# Cryptography and Elliptic curves

Inna Lukyanenko, St.Petersburg State University

JASS'07

## 1 Introduction to Cryptography

The term *Cryptography* is derived from Greek words "hidden" and "write" and it's original meaning is the study of message secrecy. In modern times, it has become a branch of information theory, as the mathematical study of information and especially its transmission from place to place.

One of cryptography's primary purposes is hiding the meaning of messages. That's why, until modern times, cryptography was concerned solely with message *confidentiality*, i.e. encryption. *Encryption* is the process of converting ordinary information (plaintext) into a ciphertext. *Decryption* is the reverse process. A *cipher* is a pair of algorithms, which perform this encryption and the reversing decryption. The both operations are controlled by a *key* - a secret parameter for the cipher algorithm. Keys are very important because ciphers without variable keys are trivially breakable.

The modern cryptography can be divided into two main areas of study:

1. *Symmetric-key cryptography.* It refers to encryption methods, which use the same (secret) key for both encryption and decryption. Or, less commonly, the keys are different, but related in an easily computable way. This was the only kind of encryption publicly known until 1976. A significant disadvantage of this method is that it requires the prior agreement about the key between the sender and receiver, using some secure channel. For example, personal meeting. But in practice this may be very difficult to achieve.
2. *Public-key cryptography.* The idea of public-key cryptography, which was proposed in 1976 by Whitfield Diffie and Martin Hellman, is to use a pair of cryptographic keys - a public key (for encryption) and a private key (for decryption). The private key is kept secret, while the public key may be widely distributed. The keys are related mathematically, but the private key cannot be practically derived from the public.

### 1.1 Public-key cryptography

There are two main branches of publc-key cryptography depending on its purpose:

- *Public-key encryption*: a message is encrypted with users public key, but cannot be decrypted without the corresponding private key. This is used to ensure confidentiality.
- *Digital signatures*: a message is signed with users private key, but can be verified by anyone who has access to the users public key. This is used to ensure authenticity.

This can be illustrated on the following two examples. An analogy for public-key encryption is the locked mailbox with a mail slot. Everyone can drop the message through the slot, but only the owner of the key can open the box and read the message. An analogy for digital signatures is the sealing of an envelope with the personal wax seal. The message can be opened and red by anyone, but the presence of the seal authenticates the sender.

The security of public-key algorithms is based on the computational complexity of "hard" problems, often from number theory. For example, the integer factorization problem and the discrete logarithm problem.

## 1.2 Digital Signatures

A *digital signature scheme* is a type of public-key cryptosystem used to simulate the security properties of a signature in digital form. Digital signatures normally give two algorithms: one for signing, which involves the user's secret key, and one for verifying, which involves the user's public key. The output of signature process is called the "digital signature". Digital signatures are used to provide the authentication of the associated input, usually called a "message".

### 1.2.1 Overview of Digital Signatures

- *RSA* was invented in 1978 by Ronald Rivest, Adi Shamir and Leonard Adleman and its security is based on the integer factorization problem.
- *DSA (Digital Signature Algorithm)* was developed in 1991 and is related to the discrete logarithm problem.
- *ECDSA (Elliptic Curve Digital Signature Algorithm)* is a modification of DSA involving elliptic curve groups, which was proposed in 1992 by Scott Vanstone. It provides smaller key sizes for the same security level and that's why it has become the most popular digital signature.

### 1.2.2 The general description

A digital signature scheme typically consists of three algorithms:

1. A *key generation* algorithm $G$ that randomly produces a key pair $(PK, SK)$ for the signer. $PK$ is the verifying key, which is to be public, and $SK$ is the signing key, to be kept private.
2. A *signing* algorithm $S$ that, on input of a message $m$ and a signing key $SK$, produces a signature $\sigma$.
3. A *verifying* algorithm $V$ that, on input of a message $m$, a verifying key $PK$ and a signature $\sigma$, either accepts or rejects.

Let us notice, that the public-key systems are computationally significant more expensive than the symmetric ones. That's why a message is previously hashed (using a *cryptographic hash function*) and the smaller "hash value" is signed. Before verifying a signature, the receiver computes the hash of the message himself, and compares it with the decrypted one to check that the message has not been tampered with.

