ (a_2, b_1) = (2, 1) &\mapsto x = 17 \\ (a_2, b_2) = (2, 7) &\mapsto x = 23 \end{aligned}$$

5.2.2 General approach to find quadratic residues

Let $x^2 \equiv r \bmod n$

- if $r = 0$ a unique solution does exist, that is $x \equiv 0 \bmod n$;
- if $r > 0$, use CRT to split the equation in many simpler 2^{nd} degree equations $x^2 \equiv a \bmod p^k$ where p is one of the prime factors of n and $\gcd(a, p) = 1$. For each equation:
  - if p is odd, apply Euler's Criterion:

    $a^{\frac{p-1}{2}} \equiv -1 \bmod p \quad \longrightarrow \quad$ no solutions (a is not a quadratic residue modulus p);

    $a^{\frac{p-1}{2}} \equiv \ \ 1 \bmod p \quad \longrightarrow \quad$ two solutions x_1, x_2;
  - if $p = 2$:
    - if $k = 1$ a unique solution does exist, that is $x \equiv 1 \bmod 2$;
    - if $k = 2$ there are two solutions, but if and only if $a \equiv 1 \bmod 4$, and these solutions are:

      $$x_1 \equiv 1 \bmod 4$$
      $$x_2 \equiv 3 \bmod 4$$

    - if $k \geq 3$ there are four solutions if $a \equiv 1 \bmod 8$ and no solutions otherwise. Specifically for $k = 3$, these solutions are {1,3,5,7}. If $k > 3$ the solutions can be computed with the procedure below which starts with each of the solutions mod 2^3 and produces solutions by induction for higher powers of 2:

      Suppose $x_k^2 \equiv a \bmod 2^k$ for $k \geq 3$. By definition, this means $x_k^2 - a$ is divisible by 2^k. If $(x_k^2 - a)/2^k$ is odd, let $i = 1$. Otherwise, let $i = 0$. Then

      $$x_{k+1} = x_k + i \cdot 2^{k-1}$$

      is a solution to $x_{k+1}^2 \equiv a \bmod 2^{k+1}$.

      Disclaimer: I tried this method on $x^2 \equiv 1 \bmod 16$ but I didn't succeed. Here's the source: https://www.johndcook.com/blog/quadratic_congruences/

The number of solutions modulo n is given by the product of the number of solutions obtained for each equation modulo p^k. Moreover, if x is a quadratic residue also $-x$ is a quadratic residue!

5.3 Past exams exercises (Quadratic residues and CRT)

5.3.1 2020-07-03 ex.1 v.1

Let $p = 11,\ q = 19,\ n = pq = 209$. How many solutions does the equation $x^2 \equiv 171 \bmod 209$ have?

Solution

Let's follow the general approach previously described. Since $r = 171 > 0$, let's use CRT to split the equation:

$$\begin{cases} x^2 \equiv 171 \bmod 11 \\ x^2 \equiv 171 \bmod 19 \end{cases} \longrightarrow \begin{cases} x^2 \equiv 6 \bmod 11 \\ x^2 \equiv 0 \bmod 19 \end{cases}$$

By looking at the 2nd equation, we can already conclude that there's only one solution to it:

$$x \equiv 0 \bmod 19$$

However, since this solution doesn't verify the 1st equation of the system, we can say for sure that the equation $x^2 \equiv 171 \bmod 209$ has no solution.

5.3.2 2020-07-03 ex.1 v.2

Let $p = 11,\ q = 19,\ n = pq = 209$. How many solutions does the equation $x^2 \equiv 130 \bmod 209$ have?

Solution

As in the previous exercise, we use again CRT to split the equation:

$$\begin{cases} x^2 \equiv 130 \bmod 11 \\ x^2 \equiv 130 \bmod 19 \end{cases} \longrightarrow \begin{cases} x^2 \equiv 9 \bmod 11 \\ x^2 \equiv 16 \bmod 19 \end{cases}$$

Since 11 and 19 are both odd primes, let's apply Euler's Criterion to both equations:

$$9^{\frac{11-1}{2}} = 9^5 = \left(9^2\right)^2 \cdot 9 \equiv 4^2 \cdot 9 = 144 \equiv 1 \bmod 11$$

$$16^{\frac{19-1}{2}} = 16^9 = \left(16^2\right)^4 \cdot 16 \equiv 9^4 \cdot 16 = 81^2 \cdot 16 \equiv 5^2 \cdot 16 \equiv 6 \cdot 16 = 96 \equiv 1 \bmod 19$$

Euler's Criterion is satisfied for both equations! So, each equation has 2 solutions. We can conclude then that the equation $x^2 \equiv 130 \bmod 209$ has 4 solutions.

5.3.3 January 2021 ex.2

How many solutions does the equation $x^2 \equiv 173 \bmod 291$ have?

Solution

By using CRT to split the equation, we obtain:

$$\begin{cases} x^2 \equiv 173 \bmod 3 \\ x^2 \equiv 173 \bmod 97 \end{cases} \longrightarrow \begin{cases} x^2 \equiv 2 \bmod 3 \\ x^2 \equiv 76 \bmod 97 \end{cases}$$

By applying Euler's Criterion to the 1st equation

$$2^{\frac{3-1}{2}} = 2 \equiv -1 \bmod 3$$

we find that it has no solution. So, the equation $x^2 \equiv 173 \bmod 291$ doesn't have any solution as well.

6 RSA

6.1 Euler's Phi Function

We consider the ring $\mathbb{Z}_m$, i.e., the set of integers $\{0, 1, ..., m-1\}$. We are interested in the problem of knowing how many numbers in this set are relatively prime to m. This quantity is given by *Euler's phi function*. We can guess that calculating the function by running through all the elements of the ring and computing their gcd with m is extremely slow if the numbers are large.

There exists a relation to calculate it much more easily if we know the factorization of m, which is given in the following theorem:

Euler's Phi Function Let m have the following canonical factorization

$$m = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_n^{e_n}$$

where the p_i are distinct prime numbers and e_i are positive integers. Then:

$$\Phi(m) = \prod_{i=1}^{n} (p_i^{e_i} - p_i^{e_{i-1}})$$

Since the value of n, i.e., the number of distinct prime factors, is always quite small even for large numbers **m**, evaluating the product symbol is computationally easy.

Example) Let $m = 240$. The factorization of 240 in the canonical form is

$$m = 240 = 15 \cdot 15 = 2^4 \cdot 3 \cdot 5 = p_1^{e_1} \cdot p_2^{e_2} \cdot p_3^{e_3}$$

There are three distinct prime factors, i.e., $n = 3$. The value for Euler's phi function is then:

$$\Phi(m) = (2^4 - 2^3)(3^1 - 3^0)(5^1 - 5^0) = 8 \cdot 2 \cdot 4 = 64$$

That means that 64 integers in the range $\{0, 1, ..., 239\}$ are coprime to $m = 240$.

It is important to stress that we need to know the factorization of m in order to calculate Euler's phi function quickly in this manner. As we will see, this property is at the **heart of the RSA** public-key scheme.

6.2 Fermat's Little Theorem

Fermat's little theorem is helpful for **primality testing**.

Fermat's Little Theorem
Let a be an integer and p be a prime, then:

$$a^p \equiv a \bmod p$$

We note that arithmetic in finite fields $GF(p)$ is done modulo p, and hence, the theorem holds for all integers a which are elements of a finite field $GF(p)$. The theorem can be stated in the form:

$$a^{p-1} \equiv 1 \bmod p$$

which is often useful in cryptography. One application is the computation of the inverse in a finite field. We can rewrite the equation as $a \cdot a^{p-2} \equiv 1 \bmod p$. This is exactly the definition of the multiplicative inverse. Thus, we immediately have a way for inverting an integer a modulo a prime:

$$a^{-1} \equiv a^{p-2} \bmod p$$

We note that this inversion method holds only if p is a prime.

Example) Let $p = 7$ and $a = 2$. We can compute the inverse of a as:

$$a^{p-2} = 2^5 = 32 \equiv 4 \bmod 7$$

This is easy to verify: $2 \cdot 4 \equiv 1 \bmod 7$.

Performing the exponentiation like in the previous example is usually slower than using the EEA. However, there are situations where it is advantageous to use Fermat's Little Theorem (e.g., smart cards or other devices which have a hardware accelerator for fast exponentiation).

6.3 Euler's Theorem

A generalization of Fermat's Little Theorem to any integer moduli, i.e., moduli that are not necessarily primes, is *Euler's theorem.*

Euler's Theorem
Let a and m be integers with $\gcd(a, m) = 1$, then:

$$a^{\Phi(m)} \equiv 1 \bmod m$$

Since it works modulo m, it is applicable to integer rings $\mathbb{Z}_m$. We show now an example for Euler's theorem with small values.

Example) Let $m = 12$ and $a = 5$. First, we computer Euler's phi function of m:

$$\Phi(12) = \Phi(2^2 \cdot 3) = (2^2 - 2^1)(3^1 - 3^0) = 4$$

Now we can verify Euler's theorem:

$$5^{\Phi(12)} = 5^4 = 625 \equiv 1 \bmod 12$$

It is easy to show that Fermat's Little Theorem is a special case of Euler's theorem. If m is prime, it holds that $\Phi(m) = (m^1 - m^0) = m - 1$. If we use this value for Euler's theorem, we obtain: $a^{\Phi(m)} = a^{m-1} \equiv 1 \bmod m$, which is exactly Fermat's Little Theorem.

Just like Fermat's Little Theorem, Euler's Theorem can be written as:

$$a \cdot a^{\Phi(m)-1} \equiv 1 \bmod m$$

$$a^{-1} \equiv a^{\Phi(m)-1} \bmod m$$

6.4 The RSA Cryptosystem

6.4.1 Introduction

There are many applications for RSA, but in practice it is most often used for:

- encryption of small pieces of data, especially for key transport;
- digital signatures, e.g., for digital certificates on the Internet;

RSA encryption is not meant to replace symmetric ciphers because it is several times slower than ciphers such as AES. The main use of the encryption feature is to securely exchange a key for a symmetric cipher. The underlying one-way function of RSA is the integer factorization problem: multiplying two large primes is computationally easy, but factoring the resulting product is very hard. Euler's theorem and Euler's phi function play important roles in RSA.

6.4.2 Encryption and Decryption

RSA enryption and decryption is done in the integer ring $\mathbb{Z}_n$. RSA encrypts plaintexts x, where we consider the bit string representing x to be an element in $\mathbb{Z}_n = \{0, 1, ..., n-1\}$. As a consequence, the binary value of the plaintext x must be less than n. The same holds for the ciphertext.

RSA Encryption Given the public key $(n, e) = k_{pub}$ and the plaintext x, the encryption function is:

$$y = e_{k_{pub}}(x) \equiv x^e \bmod n$$

where $x, y \in \mathbb{Z}_n$.

RSA Decryption Given the private key $d = k_{pr}$ and the ciphertext y, the decryption function is:

$$x = d_{k_{pr}}(y) \equiv y^d \bmod n$$

where $x, y \in \mathbb{Z}_n$.

In practice, x, y, n and d are very long numbers, usually 1024 bit long or more. The value e is sometimes referred to as encryption exponent or **public exponent**, and the private key d is sometimes called decryption exponent or **private exponent**. If Alice wants to send an encrypted message to Bob, Alice, needs to have his public key (n, e), and Bob decrypts with his private key d.

We can already state a few requirements for the RSA cryptosystem:

- since an attacker has access to the public key, it must be computationally infeasible to determine the private key d given the public-key values e and n;
- since x is only unique up to the size of the modulus n, we cannot encrypt more than l bits with one RSA encryption, where l is the bit length of n;
- it should be relatively easy to calculate $x^e \bmod n$, i.e., to encrypt, and $y^d \bmod n$, i.e., to decrypt. So, we need a method for fast exponentiation with very long numbers;
- for a given n, there should be many private-key/public-key pairs, otherwise an attacker might be able to perform a brute-force attack (it turns out that this requirement is easy to satisfy);

6.4.3 Key Generation

Depending on the public-key scheme, key generation can be quite complex. As a remark, we note that key generation is usually not an issue for block or stream ciphers. Here are the steps involved in computing the public and private-key for an RSA cryptosystem:

RSA Key Generation

Output:

- **public key**: $k_{pub} = (n, e)$
- **private key**: $k_{pr} = (d)$

Steps:

1. Choose two large primes p and q;
2. Compute $n = p \cdot q$;
3. Compute $\Phi(n) = (p-1)(q-1)$;
4. Select the public exponent $e \in \{1, 2, ..., \Phi(n) - 1\}$ such that
$$\gcd(e, \Phi(n)) = 1$$
5. Compute the private key d such that
$$d \cdot e \equiv 1 \bmod \Phi(n)$$

The condition that $\gcd(e, \Phi(n)) = 1$ ensures that the inverse of e exists modulo $\Phi(n)$, so that there is always a private key d. The computation of the keys d and e can be done at once using the extended Euclidean algorithm (EEA). In practice, one often starts by first selecting the public parameter e in the range $0 < e < \Phi(n)$. The value e must satisfy the condition $\gcd(e, \Phi(n)) = 1$. We apply the EEA with the input parameters n and e and obtain the relationship:

$$\gcd(e, \Phi(n)) = s \cdot \Phi(n) + t \cdot e$$

If $\gcd(e, \Phi(n)) = 1$, we know that e is a valid public key. Moreover we also know that the parameter t computed by the EEA is the inverse of e, and thus:

$$d = t \bmod \Phi(n)$$

In case that e and $\Phi(n)$ aren't relatively prime, we simply select a new value for e and repeat the process. Note that the coefficient s of the EEA is not required for RSA and does not need to be computed.

What is interesting is that the message x is first raised to the e-th power during encryption and the result y is raised to the d-th power in the decryption, and the result of this is again equal to the message x. Expressed as an equation, this process is:

$$d_{k_{pr}}(y) = d_{k_{pr}}(e_{k_{pub}}(x)) \equiv (x^e)^d \equiv x^{de} \equiv x \bmod n$$

6.4.4 Fast Exponentiation: The Square-and-Multiply Algorithm

Public-key algorithms are based on arithmetic with very long numbers. Unless we pay close attention to how to realize the necessary computations, we can easily end up with schemes that are too slow for practical use. RSA encryption and decryption are both based on modular exponentiation. We restate both operations here for convenience:

$$y = e_{k_{pub}}(x) \equiv x^e \bmod n \text{ (encryption)}$$

$$x = d_{k_{pr}}(y) \equiv y^d \bmod n \text{ (decryption)}$$

One algorithm for fast exponentiation is the *square-and-multiply algorithm*, which provides a systematic way for finding the sequence in which we have to perform squarings and multiplications by x for computing x^H.

Example) We want to compute x^{26}. What is the sequence of squarings and multiplications which minimizes the number of operations?

$$x \xrightarrow{SQ} x^2 \xrightarrow{MUL} x^3 \xrightarrow{SQ} x^6 \xrightarrow{SQ} x^{12} \xrightarrow{MUL} x^{13} \xrightarrow{SQ} x^{26}$$

This approach takes a total of six operations! Let's see how is this achieved...

Square-and multiply algorithm The algorithm is based on scanning the bit of the exponent from the left (the most significant bit) to the right (the least significant bit). In every iteration, i.e., for every exponentiation bit, the current result is squared. If and only if the currently scanned exponent bit has the value 1, a multiplication of the current result by x is executed following the squaring.

Example) We again consider the exponentiation x^{26}. The binary representation of the exponent is:

$$26 = 11010_2 = (h_4 h_3 h_2 h_1)_2$$

The algorithm scans the exponent bits, starting from the MSB h_4 up to the LSB h_0.

The initialization value of x for the algorithm is x^0:

0) $h_4 = 1 \rightarrow res_0 = (x^0)^2 \cdot x = x$
1) $h_3 = 1 \rightarrow res_1 = (res_0)^2 \cdot x = x^3$
2) $h_2 = 0 \rightarrow res_2 = (res_1)^2 = x^6$
3) $h_1 = 1 \rightarrow res_3 = (res_2)^2 \cdot x = x^{13}$
4) $h_0 = 0 \rightarrow res_4 = (res_3)^2 = x^{26}$

When performing exponentiation modulo n with this algorithm, the modulo reduction must be applied after each multiplication and squaring operation in order to keep the intermediate results small.

How many operations are required on average for an exponentiation with a 1024-bit exponent? Straightforward exponentiation takes $2^{1024} \approx 10^{300}$ multiplications. That is completely impossible, no matter what computer resources we might have at hand (the number of atoms in the visible universe is estimated to be around 2^{300}, `LOL`). However, this algorithm requires (on average) only:

$$1.5 \cdot 1024 = 1536$$

squarings and multiplications. This is an impressive example for the difference of an algorithm with linear complexity (straightforward exponentiation) and logarithmic complexity (square-and-multiply algorithm). Remember, though, that each of the 1536 individual squarings and multiplications involves 1024-bit numbers.

