ISSN 2278-3091
Volume 9, No.4, July – August 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse259942020.pdf
https://doi.org/10.30534/ijatcse/2020/259942020

Elliptic Curves in Modern Cryptographic Systems

Sahun A. V[1]., Lakhno V. A[2]., Kravchuk P. Y.[3]., Kosenko S. S.[4]., Kisiliuk E. M.[5]
[1]National University of Life and Environmental Sciences of Ukraine, Department of Computer systems and networks, Kyiv, Ukraine, avd29@ukr.net
[2]National University of Life and Environmental Sciences of Ukraine, Department of Computer systems and networks, Kyiv, Ukraine, lva964@gmail.com
[3]Private higher educational institution "European University", Kyiv, Ukraine, p.kr@ukr.net
[4] State Scientific Research Forensic Center of the Ministry of Internal Affairs of Ukraine, Kyiv, Ukraine, sveta_kosenko@bigmir.net
[5] Head of the Professional Training Division of the Department of Personnel of the Ministry of Internal Affairs of Ukraine, Kyiv, Ukraine, kisilyk@ukr.net

ABSTRACT

Modern cryptographic algorithms used in cryptocurrencies applications, EDS systems, and blockchain technologies use a mathematical apparatus based on elliptic curves (EC). It is known that not every EC, described by elliptic functions can be used in practice. This is due to the requirements for cryptographic stability. For EC-based cryptosystem algorithms, it is necessary to use subgroups of high-order points. It is theoretically clear that the parameters a, b, k of random ECs can be chosen by the attacker arbitrarily. These curves, used to generate public and private keys, exist in the public domain. These ECs are described in public libraries. It is important to know whether it is possible to predict in practice the values of the secret parameter k, which depend on a particular EC, for all signatures or random number generators used in asymmetric cryptographic algorithms and systems.

Key words: elliptic curves, cryptocurrencies, asymmetric cryptosystem, electronic signature, blockchain.

1. INTRODUCTION

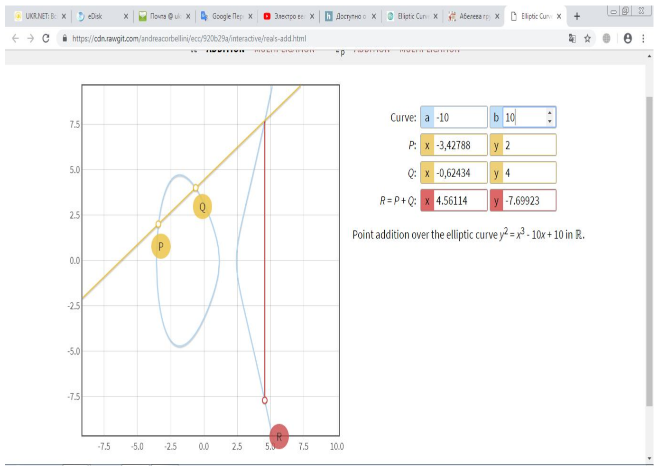
ECC is a type of cubic curve whose solutions are bounded by a space that is typologically equivalent to a geometric shape of the torus type [1]. Although there are a large number of cryptographic systems based on EC [2 – 4], to limit the review of the theoretical part, it is need to consider the following: elliptic curves over real numbers and group law; elliptic curves over finite fields and the problem of discrete logarithm; key pair generation and EC algorithm – ECDSA.
Practical cryptographic systems do not use so-called "special" EC (curves for which there is a feature of mathematical diversity). For example, there are points on them where the tangent space to diversity ones, cannot be correctly defined. Such curves include the followings: EC, intersecting themselves; curves with turning points etc.
Thus, for applied cryptographic applications, elliptic curves of the following form are used: $y^2 = x^3 + ax + b$, where *a*, *b* – coefficients on which the shape of the elliptic curve depends [1-2]; $4a^3 + 27b^2 \neq 0$ is a condition that is necessary in order to exclude "special" ECs.
For use in EC-based cryptosystems, let's take the case that part of a curve is an infinitely distant point (also known as an ideal point). Next, we denote the infinitely distant point by the symbol 0 (zero).
If it is necessary to explicitly consider a point at infinity, then the definition of the elliptic curve can be written as following expression (1) [5]:

$$\left\{(x,y)\in \mathrm{R}^2 \middle| y^2 = x^3 + ax + b, 4a^3 + 27b^2 \neq 0\right\} \cup \{0\} \tag{1}$$

Not all ECs are complete and can be practical used for cryptographic transformations [5-8]. The equation in $y^2 = x^3 + ax + b$ form is called the usual Weierstrass formulation for ECC. It is also a common formulation of ECC. To use in EC-based cryptosystems, take a case when that part of the curve is an infinitely distant point (also known as a perfect point). Next, denote an infinitely distant point by the symbol 0 (zero).
It should be noted that in cryptosystems on the basis of EC property of addition (last) requires the presence of only three points on one line, and the order of arrangement of these three points is irrelevant. This means that if the three points *P*, *Q* and *R* lie on one line, we can get geometric interpretation of inequality appearance for these three points of EC's equation:

$$P + (Q + R) = Q + (P + R) = R + (P + Q) = \cdots = 0. \tag{2}$$

The expression (2) has geometric interpretation, shown below in Figure 1, using online mean [9]. That is, this operator has the properties of associativity and commutativity (belongs to the Abelian group).