### 1.2.3 One-way functions

Almost all digital signatures are based on the existence of the so called one-way functions. A *one-way function* is a function that is easy to compute, but hard to invert. The precise meanings of "easy" and "hard" can be expressed mathematically: "easy" means that there exists an algorithm that can compute this function in probabilistic polynomial time. "Hard" means that no such algorithm exists.

We are interested in a special kind of one-way function - a *trapdoor one-way function*. It is hard to invert unless some secret information, called *trapdoor*, is known. We should notice, that the existence of one-way functions is an open question. But there are some candidates, for which no polynomial-time inverting algorithm is known:

- a product of two large primes: it is believed to be difficult to factorize a product of two large primes, but it is easy as soon as you know one of them. This is called the *integer factorization problem* and used in RSA.
- an exponentiation in the finite field: it is believed to be difficult to extract discrete logarithms in a finite field. This is called the *discrete logarithm problem* and the security of DSA and ECDSA is based on it.

### 1.2.4 Discrete Logarithm Problem

*Discrete logarithm* can be viewed as a group-theoretic analogy of the ordinary logarithm. Let us consider a finite multiplicative group $(G, \cdot)$. A group, which is used in DSA, is the multiplicative group of a finite field $\mathbb{Z}_p$, where $p$ is prime. For an element $g \in G$ of order $n$ we define

$$\langle g \rangle = \{g^i : 0 \leq i \leq n-1\}.$$

Obviously, $\langle g \rangle$ is a cyclic subgroup of order $n$ in $(G, \cdot)$. Given $y \in \langle g \rangle$ there exists a unique integer $x$, $0 \leq x \leq n-1$, such that $y = g^x$. It is called the *discrete logarithm* of $y$ to the base $g$ and denoted by $x = log_g y$. It is the inverse operation to discrete exponentiation.

No efficient algorithm for computing discrete logarithms is known. The naive algorithm is to raise $g$ to the higher and higher powers until the desired $y$ is found. This algorithm is exponential in the number of digits in the group size. There exist more sophisticated algorithms, which run faster than the naive, but none of them runs in polynomial time. That's why the discrete exponentiation is believed to be a one-way function.

## 1.3 The Digital Signature Algorithm (DSA)

The DSA was proposed in August 1991 by the U.S. National Institute of Standards and Technology (NIST). Its security is based on the intractability of the discrete logarithm problem in prime-order subgroups of $\mathbb{Z}_p^*$.

*Domain parameters generation.* Domain parameters are generated for each entity in a particular security domain according to the following algorithm:

1. Select a 160-bit prime $q$ and 1024-bit prime $p$ with the property that $q|p-1$.
2. Select an element $h \in \mathbb{Z}_p^*$ and compute $g = h^{(p-1)/q} \bmod p$. (Repeat until $g \neq 1$.)
   From *Fermat's little theorem* follows that $g^q \equiv h^{(p-1)} \equiv 1 \bmod p$. That's why $g$ is a generator of a cyclic subgroup of order $q$ in $\mathbb{Z}_p^*$.
3. $(p, q, g)$ are domain parameters.

*Key generation.* Each entity in the domain with domain parameters $(p, q, g)$ does the following:

1. Select a random integer $x$ such that $1 \leq x \leq q-1$.
2. Compute $y = g^x \bmod p$.
3. $y$ is a public key; $x$ is a private key.

*DSA Signature generation.* To generate the signature from the original message $m$ one does the following:

1. Select a random integer $k$, $1 \leq k \leq q-1$.
2. Compute $e = HASH(m)$, where $HASH$ is a cryptographic hash function, such as $SHA-1$.
3. Compute $r = (g^k \bmod p) \bmod q$.
4. Compute $s = (k^{-1}(e + xr)) \bmod q$.
5. Go to step 1 if $r = 0$ or $s = 0$.
6. $(r, s)$ is a signature for the message $m$.