6.4.5 Fast Encryption with Short Public Exponents

A surprisingly simple and very powerful trick can be used when RSA operations with the public key e, like encryption, are concerned. The public key e can be chosen to be a very small value. In practice, the three values $e = 3$, $e = 17$ and $e = 2^{16} + 1 = 65537$ are of particular importance. The resulting complexities when using these public keys are given in the following table:

Public key e	e as binary string	#MUL + #SQ
3	11_2	2
17	$1\ 0001_2$	5
65537	$1\ 0000\ 0000\ 0000\ 0001_2$	17

We note that all three exponents listed above have a low Hamming weight, i.e., number of ones in the binary representation. This results in a particularly low number of operations for performing an exponentiation. Interestingly, RSA is still secure if such short exponents are being used. Note that the private key d still has in general the full bit length even though e is short. An important consequence of the use of short public exponents is that encryption of a message and verification of an RSA signature is a very fast operation.

Unfortunately, there is no such easy way to accelerate RSA when the private key d is involved, i.e., for decryption and signature generation. Hence, these two operations tend to be slow. Other public-key algorithms, in particular elliptic-curves, are often much faster for these two operations. The following section shows how we can achieve a more moderate speed-up when using the private exponent d.

WATCH OUT: if m is small, it is good practice to use $e = 65537$ because of the *low public exponent attack* which can be mounted to decrypt small messages. If $e = 3$ and x is small, it's probable that the (mod n) operation won't be used even once during the encryption. If that's the case, x could be simply recovered by computing $\sqrt[3]{y}$. If m is big and the (mod n) operation is used at least once, we're good.

6.4.6 Fast Decryption with the Chinese Remainder Theorem

The private key must have a length of at least $0.3t$ bit, where t is the bit length of the modulus n. In practice, e is often chosen short and d has full bit length.

Our goal is to perform the exponentiation $y^d \bmod n$ efficiently. First we note that the party who possesses the private key also knows the primes p and q. The basic idea of the CRT is that rather than doing arithmetic with one "long" modulus n, we do two individual exponentiations modulo the two "short" primes p and q. Like any transform, there are three steps: transforming into the CRT domain, computation in the CRT domain, and inverse transformation of the result. Those three steps are explained below:

Transformation of the Input into the CRT Domain

We simply reduce the base element y modulo the two factors p and q of the modulus n, and obtain what is called the **modular representation** of y:

$$y_p \equiv y \bmod p$$
$$y_q \equiv y \bmod q$$

Exponentiation in the CRT Domain

With the reduced versions of y we perform the following two exponentiations:

$$x_p = y_p^{d_p} \bmod p$$
$$x_q = y_q^{d_q} \bmod q$$

where the two new exponents are given by:

$$d_p \equiv d \bmod (p-1)$$
$$d_q \equiv d \bmod (q-1)$$

Inverse Transformation into the Problem Domain

The remaining step is now to assemble the final result x from its modular representation (x_p, x_q). This follows from the CRT and can be done as:

$$x \equiv [qc_p]x_p + [pc_q]x_q \bmod n$$

where the coefficients c_p and c_q are computed as:

$$c_p \equiv q^{-1} \bmod p, \quad c_q \equiv p^{-1} \bmod q$$

Since the primes change very infrequently for a given RSA implementation, the two expressions in brackets can be precomputed. After the precomputations, the entire reverse transformation is achieved with merely two modular multiplications and one modular addition.

Example) Let the RSA parameters be given by:

$$\begin{aligned} p &= 11 & e &= 7 \\ q &= 13 & d &\equiv e^{-1} \equiv 103 \bmod 120 \\ n &= p \cdot q = 143 \end{aligned}$$

We now compute an RSA decryption for the ciphertext $y = 15$ using the CRT,
i.e., the value $y^d = 15^{103} \bmod 143$. In the first step, we compute the modular representation of y:

$$y_p \equiv 15 \equiv 4 \bmod 11$$

$$y_q \equiv 15 \equiv 2 \bmod 13$$

In the second step, we perform the exponentiation in the transform domain with the short exponents.

These are:

$$d_p \equiv 103 \equiv 3 \bmod 10$$

$$d_q \equiv 103 \equiv 7 \bmod 12$$

Here are the exponentiations:

$$x_p \equiv y_p^{d_p} = 4^3 = 64 \equiv 9 \bmod 11$$

$$x_q \equiv y_q^{d_q} = 2^7 = 128 \equiv 11 \bmod 13$$

In the last step, we have to compute x from its modular representation (x_p, x_q).

For this, we need the coefficients:

$$c_p \equiv 13^{-1} \equiv 2^{-1} \equiv 6 \bmod 11 \quad c_q = 11^{-1} \equiv 6 \bmod 13$$

The plaintext x follows now as:

$$\begin{aligned} x &\equiv [qc_p]x_p + [pc_q]x_q \bmod n \\ x &\equiv [13 \cdot 6]9 + [11 \cdot 6]11 \bmod 143 \\ x &\equiv 702 + 726 = 1428 \equiv 141 \bmod 143 \end{aligned}$$

If you want to verify the result, you can compute $y^d \bmod 143$ using the square-and-multiply algorithm.

The **total speedup** obtained through the CRT is a factor of 4, which can be very valuable in practice. Since there are hardly any drawbacks involved, CRT-based exponentiations are used in many cryptographic products, e.g., for Web browser encryption, smart cards, banking applications, etc...

6.5 Past exams exercises (RSA)

6.5.1 2021-09-14 ex.3

Given the RSA parameters $p = 5$ and $q = 11$, what is a valid combination for RSA?

a) $e = 12 \quad M = 6$

b) $e = 17 \quad d = 33 \quad M = 6 \quad C = 41$

c) $e = 11 \quad d = 11 \quad M = 6 \quad C = 16$

where M is the message (plaintext) and C is the ciphertext.

Solution

There's no other way for us to identify the correct answer to this problem other than "testing" all the given answers. Here a few checks we can perform (based on what we've explained in the previous paragraphs):

- verify that $e \in \mathbb{Z}_{\Phi(n)} = \{1, 2, ..., \Phi(n) - 1\}$ such that $\gcd(e, \Phi(n)) = 1$ (e is invertible);
  - if so, verify that $d \cdot e \equiv 1 \mod \Phi(n)$ (d is the inverse of e modulo $\Phi(n)$);
    - if so, verify that $C = M^e \mod n$ (C is the ciphertext for M);

First of all, let's compute:

- $n = p \cdot q = 5 \cdot 11 = 55$;
- $\Phi(n) = (p - 1)(q - 1) = 4 \cdot 10 = 40$;

Let's then test each answer:

- answer (a):
  - we can already exclude answer (a) because $\gcd(e, \Phi(n)) = 2 \neq 1$;
- answer (b):
  - $\gcd(e, \Phi(n)) = 1$; ✓
  - $d \cdot e \equiv 1 \mod \Phi(n) \longrightarrow 33 \cdot 17 = 561 \equiv 1 \mod 40$; ✓
  - $C = M^e \mod n \longrightarrow C = 6^{17} \mod 55$

    $$\begin{cases} C = 6^{17} \equiv 1^{17} = 1 \mod 5 \\ C = 6^{17} = 6 \cdot 36^8 \equiv 6 \cdot 3^8 = 2 \cdot 3^9 = 2 \cdot 27^3 \equiv 2 \cdot 5^3 = 10 \cdot 25 \equiv 10 \cdot 3 = 30 \equiv 8 \mod 11 \end{cases}$$

    Since $C = 41$ both the previous equations are verified and the system is solved. ✓

    So, C is the ciphertext for M and answer (b) is correct!
- answer (c):
  - $\gcd(e, \Phi(n)) = 1$; ✓
  - $d \cdot e \equiv 1 \mod \Phi(n)$: $11 \cdot 11 = 121 \equiv 1 \mod 40$ ✓
  - $C = M^e \mod n$: $C = 6^{11} \mod 55$

    $$\begin{cases} C = 6^{11} \equiv 1^{11} = 1 \mod 5 \\ C = 6^{11} = 6 \cdot 36^5 \equiv 6 \cdot 3^5 = 2 \cdot 27^2 \equiv 2 \cdot 5^2 = 50 \equiv 6 \mod 11 \end{cases}$$

    Since $C = 16 \equiv 5 \not\equiv 6 \mod 11$ the second equation isn't verified and answer (c) is wrong;

6.5.2 FacsimileCryptography 2020 ex.3

In RSA:

- user A keys are $\text{sk}_A = (p_A, q_A, d_A) = (5, 11, 23)$ and $\text{pk}_A = (n_A, e_A) = (55, 7)$;
- user B keys are $\text{sk}_B = (p_B, q_B, d_B) = (3, 7, 5)$ and $\text{pk}_B = (n_B, e_B) = (21, 5)$;

A wants to send to B the message M. So, she sends the ciphertext $C = \text{Enc}_{\text{pk}_B}(M)$ and adds as her digital signature the pair $(\text{Enc}_{\text{pk}_B}(F), h(M))$ where $F = \text{Enc}_{\text{sk}_A}(h(M))$ and $h(x)$ is a fixed hash function.

Assuming $h(M) = 18$ find the digital signature, i.e., the pair $(\text{Enc}_{\text{pk}_B}(F), h(M))$.

Solution

In order to find the digital signature we must perform the same computations user A did before sending the message to B:

- compute $F = \text{Enc}_{\text{sk}_A}(h(M)) = \text{Enc}_{\text{sk}_A}(18) = 18^{d_A} \bmod n_A = 18^{23} \bmod 55$;
- compute $\text{Enc}_{\text{pk}_B}(F) = (F)^{e_B} \bmod n_B = F^5 \bmod 21$;

One "fast" way to compute F by hand could be using the square-and-multiply algorithm:

$$23 = 1\ 0111_2 = h_4 h_3 h_2 h_1 h_0$$

Steps:

- $h_4 = 1 \quad res_0 = 18$
- $h_3 = 0 \quad res_1 = res_0^2 = 324 \equiv -6 \bmod 55$
- $h_2 = 1 \quad res_2 = res_1^2 \cdot 18 = 36 \cdot 18 = 4 \cdot 9 \cdot 2 \cdot 9 = 8 \cdot 81 \equiv 8 \cdot 26 = 2 \cdot 104 \equiv 2 \cdot (-6) \bmod 55$
- $h_1 = 1 \quad res_3 = res_2^2 \cdot 18 = (2 \cdot (-6))^2 \cdot 18 = 4 \cdot 36 \cdot 18 \equiv 4 \cdot 2 \cdot (-6) \bmod 55$
- $h_0 = 1$

$$\begin{aligned} res_4 = res_3^2 \cdot 18 = 4^2 \cdot (2 \cdot (-6))^2 \cdot 18 \equiv \\ \equiv 4^2 \cdot 4 \cdot 2 \cdot (-6) = -16 \cdot 48 \equiv \\ \equiv -16 \cdot (-7) = 2 \cdot 8 \cdot 7 = \\ = 2 \cdot 56 \equiv 2 \bmod 55 \end{aligned}$$

Notice how we've re-used the results obtained during previous steps (marked with the same color) in order to speed up the computations!

So, we've obtained $F \equiv 2 \bmod 55$, from which we can compute:

$$\text{Enc}_{\text{pk}_B}(F) = F^5 \bmod 21 = 32 \bmod 21 \equiv 11 \bmod 21$$

In the end, the digital signature is:

$$(\text{Enc}_{\text{pk}_B}(F), h(M)) = (11, 18)$$

6.6 Exercises from slides (RSA)

6.6.1 Exercise 9.2.11

Consider an RSA cryptosystem with $p = 11$, $q = 23$ and $e = 3$.
Encipher $M = (111001)_2$ and find the secret key $k_{pr} = (\Phi(n), d)$. Decipher $C = (11010101)_2$.

Solution

First of all, let's convert M and C from binary to decimal:

$$M = 57 \quad C = 85$$

Let's perform the steps for the key generation:

1. $n = p \cdot q = 11 \cdot 23 = 253$;
2. $\Phi(n) = (p-1)(q-1) = 220$;
3. Let's choose one of the short public exponents: $e = 3$.

   Let's test the condition $\gcd(e, \Phi(n)) = 1$ by applying the EEA:

   $$\begin{aligned} &\gcd(220, 3) = \gcd(3, 1) && 220 = 73 \cdot 3 + 1 \\ &\gcd(3, 1) \quad = 1 \ \checkmark \end{aligned}$$

   As we can see, the condition is satisfied and we can proceed by computing the inverse of e:

   $$r_2 = 1 = 220 - 73 \cdot 3 = [1]r_0 + [-73]r_1$$

   $$e^{-1} = 3^{-1} \equiv -73 \equiv 147 \bmod 220$$

4. $d = e^{-1} \equiv 147 \bmod 220$

So, the secret key is:

$$k_{pr} = (\Phi(n), d) = (220, 147)$$

Now, let's encrypt M:

$$C_M \equiv M^e \bmod n \equiv 57^3 \bmod 253$$

$$C_M = 57^3 \equiv 3249 \cdot 57 \equiv 213 \cdot 57 = 12141 \equiv 250 \bmod 253$$

For completeness, we can verify that everything went smoothly by decrypting the ciphertext:

$$M = C_M^d \bmod n \equiv 250^{147} \bmod 253$$

$$\begin{aligned} M = 250^{147} &\equiv -3^{147} = \\ &= -\left(3^5\right)^{29} \cdot 3^2 = -243^{29} \cdot 3^2 \equiv \\ &\equiv -(-10)^{29} \cdot 3^2 = 10^{29} \cdot 3^2 = \\ &= \left(10^3\right)^9 \cdot 10^2 \cdot 3^2 \equiv -12^9 \cdot 10^2 \cdot 3^2 = \\ &= -4^9 \cdot 3^9 \cdot 10^2 \cdot 3^2 = -4^9 \cdot 3^5 \cdot 3^5 \cdot 10^2 \cdot 3 \equiv \\ &\equiv -4^9 \cdot (-10) \cdot (-10) \cdot 10^2 \cdot 3 = -4^9 \cdot 10^3 \cdot 10 \cdot 3 \equiv \\ &\equiv -4^9 \cdot (-12) \cdot 10 \cdot 3 = 4^{10} \cdot 10 \cdot 3^2 = 2^{10} \cdot 2^{10} \cdot 10 \cdot 3^2 \equiv \\ &\equiv 12 \cdot 12 \cdot 10 \cdot 3^2 = 12 \cdot 1080 \equiv 12 \cdot 68 = 816 \equiv 57 \bmod 253 \ \checkmark \end{aligned}$$

The same reasoning goes for decrypting C:

$$M_C \equiv 85^{147} \bmod 253 \equiv \ldots \equiv 101 \bmod 253$$

7 Diffie-Hellman Key Exchange (DHKE)

7.1 Introduction to the DHKE

The *Diffie-Hellman key exchange (DHKE)* was the first asymmetric scheme published in the open literature. It provides a practical solution to the key distribution problem, i.e., it enables two parties to derive a common secret key by communicating over an insecure channel. This fundamental key agreement technique is implemented in many cryptographic protocols like SSH, TLS and IPsec. The basic idea behind the DHKE is that exponentiation in $\mathbb{Z}_p^*$, with p prime, is a one-way function and that exponentiation is commutative:

$$k \equiv (\alpha^x)^y \equiv (\alpha^y)^x \bmod p$$

The value k is the joint secret which can be used as the session key between the two parties.

7.2 DHKE Protocol Explanation

Let's consider two parties, Alice and Bob, who would like to establish a shared secret key. There is possibly a trusted third party that properly chooses the public parameters which are needed for the key exchange. However, it is also possible that Alice or Bob generate the public parameters. Strictly speaking, the DHKE consists of two protocols, the set-up protocol and the main protocol, which performs the actual key exchange. The set-up protocol consists of the following steps:

Diffie-Hellman Set-up

1. Choose a large prime p;
2. Choose an integer $\alpha \in \{2, 3, ..., p-2\}$;
3. Publish p and α;

These two values are sometimes referred to as *domain parameters*. If Alice and Bob both know the public parameters p and α, they can generate a joint secret key k with the following key-exchange protocol:

Diffie-Hellman Key Exchange

Alice		**Bob**
choose $a = k_{prA} \in \{2, 3, ..., p-2\}$		choose $b = k_{prB} \in \{2, 3, ..., p-2\}$
compute $A = k_{pubA} \equiv \alpha^a \bmod p$		compute $B = k_{pubB} \equiv \alpha^b \bmod p$
	$\xrightarrow{A}$	
	$\xleftarrow{B}$	
compute $k_{AB} \equiv B^a \bmod p$		compute $k_{AB} \equiv A^b \bmod p$

Now Alice and Bob both share the session key $k_{AB} \equiv \alpha^{ab} \bmod p$. The key can be used to establish a secure communication between the parties, e.g., by using k_{AB} as key for a symmetric algorithm like AES.