Curve: a -10 b 10

P: x -3,42788 y 2

Q: x -0,62434 y 4

$R = P + Q$: x 4.56114 y -7.69923

Point addition over the elliptic curve $y^2 = x^3 - 10x + 10$ in $\mathbb{R}$.

Figure 1: Geometric interpretation of inequality of appearance $P + Q + R = 0$ for $a = -10$, $b = 1$

Thus, to have the possibility for getting sum of two arbitrary points calculation, it is necessary to consider geometrical rather than algebraic addition. The geometric way of adding points is effective for calculating R, but there are a number of partial cases considered, including in [5], [6], [8].

On the one hand, EC-based cryptosystems should be sufficiently cryptocurrency, and on the other hand, should be relatively non-resourceful for use in computer-based hardware resources.

For example, in addition to adding, it can define a scalar multiplication operation for points of the EC group, which is formed according to the expression:

$$nP = \sum_{i=1}^{n} P, \qquad (3)$$

where n is a natural number.

From expression (3) it is obvious that computing the product n and P – numbers requires n additions. If n consists of k decimal digits, then the algorithm will have complexity, as $O(2k)$, which increases the computational complexity [10].

However, there are much faster algorithms. One of these is the doubling-addition algorithm [11]. The principle of his work is written below.

Take for example the number n=151. In a system of calculus modulo 2 it has the following form: 10010111_2. This form can be written as a polynomial – the sum of the degrees of two: 151=1*27 + 0*26 + 0*25 + 1*24 + 0*23 + 1*22 + 1*21 + 1*20. From the above, can be written: 151**P*=27*P* + 24*P* + 22*P* + 21*P* + 20*P*.

As a result, of this algorithm, it is possible to calculate 151**P* after completing only seven doubles and four additions. If doubling and adding are $O(l)$ operations, then the above algorithm has complexity $O(\log n)$ (or $O(k)$ – considering the bit length). Accordingly, such an algorithm has much less computational complexity than the original one $O(n!)$.

Naturally, there is a problem of choosing the correct EC for practical applications.

2. REVIEW AND ANALYSIS OF PREVIOUS STUDIES

The choice of an arbitrary EC in practice can lead to compromising the results of applied cryptocurrency or to the collision of cryptocurrency generation. Some of the problems listed above is known that the problem of discrete logarithm refers to "complex" problems [12], [13]. There are certain classes of EC, which are quite weak in terms of cryptographic stability. There are special discrete logarithm algorithms for solving such problems [11]. For example, all curves for which the order of the final field is equal to the order of EC are vulnerable to a Smart attack, which can be used to solve discrete logarithms for polynomial time on classical computers [14], [15]. In some researches is noted that at known parameters of the domain of EC definition there is a probability that the cryptanalyst has found out previously unknown new class of weak EC, and, probably, has created "fast" algorithm of calculation of discrete logarithms for such "weak" curve [11], [14–16]. In this case, there is a question: would be cryptanalyst can convince the owner of the cryptosystem to the contrary, that is, that he knows nothing about the vulnerability. And he can guarantees that such an EC is "protected" (the cryptanalyst will not be able to use it for its own attacks).

3. FORMULATION OF THE PROBLEM

The problem of compromising the cryptosystem on EC is based on the assumption that for given numbers n and P there is at least one polynomial calculation algorithm with complexity $Q = nP$. The inverse problem was also built on the previous assumption. The inverse problem (the logarithm problem) is formulated as if the points Q and P are known, then it is necessary to define n [10], [17]. The use of the term "logarithm" instead of the term "divisibility" is used for consistency with other cryptosystems.

Now there is not any "simple" algorithm for solving the logarithm problem. All of them have the exponential complexity of discrete algorithm calculating. However, when experimenting with multiplication, there are some regularities [17].

For example, take the ECC of the form $y^2 = x^3 - 3x + 1$ and the point $P = (0;1)$. It can be assured that if the number n is odd, then nP is on the curve in the left half plane, and if n is even, and then nP is in the right half plane.