*DSA Signature verification.* To verify the signature $(r, s)$ on the message $m$ using the public key $y$ and domain parameters $(p, q, g)$ the receiver does the following:

1. Verify that $1 \leq r, s \leq q - 1$.
2. Compute $e = HASH(m)$.
3. Compute $w = s^{-1} \bmod q$.
4. Compute $u_1 = (ew) \bmod q$.
5. Compute $u_2 = (rw) \bmod q$.
6. Compute $v = (g^{u_1} y^{u_2} \bmod p) \bmod q$.
7. Accept the signature if and only if $v = r$.

*DSA Correctness.* We should explain that the signature scheme is correct in the sense that it always accepts the true signatures. Let $(r, s)$ be a signature constructed with the private key $x$. The signer computes $s = k^{-1}(e + xr) \bmod q$. Rearranging gives

$$k \equiv (e + xr)s^{-1} \equiv (e + xr)w \bmod q.$$

Since $g$ has order $q$ we have

$$g^k \equiv g^{ew} g^{xrw} \equiv g^{ew} y^{rw} \equiv g^{u_1} y^{u_2} \bmod p.$$

And finally

$$r = (g^k \bmod p) \bmod q = (g^{u_1} y^{u_2} \bmod p) \bmod q = v.$$

# 2 Finite fields

A *finite field* is a finite set of elements $F$ together with two binary operations " $+$ " and " $\cdot$ ", that satisfy certain arithmetic properties. The *order* of $F$ is the number of its elements. We know, that there exists a finite field of order $q$ if and only if $q = p^m$, where $p$ is prime, and this field is essentially unique. It is denoted by $\mathbb{F}_q$. In this case $p$ is called a *characteristic* of $\mathbb{F}_q$, $m$ is called the *extension degree* of $\mathbb{F}_q$. For elliptic curve cryptography we need one of two cases: $q = p$ , where $p$ is an odd prime, or $q = 2^m$. Let us notice, that there are many ways of representing the elements of $\mathbb{F}_q$ providing different efficiency. In the next sections we describe some of them.

## 2.1 The finite field $\mathbb{F}_p$

Let $p$ be a prime number. The finite field $\mathbb{F}_p$, called a *prime field*, is comprised of the set of integers $\{0, 1, 2, ..., p - 1\}$ with the following arithmetic operations:

- *Addition:* If $a, b \in \mathbb{F}_p$, then $a + b = r$, where $r = (a + b) \bmod p$, $0 \leq r \leq p - 1$. This is known as *addition modulo p*.
- *Multiplication:* If $a, b \in \mathbb{F}_p$, then $a \cdot b = s$, where $s = a \cdot b \bmod p$, $0 \leq s \leq p - 1$. This is known as *multiplication modulo p*.
- *Inversion:* If $a \in \mathbb{F}_p$, $a \neq 0$, then there exists the unique integer $a^{-1} \in \mathbb{F}_p$, such that $a \cdot a^{-1} = 1$.

## 2.2 The finite field $\mathbb{F}_{2^m}$

The finite field $\mathbb{F}_{2^m}$, called a *binary finite field*, can be viewed as a vector space of dimension $m$ over the field $\mathbb{F}_2$, which consists of the two elements 0 and 1. It means, that there exists a basis of $m$ elements $\{\alpha_0, \alpha_1, ..., \alpha_{m-1}\} \in \mathbb{F}_{2^m}$, such that each element $\alpha \in \mathbb{F}_{2^m}$ can be uniquely written in the form:

$$\alpha = a_0\alpha_0 + a_1\alpha_1 + ... + a_{m-1}\alpha_{m-1}, \quad \text{where} \quad a_i \in \{0, 1\}.$$

If the basis is fixed, each element $\alpha$ can be represented as the bit string $(a_0a_1...a_{m-1})$. Addition is performed by bitwise XOR-ing the vector representations. The multiplication depends on the basis selected. There are many different kinds of bases of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ and some of them lead to more efficient implementations. Let us consider two of them: polynomial bases and normal bases.