The computational aspects of DHKE are quite similar to those of RSA. During the set-up phase, we generate p using a probabilistic prime-finding algorithm. p should have a similar length as the RSA modulus n, i.e., 1024 bit or beyond, in order to provide strong security. The integer α needs to have a special property: it should be a generator (a topic which we discuss in the following sections). The session key k_{AB} has the same bit length as p. If we want to use it as a symmetric key for algorithms such as AES, we can simply take the 128 most significant bits. Alternatively, a hash function is sometimes applied to k_{AB} and the output is then used as a symmetric key.

During the actual protocol, we first have to choose the private keys a and b. They should stem from a true random generator in order to prevent an attacker from guessing them. For computing the public keys A and B as well as for computing the session key, both parties can make use of the square-and-multiply algorithm. The public keys are typically precomputed. The main computation that needs to be done for a key exchange is thus the exponentiation for the session key.

7.3 Exercises from slides (DHKE)

7.3.1 Exercise 9.1.6

Consider DH key agreement with modulus p of 1024 bit and g a generator of a subgroup of order $\approx 2^{160}$.

i) What is the maximum value that the private keys should have?

ii) How long does the computation of the session key take on average if one modular multiplication takes 700 μs, and one modular squaring 400 μs? Assume that the public keys have already been computed.

Solution

i) As we've already seen in the main protocol, the maximum value of private keys is $p - 2$. Since p has a length of 1024 bit, its maximum value is $p_{max} = 2^{1024} - 1$. So, the maximum value that the private keys should have is:

$$max = p_{max} - 2 = 2^{1024} - 3$$

ii) The computation of the session key consists of an exponentiation with a 1024-bit exponent (the private key). Considering the application of the square-and-multiply algorithm and assuming a uniform distribution of the exponent bit values, i.e., 512 bit equal to 1 and 512 bit equal to 0, we know that we need to perform:

- 1024 squarings $\longrightarrow$ 409.6 ms
- 512 multiplications $\longrightarrow$ 358.4 ms

So, on average, we would need:

$$t = 409.6 + 358.4 = 768\ ms$$

8 Cyclic Groups and Elgamal Encryption Scheme

8.1 Introduction to Finite Groups and Cyclic Groups

In cryptography we are almost always concerned with finite structures. For instance, for AES we needed a finite field. We provide now the straight forward definition of a finite group:

Definition Finite Group
A group $G, \circ$ is finite if it has a finite number of elements. We denote the cardinality or **order** *of the group G by $|G|$.*

Examples of finite groups are:

- $(\mathbb{Z}_n, +)$ - **additive group**: the cardinality of $\mathbb{Z}_n$ is $|\mathbb{Z}_n| = n$ since $\mathbb{Z}_n = \{0, 1, 2, ..., n-1\}$;
- $(\mathbb{Z}_n^*, \cdot)$ - **multiplicative group**: remember that $\mathbb{Z}_n^*$ is defined as the set of positive integers smaller than n which are relatively prime to n. Thus, the cardinality of $\mathbb{Z}_n^*$ equals Euler's phi function evaluated for n, i.e., $|\mathbb{Z}_n^*| = \Phi(n)$. For instance, the group $\mathbb{Z}_9^*$ has a cardinality of $\Phi(9) = 3^2 - 3^1 = 6$;

The reminder of this section deals with a special type of groups, namely cyclic groups, which are the basis for discrete logarithm-based cryptosystems. We start with the following definition:

Definition Order of an element
The order $\text{ord}(a)$ *of an element a of a group $(G, \circ)$ is the smallest positive integer k such that*

$$a^k = \underbrace{a \circ a \circ ... \circ a}_{k \text{ times}} = \mathbf{1}$$

where $\mathbf{1}$ *is the identity element of G (0 for additive groups and 1 for multiplicative groups).*

Example) We try to determine the order of $a = 3$ in the group $\mathbb{Z}_{11}^*$. For this, we keep computing powers of a until we obtain the identity element $\mathbf{1} = 1$ (because this is a multiplicative group).

$$\begin{aligned}
a^1 &= 3 \\
a^2 &= a \cdot a \ = 3 \cdot 3 = 9 \\
a^3 &= a^2 \cdot a = 9 \cdot 3 = 27 \equiv 5 \bmod 11 \\
a^4 &= a^3 \cdot a = 5 \cdot 3 = 15 \equiv 4 \bmod 11 \\
a^5 &= a^4 \cdot a = 4 \cdot 3 = 12 \equiv 1 \bmod 11
\end{aligned}$$

From the last line it follows that $\text{ord}(3) = 5$.

It is very interesting to look at what happens if we keep multiplying the result by a: from this point on, the powers of a run through the sequence $\{3, 9, 5, 4, 1\}$ indefinitely. This cyclic behaviour gives rise to the following definition:

Definition Cyclic Group
A group G which contains an element α with maximum order $\text{ord}(\alpha) = |G|$ *is said to be cyclic. Elements with maximum order are called primitive elements or* **generators**.

An element α of a group G with maximum order is called a generator since every element a of G can be written as a power $\alpha^i = a$ of this element for some i, i.e., α generates the entire group.

Example) We want to check whether $a = 2$ happens to be a generator of $\mathbb{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Note that the cardinality of the group is $|\mathbb{Z}_{11}^*| = 10$. Let's look at all the elements that are generated by powers of the element $a = 2$:

$$\begin{aligned} a &= 2 & a^6 &\equiv 9 \bmod 11 \\ a^2 &= 4 & a^7 &\equiv 7 \bmod 11 \\ a^3 &= 8 & a^8 &\equiv 3 \bmod 11 \\ a^4 &\equiv 5 \bmod 11 & a^9 &\equiv 6 \bmod 11 \\ a^5 &\equiv 10 \bmod 11 & a^{10} &\equiv 1 \bmod 11 \end{aligned}$$

From the last result it follows that

$$\text{ord}(a) = 10 = |\mathbb{Z}_{11}^*|.$$

This implies that $a = 2$ is a generator and $\mathbb{Z}_{11}^*$ is cyclic.

We now want to verify whether the powers of $a = 2$ actually generate all elements of the group $\mathbb{Z}_{11}^*$. Let's look again at all the elements that are generated by powers of 2:

i	1	2	3	4	5	6	7	8	9	10
a^i	2	4	8	5	10	9	7	3	6	1

By looking at the bottom row, we see that the powers 2^i in fact generate all elements of the group $\mathbb{Z}_{11}^*$. We note that the order in which they are generated looks quite arbitrary. This seemingly random relationship between the exponent i and the group elements is the basis for cryptosystems such as the Diffie-Hellman key exchange.

Cyclic groups have interesting properties. The most important ones for cryptographic applications are given in the following theorems:

Theorem *For every prime p, $(\mathbb{Z}_p^*, \cdot)$ is a commutative finite cyclic group.*

Remember that *commutative groups* are groups where the result of applying the group operation to two elements does not depend on the order in which such elements are written. The theorem above states that the multiplicative group of every prime field is cyclic. In order to understand the practical relevance of this theorem, consider that almost every Web browser has a cryptosystem over $\mathbb{Z}_p^*$ built in.

Theorem *Let G be a finite cyclic group. Then, for every $a \in G$ it holds that:*

- $a^{|G|} = \mathbf{1}$;
- $\text{ord}(a)$ *divides* $|G|$;

The first property is a generalization of Fermat's Little Theorem for all cyclic groups. The second property is very used in practice. It says that in a cyclic group only element orders which divide the group cardinality exist.

Theorem *Let G be a finite cyclic group. Then it holds that:*

- *The number of generators of G is $\Phi(|G|)$;*
- *If $|G|$ is prime, then all elements $a \neq \mathbf{1} \in G$ are generators;*

The second property follows from the previous theorem. If the group cardinality is prime, the only possible elements orders are 1 and the cardinality itself. Since only the element 1 can have an order of 1, all other elements have order p.

8.1.1 Subgroups

In this section we consider subsets of cyclic groups which are groups themselves. Such sets are referred to as *subgroups*. In order to check whether a subset H of a group G is a subgroup, one can verify if all the properties of our group definition also hold for H. In the case of cyclic groups, there is an easy way to generate subgroups which follows from this theorem:

Theorem ***Cyclic Subgroup Theorem***
Let $(G, \circ)$ be a cyclic group. Then every element $a \in G$ with $\text{ord}(a) = s$ is the generator of a cyclic subgroup with s elements.

An important special case are subgroups of prime order. If this groups' cardinality is denoted by q, all non-one elements have order q according to the last theorem introduced in the previous paragraph. From the Cyclic Subgroup Theorem we know that each element $a \in G$ of a group G generates some subgroup H. Since for every $a \in G$, it holds that $\text{ord}(a)$ divides $|G|$ (follows from the second theorem introduced in the previous paragraph), the following theorem follows:

Theorem Lagrange's theorem
Let H be a subgroup of G. Then $|H|$ divides $|G|$.

Example) The cyclic group $\mathbb{Z}_{11}^*$ has cardinality $|\mathbb{Z}_{11}^*| = 10 = 1 \cdot 2 \cdot 5$. Thus, it follows that the subgroups of $\mathbb{Z}_{11}^*$ have cardinalities 1,2,5 and 10 ($\mathbb{Z}_{11}^*$ itself) since these are all possible divisors of 10. All subgroups H of $\mathbb{Z}_{11}^*$ and their generators α are given below:

subgroup	elements	generators
H_1	{1}	$\alpha = 1$
H_2	{1,10}	$\alpha = 10$
H_3	{1,3,4,5,9}	$\alpha = 3, 4, 5, 9$

The following final theorem of this section fully characterizes the subgroups of a finite cyclic group:

Theorem
Let G be a finite cyclic group of order n and let α be a generator of G. Then, for every integer k that divides n, there exists exactly one cyclic subgroup H of G of order k. This subgroup is generated by $\alpha^{n/k}$. H consists exactly of the elements $a \in G$ which satisfy the condition $a^k = 1$. There are no other subgroups.

This theorem gives us a construction method for a subgroup from a given cyclic group. The only thing we need is a generator α and the group cardinality n. One can now simply compute $\alpha^{n/k}$ and obtains a generator of the subgroup with k elements.

8.2 The Elgamal Encryption Scheme

The *Elgamal encryption scheme*, proposed by Taher Elgamal in 1985, can be viewed as an extension of the DHKE protocol. We consider the Elgamal encryption scheme over the group $\mathbb{Z}_p^*$, where p is a prime. However, it can be applied to other cyclic groups too in which the DL and DH problems are intractable, for instance, in the multiplicative group of a Galois field $GF(2^m)$.

8.2.1 From Diffie-Hellman Key Exchange to Elgamal Encryption

In order to understand the Elgamal scheme, it is very helpful to see how it follows almost immediately from the DHKE. We consider two parties, Alice and Bob. If Alice wants to send an encrypted message x to Bob, both parties first perform a Diffie-Hellman key exchange to derive a shared key k_M. For this we assume that a large prime p and a generator α have been generated. Now, the new idea is that Alice uses this key as a multiplicative mask to encrypt x as $y \equiv x \cdot k_M \bmod p$.

Principle of Elgamal Encryption

Alice		**Bob**
		1. choose $d = k_{prB} \in \{2, ..., p-2\}$ 2. compute $\beta = k_{pubB} \equiv \alpha^d \bmod p$
	β ←	
3. choose $i = k_{prA} \in \{2, ..., p-2\}$ 4. compute $k_E = k_{pubA} \equiv \alpha^i \bmod p$		
	k_E →	
5. compute $k_M \equiv \beta^i \bmod p$ 6. encrypt message $x \in \mathbb{Z}_p^*$: $y \equiv x \cdot k_M \bmod p$		5. compute $k_M \equiv k_E^d \bmod p$
	y →	
		7. decrypt $x \equiv y \cdot k_M^{-1} \bmod p$

The protocol consists of two phases:

1. The classical DHKE (steps 1-5):
   - Bob computes his private key d and public key β. This key pair does not change;
   - Alice, however, has to generate a new public-private key pair for the encryption of every message. Her private key is denoted by i and her public key by k_E, which stands for "ephemeral key" (existing only temporarily). Reusing the same k_E would allow an attacker to break the scheme;
   - The joint key is denoted by k_M because it is used for masking the plaintext;
2. The message encryption and decryption (steps 6 and 7, respectively):
   - For the encryption, Alice multiplies the plaintext x by the masking key k_M in $\mathbb{Z}_p^*$;
   - Bob reverses the encryption by multiplying the ciphertext y with the inverse mask k_M^{-1};

8.3 Past exams exercises (Groups)

8.3.1 2023-07-03 ex. 2

Compute the order of $5 \in \mathbb{Z}_{47}^*$.

Solution

Using a "brute force" approach would be time consuming. So, let's notice that, since $\mathbb{Z}_{47}^*$ is in the form $\mathbb{Z}_p^*$ with p prime, it holds that it's a commutative finite cyclic group. So, it holds that $ord(5)$ divides $|\mathbb{Z}_{47}^*| = 46$. This reduces our search field to the following set:

$$ord(5) \in \{1, 2, 23, 46\}$$

We can immediately exclude 1 and 2, because:

$$5^1 \equiv 5 \not\equiv 1 \bmod 47$$

$$5^2 \equiv 25 \not\equiv 1 \bmod 47$$

We know for sure that:

$$5^{46} \equiv 1 \bmod 47$$

because in a cyclic group G_m it holds that $a^{|G|} \equiv 1 \bmod m,\ a \in G_m$.

So, the only thing that's left for us to check is: $5^{23} \stackrel{?}{\equiv} 1 \bmod 47$

- if yes, then $ord(5) = 23$;
- if no, then $ord(5) = 46$;

because the order of an element a is the **lowest** exponent e such that $a^e \equiv 1 \bmod m$.

Let's compute:

$$\begin{aligned} 5^{23} &= \left(5^3\right)^7 \cdot 5^2 = 125^7 \cdot 5^2 \equiv \\ &\equiv 31^7 \cdot 5^2 = \left(31^2\right)^3 \cdot 31 \cdot 5^2 = 961^3 \cdot 31 \cdot 5^2 \equiv \\ &\equiv 21^3 \cdot 31 \cdot 5^2 = 21^3 \cdot 775 = 9261 \cdot 775 = 2 \cdot 23 \equiv 46 \bmod 47 \end{aligned}$$

Since $5^{23} \equiv 46 \not\equiv 1 \bmod 47$:

$$ord(5) = 46$$

8.4 Exercises from slides (Groups)

8.4.1 Exercise 6.1.14

Find $x \in \mathbb{Z}_{44}$ of order 3.

Solution

Since we're expecting x to have $ord(x) = 3$, we're expecting it to generate a subgroup of $\mathbb{Z}_{44}$ with exactly 3 elements. Let's call this subgroup H, with $|H| = 3$. From Lagrange's theorem we know that the cardinality of each subgroup of a group G divides $|G|$. In our case, that means that the subgroups of $\mathbb{Z}_{44}$ have the following cardinalities:

$$|H_i| \in \{1, 2, 4, 11, 22, 44\}$$

since these are all the divisors of 44. By excluding the *trivial subgroup*, i.e., the subgroup with just the identity element $\mathbf{1} = 0$ (it's 0 because $\mathbb{Z}_{44}$ is an additive group), and the subgroup with 44 elements, that is $\mathbb{Z}_{44}$ itself, we're left with 4 possible subgroups with cardinalities:

$$|H_i| \in \{2, 4, 11, 22\}$$

As we can see, there's no subgroup with cardinality equal to 3. Hence, no element $x \in \mathbb{Z}_{44}$ with $ord(x) = 3$ can exist. As a matter of fact, these are all the elements of $\mathbb{Z}_{44}$ for which the order is defined:

$ord(1) = 44$	$ord(12) = 11$	$ord(23) = 44$	$ord(34) = 22$
$ord(2) = 22$	$ord(13) = 44$	$ord(24) = 11$	$ord(35) = 44$
$ord(3) = 44$	$ord(14) = 22$	$ord(25) = 44$	$ord(36) = 11$
$ord(4) = 11$	$ord(15) = 44$	$ord(26) = 22$	$ord(37) = 44$
$ord(5) = 44$	$ord(16) = 11$	$ord(27) = 44$	$ord(38) = 22$
$ord(6) = 22$	$ord(17) = 44$	$ord(28) = 11$	$ord(39) = 44$
$ord(7) = 44$	$ord(18) = 22$	$ord(29) = 44$	$ord(40) = 11$
$ord(8) = 11$	$ord(19) = 44$	$ord(30) = 22$	$ord(41) = 44$
$ord(9) = 44$	$ord(20) = 11$	$ord(31) = 44$	$ord(42) = 22$
$ord(10) = 22$	$ord(21) = 44$	$ord(32) = 11$	$ord(43) = 44$
$ord(11) = 4$	$ord(22) = 2$	$ord(33) = 44$	

Remember that, since $\mathbb{Z}_{44}$ is an additive group, the group operation is the addition + modulo 44. In other words, the order of an element $a \in \mathbb{Z}_{44}$ is the smallest positive integer k such that:

$$\underbrace{a + a + \cdots + a}_{k \text{ times}} = k \cdot a \equiv \mathbf{1} \equiv 0 \bmod 44$$

8.5 Exercises from slides (Elgamal Cipher)

8.5.1 Exercise 9.2.4

Consider an Elgamal Cipher with $p = 83$ and $g = 4$. Encipher $m = (11101)_2$ with $A = 37$.
Note: the slides actually use a different notation: $g = \alpha$, $m = x$, $A = d$.