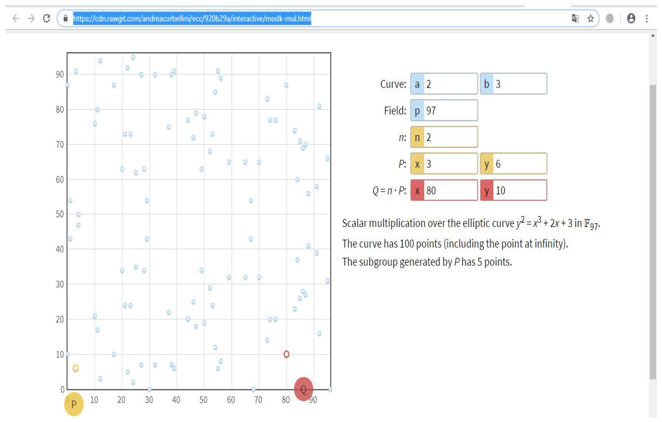
We can find other patterns that lead to the creation of an algorithm for the effective calculation of the logarithm of this curve, that is, to compromise the cryptosystem on the ECC. Along with there is a variation of the logarithmic problem: the discrete logarithm problem [10], [17]. As we know the area of definition of elliptic curves decreases, scalar multiplication remains "simple" and obtaining a discrete logarithm becomes a "difficult" task for compromising the crypto-algorithm [1], [2], [5–8]. Such quality is a key feature of cryptography on elliptic curves.

Next, we investigate the finite fields and problems of discrete

logarithmization, as well as examples and tools for experimental implementation. One knows that the EC, defined above the finite field, has a finite number of points [1–3]. Only the number of points on this curve is given. The number of points in an EC group is called the EC group order. Checking of all possible values for x in the range from 0 to $p-1$ is not a feasible way of counting points because it requires $O(p)$ steps, but if p is a prime number, then such task is considered to be computationally "complicated". That is why, to calculate the order of the EC group a faster algorithm – the Schuf algorithm is used [7], [8].
For real multiplication numbers, we can write the following expression: $nP = \sum_{i=1}^{n} P_i$, where n is a natural number. It is possible to use a doubling-addition algorithm to perform multiplication depending on time like that: $O(k)$, where k – numbers of bits n.
Multiplication of points for EC over a finite field F_p has one property. To demonstrate it we give EC's form $y^2 = x^3 + 2x + 3$ (mod 97) with point $P = (3,6)$. Next, with the help of [9] we calculate all quantities, multiples P (as in Figure 2) and EC has 100 points. All values are multiples of $P = (3,6)$ are 5 different points: $(0, 2P, 3P, 4P)$ and repeated in chain. This property of scalar multiplication for EC is similar to scalar addition in modular arithmetic [5], [10].
We will note two features of this EC: only five values of point P multiplication; the points are repeated cyclically beginning with any integer number *k*, which responds to the following valid form:

$$\begin{cases} 5kP = 0; (5k+1)P = P; (5k+2)P = 2P; \\ (5k+3)P = 3P; (5k+4)P = 4P \end{cases} \qquad (4)$$

Expression (4) shows that the 5 points taken on the EC are closed according to the operation of addition, i.e. the sum of 0,2*P*, 0,3*P* or 4*P* will always be one of these 5 points.


https://cdn.rawgit.com/andreacorbellini/ecc/920b29a/interactive/modk-mul.html

Curve: a 2 b 3

Field: p 97

n: n 2

P: x 3 y 6

$Q = n \cdot P$: x 80 y 10

Scalar multiplication over the elliptic curve $y^2 = x^3 + 2x + 3$ in $\mathbb{F}_{97}$.
The curve has 100 points (including the point at infinity).
The subgroup generated by *P* has 5 points.

Figure 2: Results of scalar multiplication over the EC of the form $y^2 = x^3 + 2x + 3$ (mod97) for point $P = (3,6)$

It means all others of EC points will never be the solutions of the system. The same can be generalized to all other EC points, not only for point $P = (3,6)$.
So, we can take P in the following form:

$$nP + mP = (n+m)P = (n+m)\sum_{i=1}^{m} P_i \qquad (5)$$

Expression (5) means that when we add two values, multiples to P, we get values that are multiples of P (that is, values that are multiples of P, closed with respect to the addition operation). This is sufficient to prove that the set of multiple points of P values is a cyclic subgroup of the group formed by EC. From the above it can be argued that: 1) subgroup is a group that is a subset of another group; 2) a cyclic subgroup is a group whose elements are cyclically repeated; 3) the point P is called the generator or base point of the cyclic subgroup.
Cyclic subgroups are the basis for cryptosystems based on elliptic curves. To determine the order of the subgroup of points generated by the point P, it is impossible to use the Schuf algorithm, because this algorithm works only for integer elliptic curves, but not for subgroups [7], [8].
To determine the order of a subgroup, one must know the facts that it is need to determinate the order of the group of points EC as the number of points of the group. However, for cyclic subgroups the order of P is also the minimum positive integer *n* such as $nP = 0$.
The example above shows in Figure 3) that the subgroup consists of five points, therefore $5P = 0$.
The order of P is related to the EC order by the Lagrange theorem, in according to which the order of the subgroup is the divisor of the initial group order. In other words, if the EC contains N points and one of the subgroups contains *n*, then *n* is a divisor of N.
The two facts above make it possible to determine the order of a subgroup with a base point *P* in the following way: 1) we calculate the order of the elliptic curve N using the Schuf algorithm; 2) find all divisors for N; 3) for each divisor n of order N we compute nP; 4) the least *n* such as $nP = 0$ is the order of the subgroup.
For example, EC in form $y^2 = x^3 - x + 3\,(mod 97)$ over the field F_{97} has an order of $N = 42$. Subgroups of such EC can have order like *n*=1, 2, 3, 6, 7, 14, 21 or 42. If we set point $P = (2,3)$ in the equation, then $P \neq 0, 2P \neq 0, .., 7P = 0$ and subgroup order of point P will be $n = 7$.
For implementation in applicated cryptosystems on EC it is necessary to choose the smallest divisor [18].
Typically, EC-based cryptosystems are asymmetric [1], [5–6], [10], [19]. Thus, they have two types of keys: private and public. A private (secret) key is a random integer number *d*, chosen from range $\{1,..,n-1\}$, where n – EC point subgroup order. Public key is a point $H = d_G$, where G – basic EC subgroup point.
If numbers d and G are known (along with other parameters

