Implementing Elliptic Curve Cryptography

Leonidas Deligiannidis
Wentworth Institute of Technology
Dept. of Computer Science and Networking
Boston, MA 02115, USA
deligiannidisl@wit.edu

***Abstract*—The strength of public key cryptography utilizing Elliptic Curves relies on the difficulty of computing discrete logarithms in a finite field. Diffie-Hellman key exchange algorithm also relies on the same fact. There are two flavors of this algorithm, one using Elliptic Curves and another without using Elliptic Curves. Both flavors of the algorithm rely on the difficulty of computing discrete logarithms in a finite field. Other public key cryptographic algorithms, such as RSA, rely on the difficulty of integer factorization. Both flavors of Diffie-Hellman key exchange algorithm will be discussed in this paper, and we will show implementation details of both of them. Additionally, we will describe what Elliptic Curve Cryptography (ECC) is, and how we can implement different cryptographic algorithms in java, such as digital signatures, encryption / decryption and Key exchange. We will not utilize the java built-in implementations of ECC. Instead, we will use the java programming language as a platform to implement several cryptographic algorithms from the ground up, thus revealing the details of each algorithm and the proofs and reasons these algorithms work. We will describe the theory of ECC and show implementation details that would help students, practitioners, and researchers understand, implement and experiment with such algorithms.**

Keywords—Elliptic Curve Cryptography, Implementation, Network Security.

I. Introduction

The strength of public key cryptography utilizing Elliptic Curves relies on the difficulty of computing discrete logarithms in a finite field. Diffie-Hellman key exchange algorithm also relies on the same fact. There are two flavors of this algorithm, one using Elliptic Curves [1] and another without using Elliptic Curves [2]. Both flavors of the algorithm rely on the difficulty of computing discrete logarithms in a finite field. Other public key cryptographic algorithms, such as RSA [3], rely on the difficulty of integer factorization. Both flavors of Diffie-Hellman key exchange algorithm will be discussed in this paper, and we will show implementation details of both of them. Additionally, we will describe what Elliptic Curve Cryptography (ECC) is, and how we can implement different cryptographic algorithms in java, such as digital signatures, encryption / decryption and Key exchange. Many [4][5][6][7] implemented these algorithms in Java utilizing java's built-in libraries and they focus more on performance. We will not utilize the java built-in implementations of ECC and we will focus more on functionality instead of performance. Instead, we will use the java programming language as a platform to implement several cryptographic algorithms from the ground up, thus revealing the details of each algorithm and the proofs and reasons these algorithms work. We will describe the theory of ECC and show implementation details that would help students, practitioners, and researchers understand, implement and experiment with such algorithms. We will not evaluate the strengths of each algorithm and the technologies using ECC. A great resource for comparing these algorithms and technologies and discusses their vulnerabilities can be found in [8].

Elliptic Curves (EC) are curves that are also naturally groups. They can be used to form a group that is defined by custom arithmetic operations on its elements. We will first describe these operations geometrically and then algebraically over real numbers. Since ECs will be used to explain cryptographic algorithms, we will focus on ECs over an underlined field $\mathbb{F}_p$ (where p is a prime number); working only with whole numbers. The Elliptic Curve Discrete Logarithm Problem (ECDLP) is the discrete logarithm problem for the group of points on an EC over a finite field. The best known algorithm to solve ECDLP is exponential (or at least sub-exponential since it is faster than exponential in log(p), but slower than polynomial), and that is why EC groups are used for cryptography. In simple terms, the Discrete Logarithm Problem (DLP) is: given an element n in the subgroup generated by a point g, find an integer m satisfying $n = g^m$ or else $m=log_g(n)$. If we are working over a large finite field and are given points P and kP, it is very difficult to determine the value of k. This is called the Discrete Logarithm Problem for elliptic curves and is the basis for the cryptographic applications we will see in this paper.

An Elliptic Curve over real numbers consists of the points on the curve, along with a special point ○, which is called the point at infinity and is the identity element under the addition operation. The EC can be based on Weierstrass's equation [9] which is of the form $y^2=x^3+ax+b$ where x, y, a and b are real numbers. The only constraint for the values of a and b is that we do not want the curve to have repeated factors, or else multiple roots; we want the curve to have distinct roots. In other words, the discriminant of E, $4a^3+27b^2$ must not equal to zero, which guaranties that the curve is regular and there are no points where the first derivatives of the curve are cancelled out; there are no points with two or more different tangents [10]. The number of Points (or else the order of the curve) on an EC mod p denoted by $\#E(\mathbb{F}_p)$ is given by the Scoof's algorithm [11] and provides the range of the number of points which is:

$$p + 1 - 2\sqrt{p} \leq \#E(\mathbb{F}p) \leq p + 1 + 2\sqrt{p}.$$

Elliptic Curve Cryptography (ECC) [12] has been incorporated in a number of frequently used protocols. ECC is

appealing to implementers because it requires smaller key sizes [13] [10] than other public key crypto systems such as RSA [3], while offering the same level of security. This leads to faster and more efficient implementations of algorithms in software and in hardware, and more importantly it enables the design of more energy efficient processors for mobile devices. SSH [14] is a protocol that enables secure remote logins. Key exchange between the server and the clients is implemented using Diffie-Hellman on ECs (ECDH) [1]. Each SSH server has a host key and it is used to authenticate itself to the clients. The clients need to accept and save this key the first time they connect to the server. After the first time, they verify this server host key with the saved host key. The server and the clients, then authenticates themselves by signing a transcript of the key exchange using ECDSA [15] [16] that we will talk about later.

Bitcoin is a distributed peer-to-peer digital crypto currency. It enables users to make online payments directly to other parties without going through a financial institution [17]. To make a payment, a user transfers ownership of bitcoins to another user by attaching his/her ECDSA signature and the public key of the payee at the end of the new transaction.

II. ECC Operations

A. *ECC Operations - geometrically*

To add two distinct points P and Q on an EC where $P \neq -Q$, we draw a line through both points. This line intersects the EC in a third point called -R. Reflecting this -R point on the x-axis yields the point R which is R=P+Q, as shown in Figure 1.

If P = -Q, then P+Q= $\mathcal{O}$, as shown in Figure 2. The cubic curves used in ECC are depressed, which means that the square component of the equation is eliminated. The solutions (r_1, r_2, r_3) to such cubic equations is given by $(x-r_1)(x-r_2)(x-r_3)$. Performing the multiplications we get $x^3-(r_1+r_2+r_3)x^2+number$. Because the ECC curve does not have the x^2 term, this implies that the sum of its roots is equal to zero. Having the equation of a line y=mx+d, we set the two equations equal to find their intersection points. So, $x^3+px+q = mx+d$ which becomes: $x^3+(p-m)x+(q-d) = 0$ and the solutions are a,b,c. Since the sum of these roots equals zero, $a+b$ must equal to $-c$. Thus, the reflection of the $-c$ is our third point on the curve; and this is the reason we need to reflect the resulting point when we perform additions in ECs.

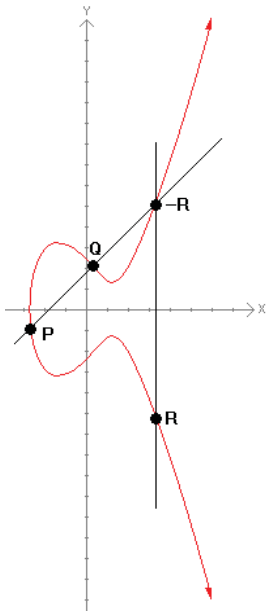

Fig. 1. Adding distinct points P and Q on an Elliptic Curve over $\mathbb{R}$. The result is point R.

Based on addition, we can define multiplication of a point P by a scalar as 2P=P+P, and 3P=2P+P, and 4P=3P+P or 4P = 2P+2P, etc, as shown in Figure 3. Note that when we add a point P to itself, we take the tangent on that point, which intersects the EC on another point, then we reflect that point to get 2P.

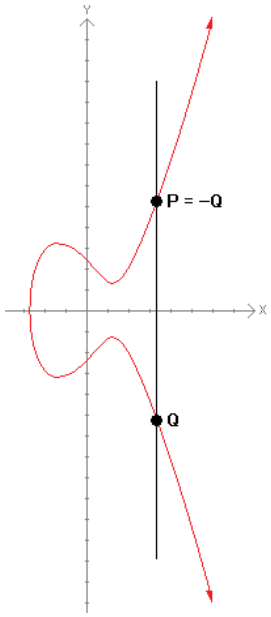

Fig. 2. Adding points P and Q where point P is equal to –Q. The result is the point at infinity $\mathcal{O}$.