Solution

First of all, let's convert x from binary to decimal:

$$x = (11101)_2 = 29$$

In order to encrypt x we need to compute the masking key first:

1. Compute $\beta \equiv \alpha^d \bmod p$:

$$\begin{aligned}\beta = 4^{37} = \left(4^4\right)^9 \cdot 4 = 256^9 \cdot 4 \equiv \\ \equiv 7^9 \cdot 4 = \left(7^3\right)^3 \cdot 4 = 343^3 \cdot 4 \equiv \\ \equiv 11^3 \cdot 4 = 121 \cdot 11 \cdot 4 \equiv \\ \equiv 38 \cdot 11 \cdot 4 = 418 \cdot 4 \equiv \\ \equiv 3 \cdot 4 \equiv 12 \bmod 83\end{aligned}$$

2. Choose random $i \in \{2, ..., p-2\}$:

$$i = 5$$

3. Compute $k_M \equiv \beta^i \bmod p$:

$$k_M = 12^5 = \left(12^2\right)^2 \cdot 12 = 144^2 \cdot 12 \equiv 61^2 \cdot 12 = 3721 \cdot 12 \equiv 69 \cdot 12 = 828 \equiv 81 \bmod 83$$

4. Compute $y \equiv x \cdot k_M \bmod p$:

$$y = 29 \cdot 81 = 2349 \equiv 25 \bmod 83$$

For completeness, let's perform also the decryption:

1. The sender computes $k_E \equiv \alpha^i \bmod p$:

$$k_E = 4^5 = 1024 \equiv 28 \bmod 83$$

2. The receiver computes $k_M \equiv k_E^d \bmod p$:

$$\begin{aligned}k_M = 28^{37} = 4^{37} \cdot 7^{37} \equiv 12 \cdot 7^{37} = \left(7^4\right)^9 \cdot 7 \cdot 12 = 2401^9 \cdot 84 \equiv \\ \equiv 77^9 \equiv (-6)^9 = -6^9 = -216^3 \equiv \\ \equiv -50^3 = -1 \cdot 125000 \equiv -1 \cdot 2 \equiv \\ \equiv 82 \cdot 2 = 164 \equiv 81 \bmod 83\end{aligned}$$

   Note that the receiver has computed the same value of k_M;

3. The receiver computes $x \equiv y \cdot k_M^{-1} \bmod p$:

$$x = 25 \cdot 81^{-1} \equiv 25 \cdot 41 = 1025 \equiv 29 \bmod 83 \checkmark$$

   The inverse $81^{-1} \bmod 83$ computation with the EEA has been skipped.

9 Digital Signature Algorithm (DSA)

9.1 The Elgamal Digital Signature Scheme

The Elgamal signature scheme is based on the difficulty of computing discrete logarithms.

9.1.1 Key generation

There is a set-up phase during which the keys are computed. We construct a discrete logarithm problem as follows:

Key Generation for Elgamal Digital Signature

1. Choose a large prime p;
2. Choose a generator α of $\mathbb{Z}_p^*$ or a subgroup of $\mathbb{Z}_p^*$;
3. Choose a random integer $d \in \{2, 3, ..., p-2\}$;
4. Compute $\beta = \alpha^d \bmod p$;

The public key is now formed by $k_{pub} = (p, \alpha, \beta)$, and the private key by $k_{pr} = d$.

9.1.2 Signature and verification

Using the private key and the parameters of the public key, the signature

$$\text{sig}_{k_{pr}}(x, k_E) = (r, s)$$

for a message x is computed during the signing process. Note that the signature consists of two integers r and s. The signing consists of two main steps: choosing a random value k_E, which forms an ephemeral private key, and computing the actual signature of x:

Elgamal Signature Generation

1. Choose a random ephemeral key $k_E \in \{0, 1, 2, ..., p-2\}$ such that $\gcd(k_E, p-1) = 1$;
2. Compute the signature parameters:

$$r \equiv \alpha^{k_E} \bmod p,$$
$$s \equiv (x - d \cdot r) k_E^{-1} \bmod p - 1.$$

On the receiving side, the signature is verified as $\text{ver}_{k_{pub}}(x, (r, s))$ using a verification function which receives as input parameters the public key (of the signer), the signature and the message:

Elgamal signature verification

1. Compute the value

$$t \equiv \beta^r \cdot r^s \bmod p$$

2. The verification follows from:

$$t \begin{cases} \equiv \alpha^x \bmod p \longrightarrow \text{valid signature} \\ \not\equiv \alpha^x \bmod p \longrightarrow \text{invalid signature} \end{cases}$$

9.1.3 Computational aspects

Because the security of the signature scheme relies on the discrete logarithm problem:

- p should have length of at least 1024 bits;
- the private key should be generated by a true random number generator;
- the public key requires one exponentiation using the square-and-multiply algorithm;

The signature consists of the pair (r, s). Both have roughly the same bit length as p, so that the total length of the package $(x, (r, s))$ is about three times as long as only the message x. Computing r requires an exponentiation modulo p, which can be achieved with the square-and-multiply algorithm. The main operation when computing s is the inversion of k_E. This can be done using the Extended Euclidean Algorithm.

9.2 DSA

The native Elgamal signature algorithm described in this section is rarely used in practice. Instead, a much more popular variant is used, known as the *Digital Signature Algorithm (DSA)*. Its main advantages over the Elgamal signature scheme are that the signature is only 320-bit long and that some of the attacks that can threaten the Elgamal scheme are not applicable.

9.2.1 Algorithm explanation

We introduce here the DSA standard with a bit length of 1024 bits (longer lengths are also possible in the standard).

Key Generation

The keys for DSA are computed as follows:

Key Generation for DSA

1. Generate a prime p with $2^{1023} < p < 2^{1024}$;
2. Find a prime divisor q of $p - 1$ with $2^{159} < q < 2^{160}$;
3. Find an element $\alpha \in \mathbb{Z}_p^*$ with $ord(\alpha) = q$, i.e., α generates the subgroup with q elements;
4. Choose a random integer d with $0 < d < q$;
5. Compute $\beta \equiv \alpha^d \bmod p$;
6. The keys are now:

$$k_{pub} = (p, q, \alpha, \beta)$$
$$k_{pr} = d$$

The central idea of DSA is that there are two cyclic groups involved. One is the large cyclic group $\mathbb{Z}_p^*$, whose order has bit length of 1024 bits. The second one is in the 160-bit subgroup of $\mathbb{Z}_p^*$. This set-up yields shorter signatures, as we see in the following.

In addition to the 1024-bit prime p and a 160-bit prime q, there are two other bit length combinations possible for the primes p and q. If one of the other bit lengths is required, only steps 1 and 2 of the key generation phase have to be adjusted accordingly.

According to the latest version of the standard, the following combinations are allowed:

p	q	Signature
1024	160	320
2048	224	448
3072	256	512

Signature and Verification

As in the Elgamal scheme, the DSA signature scheme consists of a pair of integers (r, s). Since each of the two parameters is only 160-bit long, the total signature length is 320 bit. Using the public and private key, the signature for a message x is computed as follows:

DSA Signature Generation

1. Choose an integer as random ephemeral key k_E with $0 < k_E < q$;
2. Compute $r \equiv (\alpha^{k_E} \bmod p) \bmod q$;
3. Compute $s \equiv (SHA(x) + d \cdot r)k_E^{-1} \bmod q$;
4. The signature of the message x with the secret key d is:

$$sig_d(x, k_E) = (r, s)$$

According to the standard, the message x has to be hashed using the hash function SHA-1 in order to compute s. SHA-1 compresses x and computes a 160-bit fingerprint. The signature verification process is as follows:

DSA Signature Verification

1. Compute auxiliary value $w \equiv s^{-1} \bmod q$;
2. Compute auxiliary value $u_1 \equiv w \cdot SHA(x) \bmod q$;
3. Compute auxiliary value $u_2 \equiv w \cdot r \bmod q$;
4. Compute $v \equiv (\alpha^{u_1} \cdot \beta^{u_2} \bmod p) \bmod q$;
5. The verification $ver_{k_{pub}}(x, (r, s))$ follows from:

$$v \begin{cases} \equiv r \bmod q \longrightarrow \text{valid signature} \\ \not\equiv r \bmod q \longrightarrow \text{invalid signature} \end{cases}$$

Example) Bob wants to send a message x to Alice that is to be signed with the DSA. Suppose the hash value of x is $h(x) = 26$. Then, the signature and verification process is as follows:

Alice		**Bob**
		key generation:
		choose $p = 59$;
		choose $q = 29$;
		choose $\alpha = 3$;
		choose private key $d = 7$;
		compute $\beta = \alpha^d \equiv 4 \bmod 59$;
	$\xleftarrow{(p,q,\alpha,\beta)=(59,29,3,4)}$	
		signature generation:
		compute hash of message $h(x) = 26$;
		choose ephemeral key $k_E = 10$;
		$r = (3^{10} \bmod 59) \equiv 20 \bmod 29$;
		$s = (26 + 7 \cdot 20) \cdot 3 \equiv 5 \bmod 29$;
	$\xleftarrow{(x,(r,s))=(x,(20,5))}$	
signature verification:		
$w = 5^{-1} \equiv 6 \bmod 29$		
$u_1 = 6 \cdot 26 \equiv 11 \bmod 29$		
$u_2 = 6 \cdot 20 \equiv 4 \bmod 29$		
$v = (3^{11} \cdot 4^4 \bmod 59) \bmod 29 = 20$		
$v \equiv r \bmod 29 \Longrightarrow$ valid signature		

9.2.2 Computational aspects

The most demanding part is the key-generation phase. However, this phase only has to be executed once at set-up time.

Key Generation

The challenge in the key-generation phase is to find a cyclic group $\mathbb{Z}_p^*$ with a bit length of 1024, and which has a prime subgroup in the range of 2^{160}. This condition is fulfilled if $p - 1$ has a prime factor q of 160 bits. The general approach to generating such parameters is to first find the 160-bit prime q and then to construct the larger prime p from it.

Signing

During signing we compute the parameters r and s. Computing r involves first evaluation $\alpha^{k_E} \bmod p$ using the square-and-multiply algorithm. Since k_E has only 160 bits, about 240 squarings and multiplications are required on average, even though the arithmetic is done with 1024-bit numbers. The result, which has also a length of 1024 bits, is then reduced to 160 bits by the operation (mod q). Computing s involves only 160-bit numbers. The most costly step is the inversion of k_E.

Verification

Computing the auxiliary parameters w, u_1 and u_2 only involves 160-bit operands, which makes verification relatively fast.

9.3 Past Exams Exercises (DSA)

9.3.1 2020-07-03 ex. 3

Alice generates a secret key $\text{SK}_\text{A} = d = 4$ and wants to generate a DS. Given that the prime numbers are $p = 11$ and $q = 5$, what is the public key?

Solution

As we've previously mentioned, the public key in the DSA is defined as $k_{pub} = (p, q, \alpha, \beta)$. Since we already know p and q, we must find α such that $\text{ord}(\alpha) = q$. Remember that:

- we're working in $\mathbb{Z}_p^* = \mathbb{Z}_{11}^* = \{1, 2, ..., 10\}$;
- the subgroup generated by α has cardinality equal to $q = 5$;

In order to find α we must find the element of $\mathbb{Z}_{11}^*$ that generates the subgroup. So, we must look for that element $\alpha \in \mathbb{Z}_{11}^*$ such that $\alpha^5 = \mathbf{1}$ and $\alpha \neq \mathbf{1}$. Let's test all elements starting from 2:

$$\begin{aligned} &2^5 = 32 \equiv 10 \bmod 11 \\ &3^5 = 3^3 \cdot 3^2 \equiv 5 \cdot 9 \equiv 1 \bmod 11 \\ &... \end{aligned}$$

So, 3 is a generator of the subgroup. We can indeed verify that:

$$\begin{aligned} &3^1 = 3 \\ &3^2 \equiv 9 \bmod 11 \\ &3^3 \equiv 5 \bmod 11 \\ &3^4 \equiv 4 \bmod 11 \\ &3^5 \equiv 1 \bmod 11 \end{aligned}$$

So the subgroup is $H = \{1, 3, 4, 5, 9\}$.

We can now compute β as:

$$\beta \equiv \alpha^d \bmod p \longrightarrow \beta = 3^4 \equiv 4 \bmod 11$$

In the end, the public key is:

$$k_{pub} = (11, 5, 3, 4)$$

9.3.2 2021-09-14 ex. 4

DSA algorithm. Given $p = 11$, $q = 5$ and $d = 4$, what is the public key?

a) (11,5,9,5);

b) (11,5,3,5);

c) (11,5,7,3);

d) (11,5,2,4);

e) (11,5,4,2);

Solution

The fastest way to solve the exercise is to test the given values of α and β to see if both the following conditions are satisfied:

- $\alpha^5 \equiv 1 \bmod 11$;
- $\beta \equiv \alpha^d \bmod 11$;

Let's test the conditions:

a) $9^5 = 9^2 \cdot 9^2 \cdot 9 \equiv 4 \cdot 4 \cdot 9 \equiv 5 \cdot 9 = 45 \equiv 1 \bmod 11$ ✓
$9^4 = 9^2 \cdot 9^2 \equiv 4 \cdot 4 \equiv 5 \bmod 11$ ✓

b) $3^5 = 3^3 \cdot 3^2 \equiv 5 \cdot 9 \equiv 1 \bmod 11$ ✓
$3^4 = 3^3 \cdot 3 \equiv 5 \cdot 3 \equiv 4 \not\equiv 5 \bmod 11$

c) $7^5 = 7^2 \cdot 7^2 \cdot 7 \equiv 5 \cdot 5 \cdot 7 \equiv 3 \cdot 7 \equiv 10 \not\equiv 1 \bmod 11$

d) $2^5 = 32 \equiv 10 \not\equiv 1 \bmod 11$

e) $4^5 = 4^2 \cdot 4^2 \cdot 4 \equiv 5 \cdot 5 \cdot 4 \equiv 3 \cdot 4 \equiv 1 \not\equiv 2 \bmod 11$

So, the right answer is (a).

9.4 Exercises from slides (DSA)

9.4.1 Exercise 10.4.6

Set $p = 59$, $q = 29$, $\alpha = 3$, $d = 7$, $\beta = \alpha^d \bmod 59$. Assuming that $SHA(x) = 26$ compute the DSA $sig(r, s)$.

Solution

Let's compute r first. Let's choose a random value for the ephemeral key: $0 < k_E = 21 < q$

$$\begin{aligned}
r &\equiv (\alpha^{k_E} \bmod p) \bmod q = \\
&= (3^{21} \bmod 59) \bmod 29 = \\
&= \left[(3^4)^5 \cdot 3 \bmod 59\right] \bmod 29 = \\
&= \left(81^5 \cdot 3 \bmod 59\right) \bmod 29 \equiv \\
&\equiv \left(22^5 \cdot 3 \bmod 59\right) \bmod 29 = \\
&= \left(22^2 \cdot 22^2 \cdot 22 \cdot 3 \bmod 59\right) \bmod 29 \equiv \\
&\equiv (12 \cdot 12 \cdot 22 \cdot 3 \bmod 59) \bmod 29 = \\
&= (144 \cdot 66 \bmod 59) \bmod 29 \equiv \\
&\equiv (26 \cdot 7 \bmod 59) \bmod 29 \equiv \\
&\equiv (5 \bmod 59) \bmod 29 \equiv \\
&\equiv 5 \bmod 29
\end{aligned}$$

Now, let's compute s:

$$\begin{aligned}
s &\equiv (SHA(x) + d \cdot r)k_E^{-1} \bmod q = \\
&= (26 + 7 \cdot 5)21^{-1} \bmod 29 = \\
&= 61 \cdot 21^{-1} \bmod 29 \equiv \\
&\equiv 3 \cdot 18 \bmod 29 = \\
&= 25 \bmod 29
\end{aligned}$$

Note that the inverse $21^{-1} \bmod 29$ can be calculated with the EEA as explained in the dedicated chapter.