of the EC definition area), then public key H determination is not a problem. However, if just only values H and G are known, then private (secret) key d searching converts into a "complex" task. Solving such a problem requires the discrete logarithm calculation.

4. TASK FORMULATRION

To solve the problem of discrete logarithm, it is sometimes necessary to use an additional parameter of the domain: generating value S (seed) [1], [2], [5–8]. In essence, the generating value is a random number that is used to generate the coefficients a and b or the base point G or both. These parameters are generated by calculating the value of the hash function S [27], [28].

As is well known, it is not difficult to calculate hash functions, but it is "difficult" to inverse it. A simple option is to generate a random EC from a generating value, in which the value of the hash function of a random number is used to calculate various parameters of the elliptic curve. If a crypto analyst or attacker wants to restore a hash function from the parameters of its definition area, it will be necessary to solve a "difficult" problem – inverting the hash function (as in Figure 3).

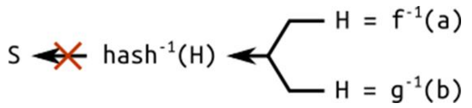


Figure 3: Illustration of the hash-function inversion problem complexity

The elliptic curve generated by the generating value is called randomly verifiable. The principle of using hashes to generate parameters is known in the literature as "nothing up my sleeve", and is widely used in cryptography [20].

The principle of hash generation, described, for example, in [20], provides some guarantee that the EC has been specifically designed to have vulnerabilities known to its author.

If an attacker or a person in the middle of a crypto algorithm gives the victim an elliptic curve along with the generating value, it means that the attacker or person in the cryptoalgorithm cannot arbitrarily choose parameters a and b for EC, and can be relatively sure that the algorithm or will not be able to use special attacks.

Next, consider the ECDSA (Elliptic Curve Digital Signature Algorithm) algorithm, which is a variant of the DSA algorithm. To generate key pairs this algorithm used EC [21]. In process of generation and verification of EDS algorithm (ECDSA) practical implementation, use the Python – based software code. This code contains some parts of the scripted software algorithm ECDH, in particular, scope options and private/public key pair generation algorithm. Here, EC stands for "an elliptic curve secp256k1 from SECG («Standards for Efficient Cryptography Group»)" [22]. Standardized EC, not a simple EC focused on a small number field. Secp256k1 of the SECG group (Standards for Efficient Cryptography Group, Certicom based) was selected as this EC. It has the following EC form of EC, as in Figure 4. This variant of EC was taken due to the fact that the same curve is used in public key crypto algorithms for Bitcoin (for EDS taken) [23].

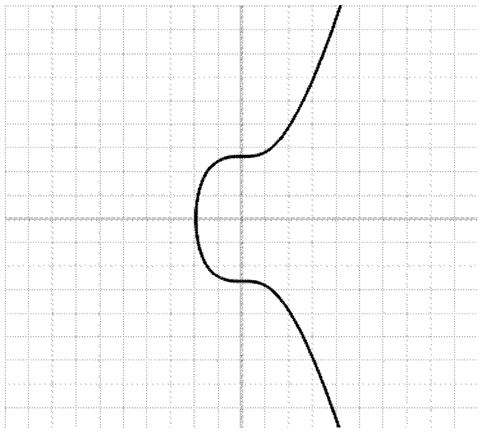


Figure 4: The form of EC secp256k1 of SECG public key crypto algorithm for Bitcoin

The ECDSA algorithm functions with the message hash, not with the message itself. The choice of the hash function is a developer's problem. In practical implementations of this algorithm, a cryptographic hash function is chosen.

The message hash must be truncated so that the bit length of the hash is the same as the bit length of n (subgroup order). A truncated hash is an integer denoted by z.

Cryptosystems on the EC operate in a cyclic subgroup of an elliptic curve over a finite field. Therefore, for their practical implementation require the following parameters: 1) a prime number P, that specifies the size of the final field; 2) coefficients a and b of the EC equation; 3) base point G, which generates a subset of points; 4) order n of subgroup; 5) cofactor h of the subgroup.