### 2.2.1 Polynomial basis representation

Let $f(x) = x^m + f_{m-1}x^{m-1} + ... + f_2x^2 + f_1x + f_0$ (where $f_i \in \{0, 1\}$ for $i = 0, 1, ..., m-1$) be an irreducible polynomial of degree $m$ over $\mathbb{F}_2$. That is, $f(x)$ cannot be factored as a product of two polynomials over $\mathbb{F}_2$ of degree less than $m$. Such polynomial is called the *reduction polynomial* and it defines a polynomial basis representation of $\mathbb{F}_{2^m}$.

*Field elements.* The field is comprised of all polynomials over $\mathbb{F}_2$ of degree less than $m$:

$$\mathbb{F}_{2^m} = \{a(x) = a_{m-1}x^{m-1} + ... + a_1x + a_0 : a_i \in \{0, 1\}\}.$$

The field element $a_{m-1}x^{m-1} + ... + a_1x + a_0$ is usually denoted by the bit string $(a_{m-1}...a_1a_0)$ of length $m$, so that

$$\mathbb{F}_{2^m} = \{(a_{m-1}...a_1a_0) : a_i \in \{0, 1\}\}.$$

Thus, the elements of $\mathbb{F}_{2^m}$ can be represented by the set of all binary strings of length $m$. The multiplicative identity element (1) is represented by the bit string (00...01), the additive identity element (0) is represented by (00...00).

*Field operations:*

- *Addition:* If $a = (a_{m-1}...a_1a_0),\ b = (b_{m-1}...b_1b_0) \in \mathbb{F}_{2^m}$, then $a + b = c = (c_{m-1}...c_1c_0)$, where $c_i = (a_i + b_i) \bmod 2$.
- *Multiplication:* If $a = (a_{m-1}...a_1a_0),\ b = (b_{m-1}...b_1b_0) \in \mathbb{F}_{2^m}$, then $a \cdot b = r = (r_{m-1}...r_1r_0)$, where $r(x) = a(x) \cdot b(x) \bmod f(x)$ over $\mathbb{F}_2$. That is, $r(x)$ is the remainder when the polynomial $a(x) \cdot b(x)$ is devided by $f(x)$ over $\mathbb{F}_2$.
- *Inversion:* If $a = (a_{m-1}...a_1a_0) \in \mathbb{F}_{2^m}$, $a \neq 0$, then there exists the unique $a^{-1} \in \mathbb{F}_{2^m}$, such that $a \cdot a^{-1} = 1$.

### 2.2.2 Normal basis representation

A *normal basis* of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ is a basis of the form $\{\beta, \beta^2, \beta^{2^2}, ..., \beta^{2^{m-1}}\}$, where $\beta \in \mathbb{F}_{2^m}$. Such a basis always exists. Each element can be written as $a = \sum_{i=0}^{m-1} a_i\beta^{2^i}$, where $a_i \in \{0, 1\}$. Normal basis representations have the computational advantage that squaring of an element can be done very efficiently. But, on the other hand, the multiplication can be cumbersome in general. For this reason, *Gaussian* normal bases are used, for which multiplication is both simpler and more efficient. But the construction itself is relatively complicated, so we wouldn't describe it here.

*Field elements:*

$$\mathbb{F}_{2^m} = \{a = \sum_{i=0}^{m-1} a_i\beta^{2^i} : a_i \in \{0, 1\}\}.$$

Each element can be represented by a bit string, so that

$$\mathbb{F}_{2^m} = \{(a_0 a_1 ... a_{m-1}) : a_i \in \{0, 1\}\}.$$

The multiplicative identity element (1) is represented by the bit string (11...11), the additive identity element (0) is represented by (00...00).