In the end, the signature is:

$$sig = (5, 25)$$

For completeness, let's perform the verification of the signature:

- Computation of the hash of the message (already provided by the text): $SHA(x) = 26$;
- Auxiliary values computation:

$$\begin{aligned}
w &\equiv s^{-1} \bmod q && \longrightarrow w = 25^{-1} \equiv 7 \bmod 29 \\
u_1 &\equiv w \cdot SHA(x) \bmod q && \longrightarrow u_1 = 7 \cdot 26 = 182 \equiv 8 \bmod 29 \\
u_2 &\equiv w \cdot r \bmod q && \longrightarrow u_2 = 7 \cdot 5 = 35 \equiv 6 \bmod 29
\end{aligned}$$

- Final computation + verification:

$$\begin{aligned}
v &\equiv (\alpha^{u_1} \cdot \beta^{u_2} \bmod p) \bmod q \equiv \\
&\equiv (3^8 \cdot 3^{42} \bmod 59) \bmod 29 = (3^{50} \bmod 59) \bmod 29 \equiv \\
&\equiv (5 \bmod 59) \bmod 29 \equiv 5 \bmod 29 \equiv r \ \checkmark
\end{aligned}$$

10 Elliptic Curve Cryptography (ECC)

ECC provides the same level of security as RSA or Discrete Logarithm (DL) systems with considerably shorter operands (approximately 160-256 bit vs. 1024-3072 bit). ECC is based on the generalized discrete logarithm problem. ECC has performance advantages (fewer computations) and bandwidth advantages (shorter signatures and keys) over RSA and DL schemes. However, RSA operations which involve short public keys are still much faster than ECC operations.

10.1 The Generalized Discrete Logarithm Problem (GDLP)

Definition Discrete Logarithm Problem (DLP) in $\mathbb{Z}_p^*$
Given the finite cyclic group $\mathbb{Z}_p^*$ of order $p-1$ and a generator $\alpha \in \mathbb{Z}_p^*$ and another element $\beta \in \mathbb{Z}_p^*$, the DLP is the problem of determining the integer $1 \leq x \leq p-1$ such that:

$$\alpha^x \equiv \beta \bmod p$$

Definition Generalized Discrete Logarithm Problem
Given a finite cyclic group G with the group operation $\circ$ and cardinality n. We consider a generator $\alpha \in G$ and another element $\beta \in G$. The GDLP is finding the integer x, where $1 \leq x \leq n$, such that:

$$\beta = \underbrace{\alpha \circ \alpha \circ ... \circ \alpha}_{x \text{ times}} = \alpha^x$$

10.2 Definition of Elliptic Curves

Since ECC is based on the GDLP, we need to find a cyclic group on which we can build our cryptosystem. Of course, the mere existence of a cyclic group is not sufficient, as the DL problem in this group must also be computationally hard.

An Elliptic Curve is a special type of polynomial equation. For cryptographic use, we need to consider the curve not over the real numbers but over a finite field. The most popular choice is Galois fields $GF(p)$, where all arithmetic is performed modulo a prime p.

Definition Elliptic Curve
The elliptic curve over $\mathbb{Z}_p$, $p > 3$, is the set of all pairs $(x, y) \in \mathbb{Z}_p$ which fulfill

$$y^2 \equiv x^3 + a \cdot x + b \bmod p$$

together with an imaginary **point of infinity** $\boldsymbol{O}$*, where*

$$a, b \in \mathbb{Z}_p$$

and the condition $4 \cdot a^3 + 27 \cdot b^2 \neq 0$ mod p.

The definition requires that the curve is nonsingular, i.e., the plot has no self-intersections or vertices, which is achieved if the discriminant of the curve $-16(4a^3 + 27b^2)$ is nonzero. For cryptographic use we are interested in studying the curve over a prime field as in the definition. However, if we plot such an elliptic curve over $\mathbb{Z}_p$, we do not get anything remotely resembling a curve. However, nothing prevents us from taking an elliptic curve equation and plotting it over the set of real numbers.

The first step for finding a group is identifying a set of elements for the group:

- the group elements are basically the points that fulfill the Elliptic Curve equation:

$$y^2 \equiv x^3 + a \cdot x + b \bmod p$$

Now we must define a new set of group operations with such elements...

10.3 Group Operations on Elliptic Curves

Let's denote the group operation with the addition symbol "+". "Addition" means that given two points and their coordinates, say $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, we have to compute the coordinates of a third point Q such that:

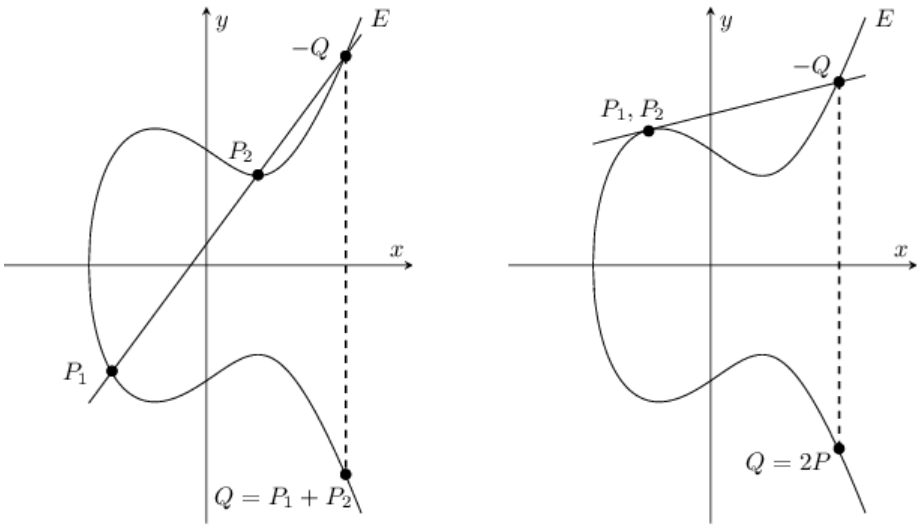
$$P_1 + P_2 = Q$$
$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$$

Keep in mind that the "+" symbol doesn't refer to the classic addition operation we're used to, as we're defining a new kind of operation (the group operation).

10.3.1 Point Addition and Point Doubling

There is a nice geometric interpretation of the addition operation if we consider a curve defined over the real numbers. For this interpretation, we have to distinguish two cases: the addition of two distinct points (named "point addition") and the addition of one point to itself (named "point doubling"):

- **Point Addition** This is the case where we compute $Q = P_1 + P_2$ and $P_1 \neq P_2$. The construction works as follows: draw a line through P_1 and P_2 and obtain a third point of intersection between the elliptic curve and the line. Mirror this third intersection point along the x-axis. This mirrored point is, by definition, the point Q;
- **Point Doubling** This is the case where we compute $P_1 + P_2$ but $P_1 = P_2$. Hence, we can write $Q = P + P = 2P$. We need a slightly different construction here. We draw the tangent line through P and obtain a second point of intersection along the x-axis. This mirrored point is the result Q of doubling;



(a) Point addition (b) Point doubling

If points on the elliptic curve are *added* in this very way, the set of points also fulfill most conditions necessary for a group, that is, closure, associativity, existence of an identity element and existence of an inverse. Of course, in a cryptosystem we cannot perform geometric constructions. However, by applying simple coordinate geometry, we can express both of the geometric constructions from above through analytic expressions. These formulae only involve the four basic algebraic operations.

These operations can be performed in any field, not only over the field of the real numbers. In particular, we can take the curve equation from above, but we now consider it over Galois fields $GF(p)$ rather than over the real numbers. This yields the following analytical expressions for the group operation.

Elliptic Curve Point Addition and Point Doubling

$$x_3 = s^2 - x_1 - x_2 \bmod p$$
$$y_3 = s(x_1 - x_3) - y_1 \bmod p$$

where

$$s = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \bmod p; \text{ if } P_1 \neq P_2 \text{ (point addition)} \\ \\ \dfrac{3x_1^2 + a}{2y_1} \bmod p; \text{ if } P_1 = P_2 \text{ (point doubling)} \end{cases}$$

Note that the parameter s is the slope of the line through P_1 and P_2 in the case of point addition, or the slope of the tangent through P in the case of point doubling.

10.3.2 Identity element

Even though we defined the operations above, we're not done yet. One thing that is still missing is an identity (or neutral) element $\boldsymbol{O}$ such that:

$$P + \boldsymbol{O} = P$$

for all points P on the elliptic curve. It turns out that there isn't any point (x, y) that fulfills the condition. Instead we define an *abstract point at infinity* as the neutral element $\boldsymbol{O}$. This point at infinity can be visualized as a point that is located either towards $+\infty$ or towards $-\infty$ along the y-axis.

10.3.3 Inverse element

We can now also define the inverse $-P$ of any group element P as:

$$P + (-P) = \boldsymbol{O}$$

The question is how do we find $-P$? If we apply the tangent-and-chord method from above, it turns out that the inverse of the point $P = (x_p, y_p)$ is the point $-P = (x_p, -y_p)$, i.e., the point that is reflected along the x-axis. Note that finding the inverse of a point $P = (x_p, y_p)$ is now trivial. We simply take the negative of its y coordinate. In the case of elliptic curves over a Galois field $GF(p)$, this is easily achieved since $-y_p \equiv p - y_p \bmod p$, hence:

$$-P = (x_p, p - y_p)$$

Now that we have defined all group properties for elliptic curves, we can look at an example:

Example) We consider a curve over the field $\mathbb{Z}_{17}$:

$$E : y^2 \equiv x^3 + 2x + 2 \bmod 17$$

We want to double the point $P = (5, 1)$.

$$\begin{aligned}
2P = P + P &= (5,1) + (5,1) = (x_3, y_3) \\
s &= \frac{3x_1^2 + a}{2y_1} = (2 \cdot 1)^{-1}(3 \cdot 5^2 + 2) = 2^{-1} \cdot 77 \equiv 9 \cdot 9 \equiv 13 \bmod 17 \\
x_3 &= s^2 - x_1 - x_2 = 13^2 - 5 - 5 = 159 \equiv 6 \bmod 17 \\
y_3 &= s(x_1 - x_3) - y_1 = 13(5 - 6) - 1 = -14 \equiv 3 \bmod 17 \\
2P &= (5,1) + (5,1) = (6,3)
\end{aligned}$$

For illustrative purposes we check whether the result $2P = (6, 3)$ is actually a point on the curve by inserting the coordinates into the curve equation:

$$\begin{aligned}
y^2 &\equiv x^3 + 2x + 2 \bmod 17 \\
3^2 &\equiv 6^3 + 2 \cdot 6 + 2 \bmod 17 \\
9 &\equiv 230 \equiv 9 \bmod 17 \checkmark
\end{aligned}$$

10.4 Building a Discrete Logarithm Problem with Elliptic Curves

What we have done so far is to establish the group operations (point addition and doubling), we have provided an identity element and we have shown a way of finding the inverse for any point on the curve. Thus, we now have all necessary requirements in place to motivate the following theorem:

Theorem *The points on an elliptic curve together with* $\boldsymbol{O}$ *have cyclic subgroups. Under certain conditions all points on an elliptic curve form a cyclic group.*

This theorem is extremely useful because we already have a good understanding of the properties of cyclic groups. In particular, we know that by definition a generator must exist such that its powers generate the entire group. Moreover, we know quite well how to build cryptosystems from cyclic groups. So, we can now have a look at an example for the cyclic group of an elliptic curve.

Example) We want to find all points on the curve:

$$E : y^2 \equiv x^3 + 2x + 2 \bmod 17.$$

It happens that all points on the curve form a cyclic group and that the order is $\#E = 19$.
For this specific curve the group order is a prime and so every element (point) is a generator. For simplicity we choose to compute all the curve points starting from the same point of the previous example, that is the element $P = (5, 1)$. So, we compute all "powers" of P. More precisely, since the group operation is addition, we compute $P, 2P, ..., (\#E)P$.

Here's what we obtain:

- $2P = (5,1) + (5,1) = (6,3)$;
- $3P = 2P + P = (6,3) + (5,1) = (10,6)$

  $$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{3-1}{6-5} = 2 \bmod 17$$
  $$x_3 = s^2 - x_1 - x_2 = 4 - 5 - 6 = -7 \equiv 10 \bmod 17$$
  $$y_3 = s(x_1 - x_3) - y_1 = 2(5-10) - 1 = -11 \equiv 6 \bmod 17$$

- $4P = (3,1)$
  - Calculated with point addition as $3P + P = (10,6) + (5,1)$:

    $$\begin{aligned} s = \frac{y_2 - y_1}{x_2 - x_1} &= \frac{6-1}{10-5} &&\equiv 1 \bmod 17 \\ x_3 = s^2 - x_1 - x_2 &= 1 - 5 - 10 = -14 &&\equiv 3 \bmod 17 \\ y_3 = s(x_1 - x_3) - y_1 &= (5-3) - 1 &&\equiv 1 \bmod 17 \end{aligned}$$

  - Calculated with point doubling as $2P + 2P = (6,3) + (6,3)$:

    $$\begin{aligned} s = \frac{3x^2 + a}{2y} &= \frac{108+2}{6} = 3^{-1} \cdot 55 \equiv 6 \cdot 4 \equiv 7 \bmod 17 \\ x_3 = s^2 - 2x &= 49 - 12 = 37 &&\equiv 3 \bmod 17 \\ y_3 = s(x - x_3) - y &= 7(6-3) - 3 = 18 &&\equiv 1 \bmod 17 \end{aligned}$$

    As we can see, by using point doubling we can focus on the coordinates of a single point (x, y) but the modular arithmetic could be a bit more complex (it obviously depends from case to case);

-

$$\begin{aligned} 5P &= (9,16) & 12P &= (0,11) \\ 6P &= (16,13) & 13P &= (16,4) \\ 7P &= (0,6) & 14P &= (9,1) \\ 8P &= (13,7) & 15P &= (3,16) \\ 9P &= (7,6) & 16P &= (10,11) \\ 10P &= (7,11) & 17P &= (6,14) \\ 11P &= (13,10) & 18P &= (5,16) \end{aligned}$$

- $19P = \boldsymbol{O}$
  - Calculated with point addition as $18P + P = (5,16) + (5,1)$:

    $$s = \frac{16-1}{5-5} = ?$$

    As we can see, we can't calculate s. That's because $18P$ and P are the inverse of each other. This is easy to verify. Since we've already checked that the two x coordinates are identical, we just have to check that the two y coordinates are each other's additive inverse modulo 17. As we've already seen, the inverse of a point $P = (x_p, y_p)$ is $-P = (x_p, -y_p)$. Let's compute the inverse of $18P$:

    $$-18P = (5,-16) \equiv (5,1) \bmod 17 = P$$

    So, $18P$ and P are indeed the inverse of each other. So:

    $$19P = 18P + P = \boldsymbol{O}$$

End of the Example.

To set up a DL cryptosystem it is important to know the order of the group. Even though knowing the exact number of points on a curve is an elaborate task, we know the approximate number due to *Hasse's theorem*:

Theorem Hasse's theorem
Given an elliptic curve E modulo p, the number of points on the curve is denoted by #E and is bounded by:

$$p + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p}$$

Hasse's theorem, which is also known as *Hasse's bound*, states that the number of points is roughly in the range of the prime p. This has major practical implications. For instance, if we need an elliptic curve with 2^{160} elements, we have to use a prime of length of about 160 bit.

Let's now turn our attention to the details of setting up the discrete logarithm problem. For this, we can strictly proceed as following:

Definition Elliptic Curved Discrete Logarithm Problem (ECDLP)
Given an elliptic curve E, we consider a generator P and another element T. The DL problem is finding the integer d, where $1 \leq d \leq \#E$*, such that:*

$$\underbrace{P + P + \cdots + P}_{d \text{ times}} = dP = T$$

In cryptosystems, d is the private key which is an integer, while the public key T is a point on the curve with coordinates $T = (x_T, y_T)$. In contrast, in the case of the DL problem in $\mathbb{Z}_p^*$, both keys were integers. The operation shown in the theorem above is called ***point multiplication***, since we can formally write $T = dP$. So, dP is merely a convenient notation for the repeated application of the group operation.

Point multiplication is analog to exponentiation in multiplicative groups. In order to do it efficiently, we can directly adopt the square-and-multiply algorithm. The only difference is that squaring becomes (point) doubling and multiplication becomes (point) addition of P. We can call this ***Double-and-Add algorithm***.

Example) We consider the scalar multiplication $26P$, which has the following binary representation:

$$26P = (11010_2)P = (d_4 d_3 d_2 d_1 d_0)_2 P$$

The algorithm scans the scalar bits starting on the left with d_4 and ending with the rightmost bit d_0. The initialization value for the algorithm is $\boldsymbol{O}$:

0) $d_4 = 1 \rightarrow res_0 = 2\boldsymbol{O} + P = P$

1) $d_3 = 1 \rightarrow res_1 = 2res_0 + P = 2P + P = 3P$

2) $d_2 = 0 \rightarrow res_2 = 2res_1 = 6P$

3) $d_1 = 1 \rightarrow res_3 = 2res_2 + P = 12P + P = 13P$

4) $d_0 = 0 \rightarrow res_4 = 2res_3 = 26P$

There's a nice geometric interpretation for the ECDLP: given a starting point P, we compute $2P, 3P, ..., dP = T$, effectively hopping back and forth on the elliptic curve. We then publish the starting point P (a public parameter) and the final point T (the public key). In order to break the cryptosystem, an attacker has to figure out how often we "jumped" on the elliptic curve. The number of hops is the secret d, the private key.