The parameters of the EC function definition used to implement the algorithm, described above, are taken from the source code of the OpenSSL cryptographic library [23]. They have the following attributes:

# Field characteristic:
p=0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f
# Curve coefficients: *a*=0; *b*=7.
# Base point:
g=(0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28
d959f2815b16f81798,
0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199
c47d08ffb10d4b8)
# Subgroup order:
n=0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0
364141
# Subgroup cofactor: *h*=1.

If necessary, we can change the EC parameters in the program as follows: field characteristic; curve coefficients; subgroup order; subgroup cofactor and use other curves and parameters of the definition area, but it needs to use only simple numeric fields and the traditional Weierstrass formulation. Otherwise, this program will function incorrectly.

The principle of the ECDSA is: 1) Alice wants to sign the message with her private key (d_A); 2) Peter wants to verify the signature with the public key Alice (H_A); 3) everyone, excluding Alice should not be able to create valid signatures; 4) each of the recipients must be able to verify the signatures.

The algorithm for the signatory (Alice) functions as follows [21]:

Step 1. We get a random integer k , selected from the range $\{1,..,n-1\}$, where n – order of EC point group.

Step 2. Calculate the point $P = kG$, where G – base point of the subgroup.

Step 3. Calculate number $r = x_P \bmod n$, where x_P – the x coordinate of point P .

Step 4. If $r = 0$, then choose another k and go to Step 2.

Step 5. Calculate expression $s = k^{-1}(z + rd_A) \bmod n$, where d_A - private key of Alice; k^{-1} multiplicative inversion of the number k modulo n .

Step 6. If $s = 0$, then choose another number k and return to Step 2.

The resulting key pair (r, s) is the EDS.

To verify a digital signature, the person intending to perform such verification must have Alice's public key H_A , (reduced) hash z and digital signature: (r, s) .

The algorithm for verifying an electronic signature using an existing public key is as follows:

1. Calculate the integer $u_1 = s^{-1} \bmod n$ (6)
2. Calculate the integer $u_2 = s^{-1} r \bmod n$ (7)
3. Calculate a point on an elliptic curve such that $P = u_1 G + u_2 H_A$.

EDS will be valid when the following condition is: $r = x_P \bmod n$.

To check the correctness of the algorithm, we can write that the definition of the public key: $H_A = d_A G$, where d_A – private key), whence, taking into account the expressions (6) and (7):

$$P = u_1 G + u_2 H_A = u_1 G + u_2 d_A G = (u_1 + u_2 d_A) G \quad (8)$$

Taking into account the definitions of u_1 (6) and u_2 (7), we can write the definition of the point P as follows:

$$P = (u_1 + u_2 d_A) G = (s^{-1} z + s^{-1} r d_A) G = s^{-1}(z + rd_A) G \quad (9)$$

Expressions (8) and (9) omit the module of the EC point number system "mod n". This is done in order to reduce the mathematical notation and for the fact that the cyclic subgroup, which is generated by a point G and has order n, eliminates the need for the part "mod n" in the mathematical notation.

We can note, that $s = k^{-1}(z + rd_A) \bmod n$, that is, the r parameter is bound to the message hash. Multiplying both parts of this equation by k and dividing by s , we obtain the following expression (10):

$$k = s^{-1}(z + rd_A) G = kG \quad (10)$$

Substituting the obtained expression (6) in equation (7) for P , we obtain next one (11):

$$P = s^{-1}(z + rd_A) G = kG \quad (11)$$

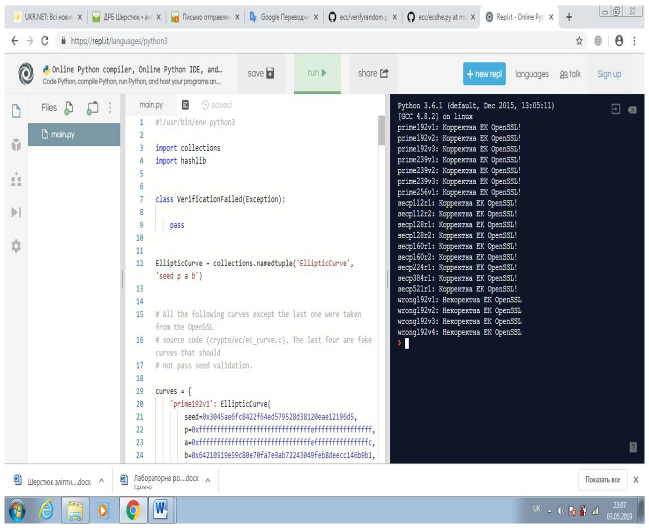
The resulting expression (8) is completely similar to the expression (9) for P , obtained in Step 2 of the DS generation algorithm. When generating EDS and checking them, the same point P is calculated on EC, but by different sets of equations. The principle of checking EDS by this algorithm is based on this effect as well.