*Field operations:*

- *Addition:* If $a = (a_0 a_1 ... a_{m-1})$, $b = (b_0 b_1 ... b_{m-1}) \in \mathbb{F}_{2^m}$, then $a + b = c = (c_0 c_1 ... c_{m-1})$, where $c_i = (a_i + b_i) \bmod 2$.
- *Squaring:* Let $a = (a_0 a_1 ... a_{m-1}) \in \mathbb{F}_{2^m}$.

$$a^2 = \sum_{i=0}^{m-1} a_i \beta^{2^{i+1}} = \sum_{i=0}^{m-1} a_{i-1} \beta^{2^i} = (a_{m-1} a_0 a_1 ... a_{m-2}).$$

- *Multiplication:* with use of *Gaussian normal basis (GNB).*
- *Inversion:* If $a \in \mathbb{F}_{2^m}$, $a \neq 0$, then there exists the unique integer $a^{-1} \in \mathbb{F}_{2^m}$, such that $a \cdot a^{-1} = 1$.

# 3 Elliptic curves

## 3.1 Elliptic curves over $\mathbb{F}_p$

Let $p > 3$ be an odd prime, $a, b \in \mathbb{F}_p$, such that $4a^3 + 27b^2 \neq 0 \bmod p$. An *elliptic curve* $E$ over $\mathbb{F}_p$ is the folloving set of elements:

$$E(\mathbb{F}_p) = \{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p : \ y^2 = x^3 + ax + b\} \cup \{\mathcal{O} - \text{point at infinity}\}.$$

The requirement $4a^3 + 27b^2 \neq 0$ means that the curve is *non-singular*, i.e. has no cusps and self-intersections.

*Addition rule.* There is a rule, called the *chord-and-tangent rule*, for adding two points on an elliptic curve $E(\mathbb{F}_p)$ to give a third point. Together with this addition operation a set $E(\mathbb{F}_p)$ forms a group with $\mathcal{O}$ serving as identity. Exactly this group is used in ECDSA. The addition rule can be best explained geometrically:

- *Point addition.* Let $P$ and $Q$ be two distinct points on an elliptic curve $E$. Then the sum of $P$ and $Q$, denoted by $R$, is defined as follows. At first, one should draw a line through $P$ and $Q$. This line intersects the elliptic curve at a third point. Then $R$ is the reflection of this point in the $x$ axis.
- *Point doubling.* Let $P$ be a point on an elliptic curve. Then the double of $P$, denoted by $R$, is defined as follows. One should draw a tangent line to the elliptic curve at $P$. It intersects the elliptic curve at the second point. Then $R$ is the reflection of this point in the $x$ axis.

The following algebraic formulas can now be easily derived from the geometric description:

1. $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in E(\mathbb{F}_p)$.
2. If $P = (x, y) \in E(\mathbb{F}_p)$, then $-P = (x, -y) \in E(\mathbb{F}_p)$.
3. Let $P = (x_1, y_1)$, $Q = (x_2, y_2) \in E(\mathbb{F}_p)$, such that $P \neq \pm Q$. Then $P + Q = (x_3, y_3)$, where

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \quad \text{and} \quad y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1.$$

4. If $P = (x_1, y_1) \in E(\mathbb{F}_p)$, such that $P \neq -P$, then $2P = (x_3, y_3)$, where

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \quad \text{and} \quad y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1.$$

## 3.2 Elliptic curves over $\mathbb{F}_{2^m}$

Let $a, b \in \mathbb{F}_{2^m}$, such that $b \neq 0$. Then an *elliptic curve* $E$ over $\mathbb{F}_{2^m}$ is the following set of elements:

$$E(\mathbb{F}_{2^m}) = \{(x, y) \in \mathbb{F}_{2^m} \times \mathbb{F}_{2^m} : \ y^2 + xy = x^3 + ax^2 + b\} \cup \{\mathcal{O} - \text{point at infinity}\}.$$

The *geometric description* of an addition operation is similar to the case of $E(\mathbb{F}_p)$. Together with this addition operation a set $E(\mathbb{F}_{2^m})$ forms a group with $\mathcal{O}$ serving as identity.