10.5 Elliptic Curve Protocols

10.5.1 Diffie-Hellman Key Exchange with Elliptic Curves (ECDH)

We can now realize a key exchange using elliptic curves. This is referred to as Elliptic Curve Diffie-Hellman key exchange, or ECDH. First, we have to agree on domain parameters, that is, a suitable elliptic curve over which we can work and a generator on this curve.

ECDH Domain Parameters

1. Choose a prime p and the elliptic curve

$$E : y^2 \equiv x^3 + a \cdot x + b \bmod p$$

2. Choose a generator $P = (x_P, y_P)$

The prime p, the curve given by its coefficients a, b, and the generator P are the domain parameters.

Note that, in practice, finding a suitable elliptic curve is a relatively difficult task. The curves have to show certain properties in order to be secure. The actual key exchange is done the same way it was done for the conventional Diffie-Hellman protocol.

Elliptic Curve Diffie-Hellman Key Exchange

Alice		**Bob**
choose $k_{prA} = a \in \{2, 3, ..., \#E - 1\}$		choose $k_{prB} = b \in \{2, 3, ..., \#E - 1\}$
compute $k_{pubA} = aP = A = (x_A, y_A)$		compute $k_{pubB} = bP = B = (x_B, y_B)$
	$\xrightarrow{A}$	
	$\xleftarrow{B}$	
compute $aB = T_{AB}$		compute $bA = T_{AB}$

Since point addition is associative, both parties compute the same result, namely the point $T_{AB} = abP$.

As we can see in the protocol, Alice and Bob choose the private keys a and b, respectively, which are two large integers. With the private keys they both generate their respective public keys A and B, which are points on the curve. The public keys are computed by point multiplication. The two parties exchange these public parameters with each other. The joint secret T_{AB} is then computed by both Alice and Bob by performing a second point multiplication involving the public key they received and their own secret parameter. The joint secret T_{AB} can be used to derive a session key, e.g., as input for the AES algorithm. Note that the two coordinates (x_{AB}, y_{AB}) are not independent of each other: given x_{AB}, the other coordinate can be computed by simply inserting the x value in the elliptic curve equation. Thus, only one of the two coordinates should be used for the derivation of a session key.

In practice, often the x-coordinate is hashed and then used as a symmetric key. Typically, not all bits are needed. For instance, in a 160-bit ECC scheme, hashing the x-coordinate with SHA-1 results in a 160-bit output of which only 128 bit would be used as an AES key.

10.5.2 Elliptic Curve Digital Signature Algorithm (ECDSA)

The steps in the ECDSA standard are conceptually closely related to the DSA scheme. However, its DLP is constructed in the group of an elliptic curve. Thus, the arithmetic to be performed for actually computing the signature is entirely different from that used for DSA. The ECDSA standard is defined for elliptic curves over prime fields $\mathbb{Z}_p$ and Galois Fields $GF(2^m)$. The former is often preferred in practice, and we will only introduce this one in what follows.

Key Generation

The keys for the ECDSA are computed as follows:

Key Generation for ECDSA

1. Use an elliptic curve E with
   - modulus p;
   - coefficients a and b;
   - a point A which generates a cyclic group of prime order q;
2. Choose a random integer d with $0 < d < q$;
3. Compute $B = dA$.

The keys are now:

$$k_{pub} = (p, a, b, q, A, B)$$

$$k_{pr} = (d)$$

Note that we have set up a DLP where the integer d is the private key and the result of the scalar multiplication, point B, is the public key. Similar to DSA, the cyclic group has an order q which should have a size of at least 160 bit or more for higher security levels.

Signature and Verification

Like DSA, an ECDSA signature consists of a pair of integers (r, s). Each value has the same bit length as q, which makes for fairly compact signatures. Using the public and private key, the signature for a message x is computed as follows:

ECDSA Signature Generation

1. Choose an integer as random ephemeral key k_E with $0 < k_E < q$;
2. Compute $R = k_E A$;
3. Let $r = x_R$;
4. Compute $s \equiv (h(x) + d \cdot r)k_E^{-1} \bmod q$;
5. The signature of the message x with the secret key d is:

$$sig_d(x, k_E) = (r, s)$$

In step 3 the x-coordinate of the point R is assigned to the variable r. The message x has to be hashed using the function h in order to compute s. The hash function output length must be at least as long as q.

The signature verification process is as follows:

ECDSA Signature Verification

1. Compute auxiliary value $w \equiv s^{-1} \bmod q$;
2. Compute auxiliary value $u_1 \equiv w \cdot h(x) \bmod q$;
3. Compute auxiliary value $u_2 \equiv w \cdot r \bmod q$;
4. Compute $P = u_1 A + u_2 B$;
5. The verification $ver_{k_{pub}}(x, (r, s))$ follows from:

$$x_P \begin{cases} \equiv r \bmod q \Longrightarrow \text{valid signature} \\ \not\equiv r \bmod q \Longrightarrow \text{invalid signature} \end{cases}$$

Example) Bob wants to send a message to Alice that is to be signed with the ECDSA. The signature and verification process is as follows:

Alice		**Bob**
		key generation: choose E with $p = 17$, $a = 2$, $b = 2$, and $A = (5, 1)$ with $q = 19$; choose $d = 7$; compute $B = dA = 7 \cdot (5, 1) = (0, 6)$;
	$\xleftarrow{\substack{(p,a,b,q,A,B)= \\ (17,2,2,19,(5,1),(0,6))}}$	
		signature generation: compute hash of message $h(x) = 26$; choose ephemeral key $k_E = 10$; $R = 10 \cdot (5, 1) = (7, 11)$; $r = x_R = 7$; $s = (26 + 7 \cdot 7) \cdot 2 \equiv 17 \bmod 19$;
	$\xleftarrow{(x,(r,s))=(x,(7,17))}$	
signature verification: $w = 17^{-1} \equiv 9 \bmod 19$ $u_1 = 9 \cdot 26 \equiv 6 \bmod 19$ $u_2 = 9 \cdot 7 \equiv 6 \bmod 19$ $P = 6 \cdot (5, 1) + 6 \cdot (0, 6) = (7, 11)$ $x_P \equiv r \bmod 19 \Longrightarrow$ valid signature		

Note that we chose the elliptic curve

$$E : y^2 \equiv x^3 + 2x + 2 \bmod 17$$

which we already used in previous examples, because all of its points form a cyclic group of order 19, i.e., a prime, there are no subgroups and hence in this case $q = \#E = 19$.

10.6 Past Exams Exercises (Elliptic Curves)

10.6.1 2022-06-27/2021-07-02 ex. 1 v.1

Let $E : y^2 \equiv x^3 + 2x + 2 \bmod 17$, $P = (5, 1)$ and $Q = (6, 3)$, such that $P + Q + R = \boldsymbol{O}$. Compute R.

Solution

It's straightforward that:

$$R = -P + (-Q)$$

Where $-P$ and $-Q$ are the inverses of P and Q:

$$-P = (5, -1) \equiv (5, 16) \bmod 17$$

$$-Q = (6, -3) \equiv (6, 14) \bmod 17$$

Let's perform the point addition

$$R = (5, 16) + (6, 14) = (x_R, y_R)$$

by computing all the necessary parameters:

$$\begin{aligned} s &= \frac{y_2 - y_1}{x_2 - x_1} &&= \frac{16 - 14}{5 - 6} = 1^{-1} \cdot (-2) \equiv 1^{-1} \cdot 15 \equiv 15 \bmod 17 \\ x_R &= s^2 - x_2 - x_1 &&= 15^2 - 5 - 6 &&= 214 &&\equiv 10 \bmod 17 \\ y_R &= s(x_1 - x_R) - y_1 &&= 15(6 - 10) - 14 &&= -74 &&\equiv 11 \bmod 17 \end{aligned}$$

So, we obtain:

$$R = (10, 11)$$

10.6.2 2021-07-02 ex. 1 v.2

Let $E : y^2 \equiv x^3 + 2x + 2 \bmod 17$, $P = (6, 3)$ and $Q = (10, 6)$, such that $P + Q + R = \boldsymbol{O}$. Compute R.

Solution

As in the previous exercise:

$$\begin{aligned} -P &= (6, -3) \equiv (6, 14) \bmod 17 \\ -Q &= (10, -6) \equiv (10, 11) \bmod 17 \end{aligned}$$

We perform the point addition

$$R = (6, 14) + (10, 11) = (x_R, y_R)$$

by computing all the necessary parameters:

$$\begin{aligned} s &= \frac{y_2 - y_1}{x_2 - x_1} &&= \frac{14 - 11}{6 - 10} = 4^{-1} \cdot (-3) \equiv 13 \cdot 14 = 182 \equiv 12 \bmod 17 \\ x_R &= s^2 - x_2 - x_1 &&= 12^2 - 6 - 10 &&= 128 &&\equiv 9 \bmod 17 \\ y_R &= s(x_1 - x_R) - y_1 &&= 12(10 - 9) - 11 &&= 1 &&\equiv 1 \bmod 17 \end{aligned}$$

So, we obtain:

$$R = (9, 1)$$

10.6.3 FacsimileCryptography 2020 ex.2/Random.pdf

Let $E : y^2 \equiv x^3 + 7$ be the elliptic curve defined on $\mathbb{Z}_{11}$.

(a) Check that $P = (2, 2)$ and $Q = (7, 3)$ are in E;

(b) Compute the addition $P + Q$ on the elliptic curve E;

(c) Check that your answer to (b) is a point of E;

Solution

(a) To check if a point P belongs to the curve, we simply check that the curve equation is still satisfied after replacing x, y with x_P, y_P.

- P belongs to the curve

$$4 \stackrel{?}{\equiv} 15 \longrightarrow 4 \equiv 4 \bmod 11 \checkmark$$

- Q belongs to the curve

$$9 \stackrel{?}{\equiv} 7 \cdot 7^2 + 7 = 7(7^2 + 1) \equiv 7 \cdot 6 = 42 \longrightarrow 9 \equiv 9 \bmod 11 \checkmark$$

(b) Let's compute the result of the point addition $R = (2, 2) + (7, 3) = (x_R, y_R)$.

$$\begin{aligned} s &= \frac{y_2 - y_1}{x_2 - x_1} &&= \frac{2-3}{2-7} &&= 5^{-1} \equiv 9 \bmod 11 \\ x_R &= s^2 - x_2 - x_1 &&= 81 - 2 - 7 &&= 72 \equiv 6 \bmod 11 \\ y_R &= s(x_1 - x_R) - y_1 &&= 9(7-6) - 3 = 6 &&\equiv 6 \bmod 11 \end{aligned}$$

So, we obtain:

$$R = P + Q = (6, 6)$$

(c) R belongs to the curve

$$36 \stackrel{?}{\equiv} 6 \cdot 6^2 + 7 \equiv 6 \cdot 3 + 7 = 25 \longrightarrow 3 \equiv 3 \bmod 11 \checkmark$$

10.7 Exercises from slides (Elliptic Curves)

10.7.1 Exercise 11.3.2

Let $G = (5, 1)$ be a point of $E: \ y^2 \equiv x^3 + 2x + 2 \mod 17$. Find:

- $P \in E$ such that $18P = (0, 6)$;
- $Q \in E$ such that $3Q = (3, 16)$;

Here's the table of all multiples of G:

$$
\begin{aligned}
G &= (5,1) & 10G &= (7,11)\\
2G &= (6,3) & 11G &= (13,10)\\
3G &= (10,6) & 12G &= (0,11)\\
4G &= (3,1) & 13G &= (16,4)\\
5G &= (9,16) & 14G &= (9,1)\\
6G &= (16,13) & 15G &= (3,16)\\
7G &= (0,6) & 16G &= (10,11)\\
8G &= (13,7) & 17G &= (6,14)\\
9G &= (7,6) & 18G &= (5,16)
\end{aligned}
$$

$$19G = \boldsymbol{O}$$

Solution

We've already encountered this curve when we defined the discrete logarithm problem for elliptic curves. So we know that:

- all points on the curve form a cyclic group and the order of such group is #E = 19;
- since the order of the curve is a prime number, every element (point) of E is a generator. This follows from *Lagrange's theorem* which states that the order of a subgroup H of group G divides the order of G.

  Therefore the only possible subgroups of a group of order p are the trivial group of order 1 and the group of order p, i.e., the group itself.

  This has a consequence, usually presented as a *lemma of Lagrange's theorem* - no element can generate a subgroup, so it must generate the full group;

Let's compute P first:

$$18P = 7G \ \longrightarrow \ P = 18^{-1} \cdot 7G \equiv 18 \cdot 7G \equiv 126G \mod 19$$

$$P = 126G = 6 \cdot 19G + 12G = \boldsymbol{O} + 12G = 12G = (0, 11)$$

Note that all computations are made (mod 19) because that is the order of the curve. It would be a mistake to make the computations modulo 17, since that is the modulus which is used to define the coordinates of the points belonging to the curve.

In my opinion, the text of this exercise is missing some fundamental info: we should be provided either with the order of the curve or with the info that G is a generator of the curve. Otherwise, how could we understand that 19 is the order of the curve? If G is a generic point, it's not always true that $ord(G) = \#E$.

Q can be found in the same way:

$$3Q = 15G \ \longrightarrow \ Q = 5G = (9, 16)$$

You can verify the results at: https://andrea.corbellini.name/ecc/interactive/modk-mul.html

10.8 Past Exams Exercises (ECDH)

10.8.1 2020-07-21 ex.1

Let $E(\mathbb{Z}_{17})$ be the elliptic curve given by the equation $y^2 \equiv x^3 + 7$. Alice and Bob use $G = (2, 7)$ as generator for a ECDH protocol to obtain a session key k. Alice's secret key is $\mathrm{SK_A} = 5$. Bob's secret key is $\mathrm{SK_B} = 12$. What of the following is the session key?

a) $(10, 15)$;

b) $(5, 9)$;

c) $(6, 6)$;

d) $(5, 8)$;

The following is the addition table on $E(\mathbb{Z}_{17})$ where ∞ is the neutral element:

+	∞	(1,5)	(1,12)	(2,7)	(2,10)	(3,0)	(5,8)	(5,9)	(6,6)	(6,11)	(8,3)	(8,14)	(10,2)	(10,15)	(12,1)	(12,16)	(15,4)	(15,13)
∞	∞	(1,5)	(1,12)	(2,7)	(2,10)	(3,0)	(5,8)	(5,9)	(6,6)	(6,11)	(8,3)	(8,14)	(10,2)	(10,15)	(12,1)	(12,16)	(15,4)	(15,13)
(1,5)	(1,5)	(2,10)	∞	(1,12)	(5,9)	(15,13)	(2,7)	(12,1)	(8,14)	(6,6)	(6,11)	(10,15)	(8,3)	(15,4)	(12,16)	(5,8)	(3,0)	(10,2)
(1,12)	(1,12)	∞	(2,7)	(5,8)	(1,5)	(15,4)	(12,16)	(2,10)	(6,11)	(8,3)	(10,2)	(6,6)	(15,13)	(8,14)	(5,9)	(12,1)	(10,15)	(3,0)
(2,7)	(2,7)	(1,12)	(5,8)	(12,16)	∞	(10,15)	(12,1)	(1,5)	(8,3)	(10,2)	(15,13)	(6,11)	(3,0)	(6,6)	(2,10)	(5,9)	(8,14)	(15,4)
(2,10)	(2,10)	(5,9)	(1,5)	∞	(12,1)	(10,2)	(1,12)	(12,16)	(10,15)	(8,14)	(6,6)	(15,4)	(6,11)	(3,0)	(5,8)	(2,7)	(15,13)	(8,3)
(3,0)	(3,0)	(15,13)	(15,4)	(10,15)	(10,2)	∞	(8,14)	(8,3)	(12,16)	(12,1)	(5,9)	(5,8)	(2,10)	(2,7)	(6,11)	(6,6)	(1,12)	(1,5)
(5,8)	(5,8)	(2,7)	(12,16)	(12,1)	(1,12)	(8,14)	(5,9)	∞	(10,2)	(15,13)	(3,0)	(8,3)	(15,4)	(6,11)	(1,5)	(2,10)	(6,6)	(10,15)
(5,9)	(5,9)	(12,1)	(2,10)	(1,5)	(12,16)	(8,3)	∞	(5,8)	(15,4)	(10,15)	(8,14)	(3,0)	(6,6)	(15,13)	(2,7)	(1,12)	(10,2)	(6,11)
(6,6)	(6,6)	(8,14)	(6,11)	(8,3)	(10,15)	(12,16)	(10,2)	(15,4)	(1,5)	∞	(1,12)	(2,10)	(2,7)	(5,9)	(3,0)	(15,13)	(12,1)	(5,8)
(6,11)	(6,11)	(6,6)	(8,3)	(10,2)	(8,14)	(12,1)	(15,13)	(10,15)	∞	(1,12)	(2,7)	(1,5)	(5,8)	(2,10)	(15,4)	(3,0)	(5,9)	(12,16)
(8,3)	(8,3)	(6,11)	(10,2)	(15,13)	(6,6)	(5,9)	(3,0)	(8,14)	(1,12)	(2,7)	(5,8)	∞	(12,16)	(1,5)	(10,15)	(15,4)	(2,10)	(12,1)
(8,14)	(8,14)	(10,15)	(6,6)	(6,11)	(15,4)	(5,8)	(8,3)	(3,0)	(2,10)	(1,5)	∞	(5,9)	(1,12)	(12,1)	(15,13)	(10,2)	(12,16)	(2,7)
(10,2)	(10,2)	(8,3)	(15,13)	(3,0)	(6,11)	(2,10)	(15,4)	(6,6)	(2,7)	(5,8)	(12,16)	(1,12)	(12,1)	∞	(8,14)	(10,15)	(1,5)	(5,9)
(10,15)	(10,15)	(15,4)	(8,14)	(6,6)	(3,0)	(2,7)	(6,11)	(15,13)	(5,9)	(2,10)	(1,5)	(12,1)	∞	(12,16)	(10,2)	(8,3)	(5,8)	(1,12)
(12,1)	(12,1)	(12,16)	(5,9)	(2,10)	(5,8)	(6,11)	(1,5)	(2,7)	(3,0)	(15,4)	(10,15)	(15,13)	(8,14)	(10,2)	(1,12)	∞	(8,3)	(6,6)
(12,16)	(12,16)	(5,8)	(12,1)	(5,9)	(2,7)	(6,6)	(2,10)	(1,12)	(15,13)	(3,0)	(15,4)	(10,2)	(10,15)	(8,3)	∞	(1,5)	(6,11)	(8,14)
(15,4)	(15,4)	(3,0)	(10,15)	(8,14)	(15,13)	(1,12)	(6,6)	(10,2)	(12,1)	(5,9)	(2,10)	(12,16)	(1,5)	(5,8)	(8,3)	(6,11)	(2,7)	∞
(15,13)	(15,13)	(10,2)	(3,0)	(15,4)	(8,3)	(1,5)	(10,15)	(6,11)	(5,8)	(12,16)	(12,1)	(2,7)	(5,9)	(1,12)	(6,6)	(8,14)	∞	(2,10)

Solution

First of all, let's compute the public keys of Alice and Bob (just for completeness sake):

- Alice's public key: $A = \mathrm{SK_A} G = 5G$
- Bob's public key: $B = \mathrm{SK_B} G = 12G$

So:

- Alice will compute: $k = \mathrm{SK_A} B = 5 \cdot 12G = 60G$
- Bob will compute: $k = \mathrm{SK_B} A = 12 \cdot 5G = 60G$

Note that we could have immediately multiplied the private keys of Alice and Bob to obtain the session key.

In order to compute the session key we can use the double-and-add algorithm:

$$k = 60G = (111100_2)G = (d_5 d_4 d_3 d_2 d_1 d_0)_2 G$$

0) $d_5 = 1 \rightarrow res_0 = 2\boldsymbol{O} + G = G = (2, 7)$

1) $d_4 = 1 \rightarrow res_1 = 2res_0 + G = 2G + G = (12, 16) + (2, 7) = 3G = (5, 9)$

2) $d_3 = 1 \rightarrow res_2 = 2res_1 + G = 6G + G = (5, 8) + (2, 7) = 7G = (12, 1)$

3) $d_2 = 1 \rightarrow res_3 = 2res_2 + G = 14G + G = (1, 12) + (2, 7) = 15G = (5, 8)$

4) $d_1 = 0 \rightarrow res_4 = 2res_3 = 30G = (5, 9)$

5) $d_0 = 0 \rightarrow res_5 = 2res_4 = 60G = (5, 8)$

Note that all doublings and additions are easily performed just by looking at the addition table!

In the end, we obtain:

$$k = 60G = (5, 8)$$

So, the correct answer is (d).

Speedup trick

Note that, by looking at the number of rows/columns in the table, we see that the elliptic curve we're using has exactly 18 elements (points). That means the order of the cyclic group our cryptosystem is based on is 18 too, i.e., $\#E = 18$. In other words, since G is a generator, we can tell for sure that $dG = \boldsymbol{O}$ where d is a multiple of 18. That means we can speed up our computations as following:

$$k = 60G = 54G + 6G = 3 \cdot 18G + 6G = \boldsymbol{O} + 6G = 6G$$

Computing $6G$ is faster than computing $60G$. We don't even have to use the double-and-add algorithm!

$$k = 6G = 2G + 2G + 2G = (12, 16) + (12, 16) + (12, 16) = (1, 5) + (12, 16) = (5, 8)$$

10.8.2 2022-07-19 ex.1

Let $E(\mathbb{Z}_{17})$ be the elliptic curve given by the equation $y^2 \equiv x^3 + 7$. Alice and Bob use $G = (2, 7)$ as generator for a ECDH protocol to obtain a session key k. Alice's secret key is $\mathrm{SK_A} = 5$. Bob's secret key is $\mathrm{SK_B} = 11$. What is the session key? The following is the addition table on $E(\mathbb{Z}_{17})$ where ∞ is the neutral element:

+	∞	(1,5)	(1,12)	(2,7)	(2,10)	(3,0)	(5,8)	(5,9)	(6,6)	(6,11)	(8,3)	(8,14)	(10,2)	(10,15)	(12,1)	(12,16)	(15,4)	(15,13)
∞	∞	(1,5)	(1,12)	(2,7)	(2,10)	(3,0)	(5,8)	(5,9)	(6,6)	(6,11)	(8,3)	(8,14)	(10,2)	(10,15)	(12,1)	(12,16)	(15,4)	(15,13)
(1,5)	(1,5)	(2,10)	∞	(1,12)	(5,9)	(15,13)	(2,7)	(12,1)	(8,14)	(6,6)	(6,11)	(10,15)	(8,3)	(15,4)	(12,16)	(5,8)	(3,0)	(10,2)
(1,12)	(1,12)	∞	(2,7)	(5,8)	(1,5)	(15,4)	(12,16)	(2,10)	(6,11)	(8,3)	(10,2)	(6,6)	(15,13)	(8,14)	(5,9)	(12,1)	(10,15)	(3,0)
(2,7)	(2,7)	(1,12)	(5,8)	(12,16)	∞	(10,15)	(12,1)	(1,5)	(8,3)	(10,2)	(15,13)	(6,11)	(3,0)	(6,6)	(2,10)	(5,9)	(8,14)	(15,4)
(2,10)	(2,10)	(5,9)	(1,5)	∞	(12,1)	(10,2)	(1,12)	(12,16)	(10,15)	(8,14)	(6,6)	(15,4)	(6,11)	(3,0)	(5,8)	(2,7)	(15,13)	(8,3)
(3,0)	(3,0)	(15,13)	(15,4)	(10,15)	(10,2)	∞	(8,14)	(8,3)	(12,16)	(12,1)	(5,9)	(5,8)	(2,10)	(2,7)	(6,11)	(6,6)	(1,12)	(1,5)
(5,8)	(5,8)	(2,7)	(12,16)	(12,1)	(1,12)	(8,14)	(5,9)	∞	(10,2)	(15,13)	(3,0)	(8,3)	(15,4)	(6,11)	(1,5)	(2,10)	(6,6)	(10,15)
(5,9)	(5,9)	(12,1)	(2,10)	(1,5)	(12,16)	(8,3)	∞	(5,8)	(15,4)	(10,15)	(8,14)	(3,0)	(6,6)	(15,13)	(2,7)	(1,12)	(10,2)	(6,11)
(6,6)	(6,6)	(8,14)	(6,11)	(8,3)	(10,15)	(12,16)	(10,2)	(15,4)	(1,5)	∞	(1,12)	(2,10)	(2,7)	(5,9)	(3,0)	(15,13)	(12,1)	(5,8)
(6,11)	(6,11)	(6,6)	(8,3)	(10,2)	(8,14)	(12,1)	(15,13)	(10,15)	∞	(1,12)	(2,7)	(1,5)	(5,8)	(2,10)	(15,4)	(3,0)	(5,9)	(12,16)
(8,3)	(8,3)	(6,11)	(10,2)	(15,13)	(6,6)	(5,9)	(3,0)	(8,14)	(1,12)	(2,7)	(5,8)	∞	(12,16)	(1,5)	(10,15)	(15,4)	(2,10)	(12,1)
(8,14)	(8,14)	(10,15)	(6,6)	(6,11)	(15,4)	(5,8)	(8,3)	(3,0)	(2,10)	(1,5)	∞	(5,9)	(1,12)	(12,1)	(15,13)	(10,2)	(12,16)	(2,7)
(10,2)	(10,2)	(8,3)	(15,13)	(3,0)	(6,11)	(2,10)	(15,4)	(6,6)	(2,7)	(5,8)	(12,16)	(1,12)	(12,1)	∞	(8,14)	(10,15)	(1,5)	(5,9)
(10,15)	(10,15)	(15,4)	(8,14)	(6,6)	(3,0)	(2,7)	(6,11)	(15,13)	(5,9)	(2,10)	(1,5)	(12,1)	∞	(12,16)	(10,2)	(8,3)	(5,8)	(1,12)
(12,1)	(12,1)	(12,16)	(5,9)	(2,10)	(5,8)	(6,11)	(1,5)	(2,7)	(3,0)	(15,4)	(10,15)	(15,13)	(8,14)	(10,2)	(1,12)	∞	(8,3)	(6,6)
(12,16)	(12,16)	(5,8)	(12,1)	(5,9)	(2,7)	(6,6)	(2,10)	(1,12)	(15,13)	(3,0)	(15,4)	(10,2)	(10,15)	(8,3)	∞	(1,5)	(6,11)	(8,14)
(15,4)	(15,4)	(3,0)	(10,15)	(8,14)	(15,13)	(1,12)	(6,6)	(10,2)	(12,1)	(5,9)	(2,10)	(12,16)	(1,5)	(5,8)	(8,3)	(6,11)	(2,7)	∞
(15,13)	(15,13)	(10,2)	(3,0)	(15,4)	(8,3)	(1,5)	(10,15)	(6,11)	(5,8)	(12,16)	(12,1)	(2,7)	(5,9)	(1,12)	(6,6)	(8,14)	∞	(2,10)

Solution

We use the same approach that we adopted in the previous exercise. The session key is $k = 5 \cdot 11 \cdot G = 55G$.

Let's apply the double-and-add algorithm to $k = (110111_2)G = (d_5 d_4 d_3 d_2 d_1 d_0)_2 G$

0) $d_5 = 1 \rightarrow res_0 = 2\boldsymbol{O} + G = G = (2, 7)$

1) $d_4 = 1 \rightarrow res_1 = 2res_0 + G = 2G + G = (12, 16) + (2, 7) = 3G = (5, 9)$

2) $d_3 = 0 \rightarrow res_2 = 2res_1 = 6G = (5, 8)$

3) $d_2 = 1 \rightarrow res_3 = 2res_2 + G = 12G + G = (5, 9) + (2, 7) = 13G = (1, 5)$

4) $d_1 = 1 \rightarrow res_4 = 2res_3 + G = 26G + G = (2, 10) + (2, 7) = 27G = \boldsymbol{O}$

5) $d_0 = 1 \rightarrow res_5 = 2res_4 + G = 54G + G = 2\boldsymbol{O} + (2, 7) = 55G = (2, 7) = G$

So, the session key is equal to the generator:

$$k = G = (2, 7)$$

Speedup trick

By using the same trick of the previous exercise, we obtain:

$$k = 55G = 54G + G = 3 \cdot 18G + G = \boldsymbol{O} + G = G = (2, 7)$$

10.9 Exercises from slides (ECDH)

10.9.1 Exercise 12.1.5

Alice and Bob are using the ECDH protocol with the elliptic curve

$$E: \ y^2 \equiv x^3 + x + 6 \bmod 11$$

Alice's secret key is A = 6. She receives Bob's public key pB = (5, 9).

1) Check that pB is a point of E;
2) Compute the session key k;

Solution

To check if pB belongs to E, let's perform the replacement $(x, y) \mapsto (5, 9)$ in E equation:

$$81 \stackrel{?}{\equiv} 125 + 5 + 6 = 136 \bmod 11 \longrightarrow 4 \equiv 4 \bmod 11 \ \checkmark$$

So, since the equation is still verified, pB is a point of E.

In order to compute the session key k, Alice must compute:

$$k = \mathrm{A} \cdot \mathrm{pB} = 6\mathrm{pB} = 2 \cdot 2\mathrm{pB} + 2\mathrm{pB}$$

Let's perform the calculations:

- $2\mathrm{pB} = 2 \cdot (5, 9) = (10, 9)$

$$\begin{aligned} s &= \frac{3x^2 + a}{2y} = \frac{3 \cdot 5^2 + 1}{2 \cdot 9} = \frac{38}{9} = 9^{-1} \cdot 38 \equiv 5 \cdot 5 = 25 \equiv 3 \bmod 11 \\ x_R &= s^2 - 2x = 3^2 - 2 \cdot 5 = -1 \equiv 10 \bmod 11 \\ y_R &= s(x - x_R) - y = 3(5 - 10) - 9 = -24 \equiv 9 \bmod 11 \end{aligned}$$

- $4\mathrm{pB} = 2 \cdot 2\mathrm{pB} = 2 \cdot (10, 9) = (3, 6)$

$$\begin{aligned} s &= \frac{3x^2 + a}{2y} = \frac{3 \cdot 10^2 + 1}{2 \cdot 9} = \frac{301}{18} = 18^{-1} \cdot 301 = 2^{-1} \cdot 9^{-1} \cdot 301 \equiv 6 \cdot 5 \cdot 4 \equiv 10 \bmod 11 \\ x_R &= s^2 - 2x = 10^2 - 2 \cdot 10 = 80 \equiv 3 \bmod 11 \\ y_R &= s(x - x_R) - y = 10(10 - 3) - 9 = 61 \equiv 6 \bmod 11 \end{aligned}$$

- $6\mathrm{pB} = 4\mathrm{pB} + 2\mathrm{pB} = (3, 6) + (10, 9) = (2, 7)$

$$\begin{aligned} s &= \frac{y_2 - y_1}{x_2 - x_1} = \frac{6 - 9}{3 - 10} = \frac{3}{7} = 7^{-1} \cdot 3 \equiv 8 \cdot 3 = 24 \equiv 2 \bmod 11 \\ x_R &= s^2 - x_2 - x_1 = 2^2 - 3 - 10 = -9 \equiv 2 \bmod 11 \\ y_R &= s(x_1 - x_R) - y_1 = 2(10 - 2) - 9 \equiv 7 \bmod 11 \end{aligned}$$

So, in the end, the session key is:

$$k = 6\mathrm{pB} = (2, 7)$$

10.10 Exercises from slides (ECDSA)

10.10.1 Exercise 12.3.4

Consider ECDSA on the elliptic curve $E(\mathbb{Z}_{17}) : y^2 = x^3 + 2x + 2$ with $G = (5, 1)$, $n = 19$ and $\text{sk} = d = 5$.

Let M be a message with $hash(M) = 8$.
Sign M and compute the public key $\text{pk} = (p, a, b, n, G, B)$.

Solution

Since it's not so straightforward, let's compute the public key first.