5. PRACTICAL SOLUTION

In the practical implementation of cryptosystems based on EC, it is necessary to take into account that there is a standardized algorithm for generating and checking random curves, which is described in the regulatory document ANSI X9.62 [24]. The one based on the cryptographic hash-function SHA-1 [25].

For example, we will check random ECs for the fact that an attacker or a "person in the middle" in a cryptoalgorithm cannot arbitrarily choose parameters a and b for EC. To do this, use the software implementation of the algorithm given by ANSI X9.62 (fig.5) on Python. These curves are used to generate public and private keys. Required EC are in the OpenSSL public library [26].

As in Figure 5, some EC (prime192v1, prime192v2, prime192v3, prime192v4) from the OpenSSL algorithm did not pass the test that this EC was specially created to be the ones, which not having known vulnerabilities to its author.



```
#!/usr/bin/env python3

import collections
import hashlib


class VerificationFailed(Exception):

    pass


EllipticCurve = collections.namedtuple('EllipticCurve',
'seed p a b')


# All the following curves except the last one were taken
from the OpenSSL
# source code (crypto/ec/ec_curve.c). The last four are fake
curves that should
# not pass seed validation.

curves = {
    'prime192v1': EllipticCurve(
        seed=0x3045ae6fc8422f64ed579528d38120eae12196d5,
        p=0xfffffffffffffffffffffffffffffffeffffffffffffffff,
        a=0xfffffffffffffffffffffffffffffffefffffffffffffffc,
        b=0x64210519e59c80e70fa7e9ab72243049feb8deecc146b9b1,
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
prime192v1: Корректна ЕК OpenSSL!
prime192v2: Корректна ЕК OpenSSL!
prime192v3: Корректна ЕК OpenSSL!
prime239v1: Корректна ЕК OpenSSL!
prime239v2: Корректна ЕК OpenSSL!
prime239v3: Корректна ЕК OpenSSL!
prime256v1: Корректна ЕК OpenSSL!
secp112r1: Корректна ЕК OpenSSL!
secp112r2: Корректна ЕК OpenSSL!
secp128r1: Корректна ЕК OpenSSL!
secp128r2: Корректна ЕК OpenSSL!
secp160r1: Корректна ЕК OpenSSL!
secp160r2: Корректна ЕК OpenSSL!
secp224r1: Корректна ЕК OpenSSL!
secp384r1: Корректна ЕК OpenSSL!
secp521r1: Корректна ЕК OpenSSL!
wrong192v1: Некоректна ЕК OpenSSL
wrong192v2: Некоректна ЕК OpenSSL
wrong192v3: Некоректна ЕК OpenSSL
wrong192v4: Некоректна ЕК OpenSSL
>
```

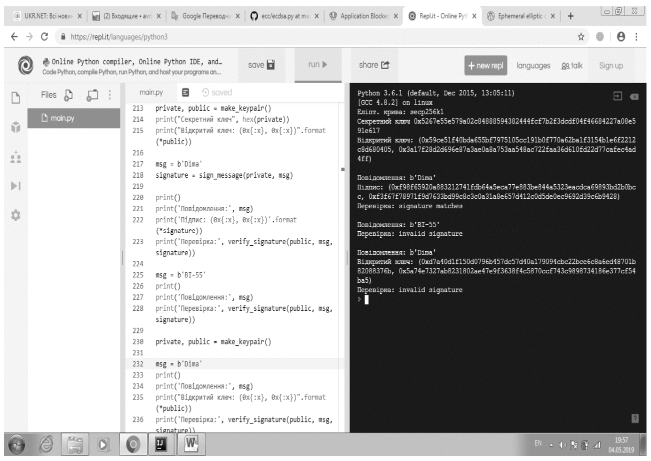
https://repl.it/languages/python3
Online Python compiler, Online Python IDE, and...
save
run
share
+ new repl
languages
talk
Sign up
Files
main.py
saved
Лабораторна ро....docx

Figure 5: The results of checking the EC from the OpenSSL library

Therefore, if the numbers d and G are known (along with other parameters of the domain), then finding the public key – H is not a problem. But, if only the parameters H and G are known, then searching for the private (secret) key d becomes a "complex" problem, because its solution requires the solution of the discrete logarithm problem.