The algebraic formulas for the sum of two points and the double of the point:

1. $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in E(\mathbb{F}_{2^m})$.
2. If $P = (x, y) \in E(\mathbb{F}_{2^m})$, then $-P = (x, x + y) \in E(\mathbb{F}_{2^m})$.
3. Let $P = (x_1, y_1),\ Q = (x_2, y_2) \in E(\mathbb{F}_{2^m})$, such that $P \neq \pm Q$. Then $P + Q = (x_3, y_3)$, where

$$x_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)^2 + \frac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a \quad \text{and} \quad y_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)(x_1 + x_3) + x_3 + y_1.$$

4. If $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$, such that $P \neq -P$, then $2P = (x_3, y_3)$, where

$$x_3 = x_1^2 + \frac{b}{x_1^2} \quad \text{and} \quad y_3 = x_1^2 + \left(x_1 + \frac{y_1}{x_1}\right)x_3 + x_3.$$

## 3.3 Basic facts

*Group order.* Let $E$ be an elliptic curve over a finite field $\mathbb{F}_q$. Hasse's theorem gives the following estimation for the number of points on an elliptic curve:

$$\#E(\mathbb{F}_q) = q + 1 - t, \ \text{where} \ \ |t| \leq 2\sqrt{q}.$$

Then $\#E(\mathbb{F}_q)$ is called the *order* of $E$ and $t$ is called the *trace* of $E$. In other words, the order of an elliptic curve $E(\mathbb{F}_q)$ is roughly equal to the size $q$ of an underlying field.

*Group structure.* $E(\mathbb{F}_q)$ is an abelian group of rank 1 or 2. That is, $E(\mathbb{F}_q) \cong \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$, where $n_2$ divides $n_1$ and $n_2$ divides $q - 1$, for unique positive integers $n_1$ and $n_2$. $\mathbb{Z}_n$ denotes the cyclic group of order $n$. In the case $n_2 = 1$ (one of these groups is trivial), $E(\mathbb{F}_q) \cong \mathbb{Z}_n$ is cyclic of order $n = n_1$ and there exists a *generator* $G \in E(\mathbb{F}_q)$, such that $E(\mathbb{F}_q) = \{kG : 0 \leq k \leq n - 1\}$.

## 3.4 ECDLP

Let us notice, that the additive cyclic subgroup of order $n$ described above can be considered similar to the multiplicative group of powers of an integer $g$ modulo prime $n$:

$$(\mathcal{O}, G, 2G, 3G, ..., (n-1)G) \ \Leftrightarrow \ (e, g, g^2, g^3, ..., g^{(n-1)}).$$

That's why the problem of finding $k$ $(0 \leq k \leq n - 1)$, if the points $G$ and $kG$ are given, is called the *elliptic curve discrete logarithm problem (ECDLP)*. The security of elliptic curve cryptography is based on the hardness of this problem.

# 4 ECDSA

The discrete logarithm problem on elliptic curve groups is believed to be more difficult than the corresponding problem in the multiplicative group of the underlying finite field. That's why the keys can be chosen much shorter for a comparable level of security. As for other popular public-key cryptosystems, no mathematical proof of difficulty has been published, but in 1999 it was accepted as an ANSI (American National Standarts Institute) standard. In practice, it is going to replace RSA and DSA.

## 4.1 Domain parameters

The domain parameters consist of a suitably chosen elliptic curve $E(\mathbb{F}_q)$ of characteristic $p$ and a base point $G \in E(\mathbb{F}_q)$. So, we should define the following parameters:

1. a field size $q = p$ or $2^m$.
2. the representation $FR$ of $\mathbb{F}_q$: in the case $q = 2^m$ elements are represented with respect to a polynomial or normal basis.
3. $a, b \in \mathbb{F}_q$ satisfying corresponding restrictions, which define the equation of EC:

$$\begin{cases} y^2 = x^3 + ax + b & \text{in the case } p > 3 \\ y^2 + xy = x^3 + ax^2 + b & \text{in the case } p = 2 \end{cases}$$

4. a generator $G = (x_G, y_G) \in E(\mathbb{F}_q)$ of prime order.
5. the order $n$ of $G$ is a large prime and $n > 4\sqrt{q}$.
6. the cofactor $h = \#E(\mathbb{F}_q)/n \quad (h \leq 4)$.

*Generation and validation.* The generation of domain parameters can be very time-consuming and troublesome to implement. That's why there are some standard domain parameters for several common field sizes.