Let's recall the general equation of an elliptic curve:

$$y^2 \equiv x^3 + a \cdot x + b \bmod p$$

- $p = 17$: it's the modulus of E and it's provided by the text, since E is defined on $\mathbb{Z}_{17}$;
- $a = 2$: it's the x coefficient in the elliptic curve equation;
- $b = 2$: it's the constant term in the elliptic curve equation;
- $n = 19$: it's the (prime) order of the cyclic group generated by G and it's provided by the text;
- $G = (5, 1)$: it's the generator of the cyclic group and it's provided by the text;
- B: it's computed as

  $$B = dG = 5G = 2 \cdot 2G + G$$

  Since the text isn't giving us the addition table for $E(\mathbb{Z}_{17})$ we have to compute everything by ourselves. Let's perform then two point doubling operations and one point addition:

  - $2G = (6, 3)$

    $$\begin{aligned} s &= \frac{3x_G^2 + a}{2y_G} = \frac{3 \cdot 5^2 + 2}{2 \cdot 1} = \frac{77}{2} = 2^{-1} \cdot 77 \equiv 9 \cdot 9 \equiv 13 \bmod 17 \\ x_{2G} &= s^2 - 2x_G = 13^2 - 2 \cdot 5 = 159 \equiv 6 \bmod 17 \\ y_{2G} &= s(x_G - x_{2G}) - y_G = 13(5 - 6) - 1 = -14 \equiv 3 \bmod 17 \end{aligned}$$

  - $4G = 2 \cdot 2G = (3, 1)$

    $$\begin{aligned} s &= \frac{3x_{2G}^2 + a}{2y_{2G}} = \frac{3 \cdot 6^2 + 2}{2 \cdot 3} = \frac{110}{2 \cdot 3} = \frac{55}{3} = 3^{-1} \cdot 55 \equiv 6 \cdot 4 \equiv 7 \bmod 17 \\ x_{4G} &= s^2 - 2x_{2G} = 7^2 - 2 \cdot 6 = 37 \equiv 3 \bmod 17 \\ y_{4G} &= s(x_{2G} - x_{4G}) - y_{2G} = 7(6 - 3) - 3 = 18 \equiv 1 \bmod 17 \end{aligned}$$

  - $B = 5G = 4G + G = (3, 1) + (5, 1) = (9, 16)$

    $$\begin{aligned} s &= \frac{y_2 - y_1}{x_2 - x_1} \equiv 0 \bmod 17 \\ x_B &= s^2 - x_1 - x_2 = -x_1 - x_2 = -8 \equiv 9 \bmod 17 \\ y_B &= s(x_1 - x_B) - y_1 = -y_1 = -1 \equiv 16 \bmod 17 \end{aligned}$$

So, in the end, the public key is:

$$\text{pk} = (17, 2, 2, 19, (5, 1), (9, 16))$$

In order to sign M let's follow the steps for the signature generation:

1. Choose an integer as random ephemeral key k_E with $0 < k_E < n$:
   - since $n = 19$ we can "pick" k_E from the set $\{1, 2, ..., 18\}$;
   - for simplicitly, because of step 2., it is convenient for us to choose either 1, 2 or 4, since we already computed the corresponding points during the public key generation. Let's pick $k_E = 4$;
2. Compute $R = k_E G = 4G = (3, 1)$;
3. Let $r = x_R = 3$;
4. Compute $s \equiv (hash(M) + d \cdot r)k_E^{-1} \bmod n$:

$$s = (8 + 5 \cdot 3)4^{-1} = 23 \cdot 4^{-1} \equiv 4 \cdot 4^{-1} \equiv 1 \bmod 19$$

So, the signature is:

$$sig_{\text{sk}}(M) = (r, s) = (3, 1)$$

For completeness, let's perform the verification of the signature:

- Computation of the hash of the message (already provided by the text): $hash(x) = 8$;
- Auxiliary values computation:

$$\begin{aligned} w &\equiv s^{-1} \bmod n &&\longrightarrow w &&= 1^{-1} \equiv 1 \bmod 19 \\ u_1 &\equiv w \cdot hash(x) \bmod n &&\longrightarrow u_1 &&\equiv 8 \bmod 19 \\ u_2 &\equiv w \cdot r \bmod n &&\longrightarrow u_2 &&\equiv 3 \bmod 19 \end{aligned}$$

- Final computation + verification:

$$\begin{aligned} P &= u_1 G + u_2 B = 8G + 3B = 8G + 15G = \\ &= 23G = 19G + 4G = \boldsymbol{O} + 4G = 4G = (3, 1) \end{aligned}$$

  Notice that we've re-used the results obtained during the public key computation to speed things up.

$$x_P = 3 \equiv r \bmod 19 \checkmark$$

11 Appendix: the infamous exercise with $\oplus$ and $\boxtimes$

11.1 Introduction

This exercise has been proposed at the 2023-07-18 exam. I think I've found a general approach to solve this kind of exercises but keep in mind that there may be better solutions...

Edit: the exercise can be solved in a faster way with a Meet In The Middle attack.

11.2 Exercise text

Let $Enc_k^1(P) = k \oplus P$ be the Vernam or XOR cipher of 3-bit blocks.

Let $Enc_k^2(P) = k \boxtimes P$ be the multiplication cipher modulo $8 = 2^3$ where k, P are binary expressions of elements of $\mathbb{Z}_8$, i.e., [011] is 3.

Let

$$Enc_k(P) = Enc_{k2}^2\left(Enc_{k1}^1(P)\right)$$

be the 3-bit double encryption.

Knowing that $Enc_k(3) = 3$ and $Enc_k(4) = 6$, find the pair $(k2, k1)$

11.3 The reasons behind my approach

Another version of this exercise has been proposed in the past and the corresponding solution in the Holy Crypto Bible seemed to be correct. However, by using that same approach in this exercise, we can't get to solve it. The fact that the Bible approach works on the other version of this exercise is just a mere coincidence. Here's what we have to keep in mind:

- Valid properties of modular arithmetic for integers in $\mathbb{Z}_n$:

Property	**Expression**
Commutative Laws	$(w + x) \bmod n = (x + w) \bmod n$ $(w \times x) \bmod n = (x \times w) \bmod n$
Associative Laws	$[(w + x) + y] \bmod n = [w + (x + y)] \bmod n$ $[(w \times x) \times y] \bmod n = [w \times (x \times y)] \bmod n$
Distributive Law	$[w \times (x + y)] \bmod n = [(w \times x) + (w \times y)] \bmod n$
Identities	$(0 + w) \bmod n = w \bmod n$ $(1 \times w) \bmod n = w \bmod n$
Additive Inverse $(-w)$	For each $w \in \mathbb{Z}_n$, there exists a z such that $w + z \equiv 0 \bmod n$

  where $\times$ is the multiplication modulo n, so it's equivalent to $\boxtimes$, and $+$ is the addition modulo n which is different from the XOR operation $\oplus$.

- Invalid properties:
  - Distributive Law with $\boxtimes$ and $\oplus$: $x \boxtimes (a \oplus b) \neq (x \boxtimes a) \oplus (x \boxtimes b)$

    e.g. if $\boxtimes$ is performed $\pmod 8$:

$$7 \boxtimes (6 \oplus 3) = 7 \boxtimes 5 = 3 \neq (7 \boxtimes 6) \oplus (7 \boxtimes 3) = 2 \oplus 5 = 7$$

11.4 What we have to know

In order to adopt this approach we have to know how multiplication modulo n is performed at the bit level. First of all, let's start with regular multiplication without the modulo operation...

Let's compute $10 \times 19 = 190$ at the bit level. Here's how to proceed:

- multiply each bit of the bit string x for all the bits of the bit string y;
- every time you go to the next bit of x move the result one column to the left;
- sum all the bits in the same column (with an eventual carry);

				0	1	0	1	0
			×	1	0	0	1	1
				0	1	0	1	0
			0	1	0	1	0	
		0	0	0	0	0		
	0	0	0	0	0			
0	1	0	1	0				
0	1	0	1	1	1	1	1	0

As you can see, we've obtained the binary representation of 190. To make things prettier you can omit the all-0 rows but remember to keep shifting one column to the left when moving from bit to bit:

				0	1	0	1	0
			×	1	0	0	1	1
				0	1	0	1	0
			0	1	0	1	0	
0	1	0	1	0				
0	1	0	1	1	1	1	1	0

Now let's consider the multiplication modulo n at the bit level:

- the only difference with the regular multiplication is that the result is truncated up to the k-th bit where k is the position of the MSB of the modulus n.

Let's compute $10 \boxtimes 19 = 30 \bmod 32$:

	0	1	0	1	0
$\boxtimes$	1	0	0	1	1
	0	1	0	1	0
	1	0	1	0	
	0				
	1	1	1	1	0

in this case the modulus $32 = 2^5$ has its most significant bit in the 5^{th} position. So, all bits beyond the 5^{th} position (included) must be discarded, since the result of an operation modulo n can't be greater or equal to n.

Now that we know this, we can see how to solve the exercise...

11.5 Solving the exercise

First of all, let's write the system of equations which describes our problem:

$$\begin{cases} k_2 \boxtimes (k_1 \oplus 3) = 3 \\ k_2 \boxtimes (k_1 \oplus 4) = 6 \end{cases}$$

Let's represent k_1 and k_2 in the binary form:

$$k_1 = (b_2 b_1 b_0)_2$$
$$k_2 = (d_2 d_1 d_0)_2$$

where $b_i, d_i \in \{0, 1\}$ are the bits of (respectively) k_1 and k_2.

Let's compute the following operations regarding the 1st equation of the system:

- $k_1 \oplus 3$:

b_2	b_1	b_0
0	1	1
b_2	$(b_1 \oplus 1)$	$(b_0 \oplus 1)$

  remember that XOR-ing with 0 is equivalent to not XOR-ing at all;

- $k_2 \boxtimes (k_1 \oplus 3)$:

b_2 d_2	$(b_1 \oplus 1)$ d_1	$(b_0 \oplus 1)$ d_0
$d_0 b_2$ $d_1(b_1 \oplus 1)$ $d_2(b_0 \oplus 1)$	$d_0(b_1 \oplus 1)$ $d_1(b_0 \oplus 1)$	$d_0(b_0 \oplus 1)$
$d_0 b_2 + d_1(b_1 \oplus 1) + d_2(b_0 \oplus 1) + c_1$	$d_0(b_1 \oplus 1) + d_1(b_0 \oplus 1)$	$d_0(b_0 \oplus 1)$

  where c_1 (last row 1st column) is a possible carry coming from the sum in the 2nd column.

  From this result we can build a new system of equations based on the bits b_i, d_i.

- $k_2 \boxtimes (k_1 \oplus 3) = 3$

$$\begin{cases} d_0 b_2 + d_1(b_1 \oplus 1) + d_2(b_0 \oplus 1) + c_1 = 0 \\ d_0(b_1 \oplus 1) + d_1(b_0 \oplus 1) = 1 \\ d_0(b_0 \oplus 1) = 1 \\ c_1 = \text{carry}(d_0(b_1 \oplus 1) + d_1(b_0 \oplus 1)) \end{cases}$$

  notice that we have a system of 4 equations with 7 unknowns. We can't solve it, for now...

Now, let's do the same thing with the 2nd equation of the system:

- $k_1 \oplus 4$:

b_2	b_1	b_0
1	0	0
$(b_2 \oplus 1)$	b_1	b_0

- $k_2 \boxtimes (k_1 \oplus 4)$:

$(b_2 \oplus 1)$ d_2	b_1 d_1	b_0 d_0
$d_0(b_2 \oplus 1)$ $d_1 b_1$ $d_2 b_0$	$d_0 b_1$ $d_1 b_0$	$d_0 b_0$
$d_0(b_2 \oplus 1) + d_1 b_1 + d_2 b_0 + c_2$	$d_0 b_1 + d_1 b_0$	$d_0 b_0$

  where c_2 (last row 1st column) is a possible carry coming from the sum in the 2nd column.

- $k_2 \boxtimes (k_1 \oplus 4) = 6$

$$\begin{cases} d_0(b_2 \oplus 1) + d_1 b_1 + d_2 b_0 + c_2 = 1 \\ d_0 b_1 + d_1 b_0 = 1 \\ d_0 b_0 = 0 \\ c_2 = \text{carry}(d_0 b_1 + d_1 b_0) \end{cases}$$

Now we can merge the systems of 4 equations obtained in the previous steps into a system of 8 equations:

$$\begin{cases} d_0(b_0 \oplus 1) = 1 \\ d_0 b_0 = 0 \\ \\ d_0 b_1 + d_1 b_0 = 1 \\ d_0(b_1 \oplus 1) + d_1(b_0 \oplus 1) = 1 \\ \\ d_0(b_2 \oplus 1) + d_1 b_1 + d_2 b_0 + c_2 = 1 \\ d_0 b_2 + d_1(b_1 \oplus 1) + d_2(b_0 \oplus 1) + c_1 = 0 \\ \\ c_1 = \text{carry}(d_0(b_1 \oplus 1) + d_1(b_0 \oplus 1)) \\ c_2 = \text{carry}(d_0 b_1 + d_1 b_0) \end{cases}$$

From the 1st equation of the system we understand that $d_0 = 1$ and $b_0 = 0$ because the product of d_0 with $b_0 \oplus 1$ must be equal to 1 and so both factors of the product must be equal to 1. Let's re-write the system:

$$\begin{cases} d_0 = 1 \\ b_0 = 0 \\ \\ d_0 b_1 + d_1 b_0 = 1 \\ d_0(b_1 \oplus 1) + d_1(b_0 \oplus 1) = 1 \\ \\ d_0(b_2 \oplus 1) + d_1 b_1 + d_2 b_0 + c_2 = 1 \\ d_0 b_2 + d_1(b_1 \oplus 1) + d_2(b_0 \oplus 1) + c_1 = 0 \\ \\ c_1 = \text{carry}((b_1 \oplus 1) + d_1) \\ c_2 = \text{carry}(b_1) = 0 \end{cases} \longrightarrow \begin{cases} d_0 = 1 \\ b_0 = 0 \\ \\ b_1 = 1 \\ (b_1 \oplus 1) + d_1 = 1 \\ \\ (b_2 \oplus 1) + d_1 b_1 = 1 \\ b_2 + d_1(b_1 \oplus 1) + d_2 + c_1 = 0 \\ \\ c_1 = \text{carry}(d_1) = 0 \\ c_2 = \text{carry}(b_1) = 0 \end{cases}$$

Where $c_2 = \text{carry}(b_1) = 0$ because, independently from the value of b_1, we can't have any carry since we're not performing any addition (because the other addend was equal to 0). Same reasoning goes for c_1.

As we can see, after performing the replacement $d_0 = 1,\ b_0 = 0$ in all the equations, we've automatically discovered that $b_1 = 1$. Let's go on...

$$\begin{cases} d_0 = 1 \\ b_0 = 0 \\ \\ b_1 = 1 \\ d_1 = 1 \\ \\ (b_2 \oplus 1) + 1 = 1 \\ b_2 + d_2 = 0 \end{cases} \longrightarrow \begin{cases} d_0 = 1 \\ b_0 = 0 \\ \\ b_1 = 1 \\ d_1 = 1 \\ \\ b_2 \oplus 1 = 0 \\ b_2 + d_2 = 0 \end{cases} \longrightarrow \begin{cases} d_0 = 1 \\ b_0 = 0 \\ \\ b_1 = 1 \\ d_1 = 1 \\ \\ b_2 = 1 \\ d_2 = 1 \end{cases}$$

We have solved the system and we've obtained the following results:

$$\begin{aligned} k_1 &= (b_2 b_1 b_0)_2 &&= (110)_2 = 6 \\ k_2 &= (d_2 d_1 d_0)_2 &&= (111)_2 = 7 \end{aligned}$$

Let's verify that they actually solve the system:

$$\begin{cases} 7 \boxtimes (6 \oplus 3) = 7 \boxtimes 5 = 35 \equiv 3 \bmod 8 \ \checkmark \\ 7 \boxtimes (6 \oplus 4) = 7 \boxtimes 2 = 14 \equiv 6 \bmod 8 \ \checkmark \end{cases}$$

11.6 Conclusions

This method may seem complex and time consuming but keep in mind that it's just a bunch of bit-wise operations. In my opinion, a brute-force approach would take much longer!

11.7 A faster approach: Meet In The Middle attack

Since we're dealing with a double encryption of a plaintext, we can use this attack to guess $k1$ and $k2$.

So, we have to compute all possible values of $Enc^1_{k1}(P)$ and $Dec^2_{k2}(C)$, with $P = 3, C = 3$ and $P = 4, C = 6$, until we find a key pair $(k2, k1)$ such that $Enc^1_{k1}(P) = Dec^2_{k2}(C)$. Because of the Vernam cipher, all computations are done modulo 8. So, it's straightforward that $k1, k2 \in \{0, 1, ..., 7\}$. Let's compute all possible results of $Enc^1_{k1}(P)$ and $Dec^2_{k2}(C)$:

$k1$	$Enc^1_{k1}(3)$	$Enc^1_{k1}(4)$
0	3	4
1	2	5
2	1	6
3	0	7
4	7	0
5	6	1
6	5	2
7	4	3

$k2$	$k2^{-1}$	$Dec^2_{k2}(3)$	$Dec^2_{k2}(6)$
0	-	-	-
1	1	3	6
2	-	-	-
3	3	1	2
4	-	-	-
5	5	5	6
6	-	-	-
7	7	5	2

Notice that $Dec^2_{k2}(C) = k2^{-1} \boxtimes C$ where $k2^{-1}$ is the inverse of $k2$ modulo 8. So, some values of $k2$ can be immediately discarded since their inverse modulo 8 does not exist, because $\gcd(k2, 8) \neq 1$.

Remember that we must compare the values obtained from the pairs (P, C) that actually make sense. So, as the text of the problem states, $(3, 3)$ and $(4, 6)$. So, it must be true that $Enc^1_{k1}(3) = Dec^2_{k2}(3)$ and $Enc^1_{k1}(4) = Dec^2_{k2}(6)$.