Software realization of EDS based on secp256k1 EC

The created program shows the action of based on ECDSA algorithm while EDS generation and verification are done. This way, the results of public and private keys for messages: “Dima” and “BI-55” were generated and the EDS was verified (as in Figure 6).



```
private, public = make_keypair()
print("Секретний ключ", hex(private))
print("Відкритий ключ: (0x{:x}, 0x{:x})".format
(*public))

msg = b'Dima'
signature = sign_message(private, msg)

print()
print('Повідомлення:', msg)
print('Підпис: (0x{:x}, 0x{:x})'.format
(*signature))
print('Перевірка:', verify_signature(public, msg,
signature))

msg = b'BI-55'
print()
print('Повідомлення:', msg)
print('Перевірка:', verify_signature(public, msg,
signature))

private, public = make_keypair()

msg = b'Dima'
print()
print('Повідомлення:', msg)
print("Відкритий ключ: (0x{:x}, 0x{:x})".format
(*public))
print('Перевірка:', verify_signature(public, msg,
signature))
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
Еліпт. крива: secp256k1
Секретний ключ 0x5267e55e579a02c84888594382444fcf7b2f3dcdf04f46684227a08e5
91e617
Відкритий ключ: (0x59ce51f40bda655bf7975105cc191b0f770a62ba1f3154b1e6f2212
c8d680405, 0x3a17f28d2d696e87a3ae0a8a753aa548ac722faa36d610fd22d77cafec4ad
4ff)

Повідомлення: b'Dima'
Підпис: (0xf98f65920a883212741fdb64a5eca77e883be844a5323eacdca69893bd2b0bc
c, 0xf3f67f78971f9d7633bd99c8c3c0a31a8e657d412c0d5de0ec9692d39c6b9428)
Перевірка: signature matches

Повідомлення: b'BI-55'
Перевірка: invalid signature

Повідомлення: b'Dima'
Відкритий ключ: (0xd7a40d1f150d0796b457dc57d40a179094cbc22bce6c8a6ed48701b
82088376b, 0x5a74e7327ab8231802ae47e9f3638f4c5870ccf743c9898734186e377cf54
ba5)
Перевірка: invalid signature
> 
```

Figure 6: The results of public and private keys for the 2 messages generating and the EDS verifying (ECDSA algorithm)

As in Figure 6, first the program signs the issued message (byte string "Dima") and then checks the received EDS. Then the program checks the same signature for another message ("BI-55"). As the result of verification, we get both correct and incorrect results of comparison. Next, if we try to check the EDS for the correct message with another random public key, we again have failed to.

6. CONCLUSION

As shown in the practical part, when generating EDS and checking them, the same point P on the EC is calculated, but with different sets of equations. The principle of checking EDS by this algorithm is based on this effect. To calculate the parameter s, the inverse of k modulo n is calculated. This technique is guaranteed to work only if n is a prime number. If the subgroup is of non-prime order, then the ECDSA algorithm cannot be used. Due to this fact, all standardized curves have a simple order, and those ECs that do not have a simple order are not applicable to the ECDSA algorithm.

When we generate an EDS using the ECDSA algorithm, it is important to keep the private key k in a location that is not accessible to the attacker. If in the practical implementation parameter k was used for all signatures, or the random number generator would be to some extent predictable, the attacker would be able to identify the private key, which would automatically lead to its use and damage to individuals and businesses. Such cases were occurred and described. This leads to significant and reputational losses, likes listed in [27–29].

For such and similar responsible applications of EC-based cryptographic algorithms, it is highly desirable to use appropriate ECs that have been pre-tested by the algorithms listed in ANSI X9.62.

Thus, the relevance of EC-based EDS cryptosystems today is due to the fact, that today the problem of discrete logarithm for elliptic curves is much more complicated than other similar tasks used in cryptography. This implies that it takes less bits for the integer to get the same level of protection as in other cryptosystems

REFERENCES

1. V. Miller. **Use of elliptic curves in cryptography. CRYPTO.** *Lecture Notes in Computer Science.* 85. pp. 417–426, 1985, doi:10.1007/3-540-39799-X_31.
2. G. Lay, H. Zimmer. **Constructing elliptic curves with given group order over large finite fields. Algorithmic Number Theory Symposium.** *Lecture Notes in Computer Science.* 877. pp. 250–263, 1994, doi:10.1007/3-540-58691-1_64.
3. Alain J. Brizard, **Notes on the Weierstrass Elliptic Function**, *Department of Physics, Saint Michael’s College, Colchester, VT 05439, USA.* October 13 2015, available at https://www.researchgate.net/publication/283335205_Notes_on_the_Weierstrass_Elliptic_Function.
4. G S. Dalbraith, N. P. Smart, **"A Cryptographic Application of Weil Descent". A cryptographic application of the Weil descent.** *Cryptography and Coding. Lecture Notes in Computer Science.* 1746. p. 799, 1999, doi:10.1007/3-540-46665-7_23.
5. Lawrence C. Washington. **Elliptic Curves: Number Theory and Cryptography, Second Edition (Discrete Mathematics and Its Applications).** *Chapman and Hall/CRC; 2 edition*, April 3, 2008, 536 p.
6. Hans Knutson. **What is the math behind elliptic curve cryptography?** *April 6th 2018*, available at: https://hackernoon.com/what-is-the-math-behind-elliptic-curve-cryptography-f61b25253da3.
7. R. Schoof. **Elliptic Curves over Finite Fields and the Computation of Square Roots mod p**. *Math. Comp.* T. 44, No. 170, pp. 483–494, 1985.
8. R. Schoof. **Counting Points on Elliptic Curves over Finite Fields**, *J. Theor. Nombres Bordeaux.* No. 7. , pp. 219–254, 1995.
9. **Elliptic Curve point addition** // *cdn.rawgit.com*, available at: https://cdn.rawgit.com/andreacorbellini/ecc/920b29a/interactive/reals-add.html.
10. Koblitz, Neal. **A Course in Number Theory and Cryptography (Graduate Texts in Mathematics)**, *Springer; 2nd edition*, September 2, 1994, 235 p.