If one wants to build his own parameters, one should at first select the underlying field and then generate the elliptic curve, with respect to the condition that its order is divisible by a large prime $n$. Several classes of curves are "weak" and should be avoided. For example, $E(\mathbb{F}_{2^m})$ with non prime $m$ and so-called "anomalous" curves, such that $\#E(\mathbb{F}_q) = q$. "Weak" means, that the ECDLP is relatively easy for these classes of curves.

In order to avoid all attacks against special classes of curves one should select a suitable elliptic curve *at random*. Then the probability of constructing a "weak" curve is negligible. A curve can be selected *verifiably at random* by choosing the coefficients $a$ and $b$ of the defining equation as the outputs of a one-way cryptographic hash function, such as $SHA-1$, according to some pre-specified procedure. There exsist also some other methods for generating cryptographically suitable elliptic curves, which include the *complex multiplication method* and method involving *Koblitz curves*.

One should always *validate* domain parameters before use. Domain parameters validation ensures, that they satisfy all required arithmetical properties. The reasons for performing it are: to prevent the malicious insertion of invalid domain parameters (it can enable some attacks) and to detect coding and transmission errors. Use of an invalid set of domain parameters can void all expected security properties.

## 4.2 Key generation

Each entity in the domain with parameters $(q, FR, a, b, G, n, h)$ does the following:

1. Select a random integer $d$ such that $1 \leq d \leq n - 1$.
2. Compute $Q = dG$.
3. $Q$ is a public key; $d$ is a private key.

Some words about the *public key validation*. A receiver should always perform this procedure to check that the public key satisfies all required arithmetical properties:

1. $Q \neq \mathcal{O}$.
2. $x_Q, y_Q \in \mathbb{F}_q$ with corresponding representation.
3. $Q \in E(\mathbb{F}_q)$, where $E(\mathbb{F}_q)$ is defined by $a$ and $b$.
4. $nQ = \mathcal{O}$.

The reasons for performing it are similar to that of the domain parameters validation.

## 4.3 Signature generation and verification

*Signature generation.* To sign a message $m$ using the key pair $(d, Q)$ one does the following:

1. Select a random integer $k$, $1 \leq k \leq n - 1$.
2. Compute $e = HASH(m)$.
3. Compute $r = x_1 \bmod n$, where $(x_1, y_1) = kG$.
4. Compute $s = k^{-1}(e + dr) \bmod n$.
5. Go to step 1 if $r = 0$ or $s = 0$.
6. $(r, s)$ is a signature for the message $m$.

*Signature verification.* To verify the signature using the public key $Q$ one does the following:

1. Verify that $1 \leq r, s \leq n - 1$.
2. Compute $e = HASH(m)$.
3. Compute $w = s^{-1} \bmod n$.
4. Compute $u_1 = (ew) \bmod n$.
5. Compute $u_2 = (rw) \bmod n$.
6. Compute $(x_1, y_1) = u_1 G + u_2 Q$.
7. Accept the signature if and only if $x_1 = r \bmod n$.

*Correctness.* The algorithm always accepts the true signatures. If a signature $(r, s)$ on a message $m$ was indeed generated using a secret key $d$, then $s = k^{-1}(e + dr) \bmod n$. Rearranging gives

$$k \equiv (e + dr)s^{-1} \equiv (e + dr)w \equiv u_1 + du_2 \bmod n$$

Thus, $u_1G + u_2Q = (u_1 + du_2)G = kG$, and so $r = v$ as required.

*Comparing DSA and ECDSA.* Conceptually, the ECDSA is simply obtained from the DSA by replacing the subgroup of order $q$ of $\mathbb{Z}_p^*$ generated by $g$ with the subgroup of order $n$ of $E(\mathbb{F}_q)$ generated by $G$. The only significant difference between DSA and ECDSA is in the generation of $r$. In DSA $r = (g^k \bmod p) \bmod q$. In ECDSA $r = x_1 \bmod n$, where $(x_1, y_1) = kG$.

# References

D. Johnson, A. Menezes, S. Vanstone: The Elliptic Curve Digital Signature Algorithm (ECDSA). 2001