11. Benger, Naomi; van de Pol, Joop; Smart, Nigel P.; Yarom, Yuval (2014). Batina, Lejla; Robshaw, Matthew (eds.). **"Ooh Aah. Just a Little Bit": A Small Amount of Side Channel Can Go a Long Way (PDF)**. *Cryptographic Hardware and Embedded Systems – CHES 2014. Lecture Notes in Computer Science. 8731. Springer.* pp. 72–95. doi:10.1007/978-3-662-44709-3_5.
12. Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman. **An Introduction to Mathematical Cryptography,** *Springer Science+Business Media New York 2014,* p. 53. doi: 10.1007/978-1-4939-1711-2.
13. Johnson, D., Menezes, A. & Vanstone, S. **The Elliptic Curve Digital Signature Algorithm (ECDSA).** *IJIS 1*, 36–63 (2001), available at https://doi.org/10.1007/s102070100002.
14. Nigel P. Smart. **Cryptography Made Simple**. *Springer International Publishing,* 1st Ed: XII, 481 p, 2015, doi:10.1007/978-3-319-21936-3.
15. Arpita Patra & Nigel P. Smart. **Progress in Cryptology – INDOCRYPT,** 2017. *Springer-Verlag.* ISBN 978-3-319-71667-1.
16. Rory Hector, Ramachandran Vaidyanathan, Gokarna Sharma and Jerry L. Trahan, **Optimal Convex Hull Formation on a Grid by Asynchronous Robots with Lights.** C*onference: 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS),* Year: 2020, Page 1051. DOI: 10.1109/IPDPS47924.2020.00111.
17. S., Dummit, David. **Abstract algebra.** Foote, Richard M., 1950- (3. ed.). *Hoboken, NJ: Wiley*OCLC 248917264, 2004.
18. Kefa Rabah. **Review of Methods for Integer Factorization Applied to Cryptography**. *Journal of Applied Sciences*, 6: 458–481 2006, DOI: 10.3923/jas.2006.458.481m, available at https://scialert.net/abstract/?doi=jas.2006.458.481.
19. Christof Paar, Jan Pelzl, Bart Preneel. **Understanding Cryptography: A Textbook for Students and** *Practitioners 1st ed. 2010 Edition,* 390 pp.
20. Schneier, Bruce. **Applied Cryptography, Second Edition,** *John Wiley & Sons,* 1996, available at: https://mrajacse.files.wordpress.com/2012/01/applied-cryptography-2nd-ed-b-schneier.pdf.
21. **An Introduction to Bitcoin, Elliptic Curves and the Mathematics of ECDSA,** N. Minstry, B121555, **Supervisor Dr. B.Winn, Module Code: Mac200,** 21.04.2015, available at: https://github.com/bellaj/Blockchain/blob/6bffb47afae6a2a70903a26d215484cf8ff03859/ecdsa_bitcoin.pdf.
22. **Standarts for Efficient Cryptography. SEC 2. Recommended Curve Domain Parametres. Certicom Research.** January 27, 2010, Version 2.0, available at http://www.secg.org/sec2-v2.pdf.
23. Raval S. **Decentralized Applications. Harnessing Bitcoin's Blockchain Technology,** *O'Reilly Media; 1st edition, 118 p.*, (August 16, 2016).
24. **ANSI X9.62 Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).** *ANSI.* 16 November 2005. 163 p.
25. **RFC 3174. US Secure Hash Algorithm 1 (SHA1).** *Network Working Group. Cisco Systems.* September 2001, available at https://tools.ietf.org/html/rfc3174
26. R. Barnes; M. Thomson; A. Pironti; A. Langley (June 2015). **"Deprecating Secure Sockets Layer Version 3.0",** available at: https://tools.ietf.org/html/rfc7568.
27. Fildes J. **iPhone hacker publishes secret Sony PlayStation 3 key** / *BBC News.* – 2011, available at https://www.bbc.com/news/technology-12116051.
28. B. Akhmetov, V. Lakhno, Y. Tkach, A. Adranova, G. Zhilkishbayeva. **Problems of Development of a Cloud-Oriented Educational Environment of the University,** *International Journal of Advanced Trends in Computer Science and Engineering (JATCSE),* Vol. 9, No.2, pp. 2196–2203, 2020. doi: 10.30534/ijatcse/2020/196922020
29. V. Lakhno, V. Malyukov, B. Satzhanov, A. Tabylov, T. Osypova, Yu. Matus, **The Model to finance the Cyber Security of the Port Information System**, *International Journal of Advanced Trends in Computer Science and Engineering (JATCSE)*, Vol. 9, No.3, pp. 2574–2581, 2020. doi: 10.30534/ijatcse/2020/14932020.