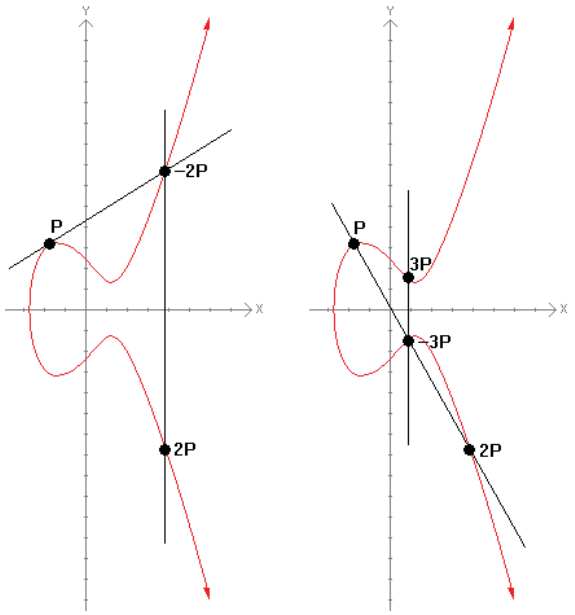

Fig. 3. Doubling the point P produces the point 2P. Adding to that P, produces 3P, etc.

B. *ECC Operations - algebraically*

Adding two points $P,Q \in EC$ with coordinates $P=(x_1,y_1)$ and $Q=(x_2,y_2)$, we get point $R=(x_3,y_3)$. Here we have 3 cases to consider [18]:

1. $x_1 \neq x_2$
2. $x_1=x_2$ and $y_1= -y_2$
3. $x_1=x_2$ and $y_1=y_2$

Case 1. The first case involves the addition of two points P and Q that are not negative of each other ($Q \neq -P$). The slope of the line passing through both points is defined as: $L=(y_2-y_1)/(x_2-x_1)$. The coordinates of point $R(x_3,y_3) = P+Q$ are: $x_3 = L^2-x_1-x_2$ and $y_3 =L(x_1-x_3)-y_1$.

Case 2. The second case involves the addition of two points $P=(x_1,y_1)$ and $Q=(x_1,-y_1)$; Q is the reflexion of point P on the x-axis. We can write this as $P + (-P) = \mathcal{O}$, where $\mathcal{O}$ is the special point at infinity (the line passing through P and Q intersects the EC at infinity. Therefore, (x,y) and (x,-y) are inverses with respect to the elliptic curve addition operation. Here we can also see how we can subtract a point $P=(x_1,y_1)$ from point $Q=(x_2,y_2)$ to produce a point R. We simply reverse the sign of the y-coordinate of point P and we perform addition as it is described earlier.

Case 3: The third case involves the addition of a $P=(x_1,y_1)$ with itself: P+P=R; this is also referred to "point doubling". In this case we assume that $y_1 \neq 0$, otherwise we would be looking at case 2 above. In this case, the slope of the line (the tangent here) is calculated as: $L=(3x_1^2+a) / 2y_1$, where *a* is the variable *a* from the equation of the Elliptic Curve. The coordinates of the point $R=(x_3,y_3)$ are calculated the same way as in case 1. Since we are adding a point to itself the calculations can be simplified to: $x_3 = L^2-2x_1$ and $y_3 = L(x_1-x_3)-y_1$.

III. Elliptic Curves over a Finite Field

To implement cryptographic algorithms we need to work with whole numbers over a finite field, which is an essential property in cryptography; floating point arithmetic is slow and inaccurate due to round off errors. Elliptic Curves over $\mathbb{Z}_p$ can be defined the same way as over $\mathbb{R}$ where p is a prime greater that 3 (in practice p is a large prime number). The equation of the Elliptic Curve now becomes: $y^2 = x^3+ax+b \pmod p$ where $4a^3+27b^2 \pmod p \neq 0$, and *a*, *b* are in the finite field $\mathbb{F}_p$.

The coordinates (x_3,y_3) of the point R=P+Q are calculated the same way as it is shown in case 1 above, but performed with (mod p): $x_3 = L^2-x_1-x_2 \pmod p$ and $y_3 =L(x_1-x_3)-y_1 \pmod p$. The slope L is calculated as follows: If $P \neq Q$ then $L=(y_2-y_1)(x_2-x_1)^{-1} \pmod p$. If P=Q then the slope L is calculated as follows: $L=(3x_1^2+a)(2y_1)^{-1} \pmod p$, where *a* is the variable *a* from the equation of the Elliptic Curve; we can rewrite the calculation of the x-coordinate as: $x_3 = L^2-2x_1 \pmod p$, and y_3 stays the same as $y_3 =L(x_1-x_3)-y_1 \pmod p$.

We explained how we can add two distinct points and how to double a point (adding a point to itself). In other words, P+P = 2P. And 2P+P = 3P, 3P+P = 4P, and we can add 3P to 4P to get 7P, etc. We can think of it as the scalar multiplication operation under the additive notation of a point where 7P = P+P+P+P+P+P+P, adding 7 copies of the point P to itself. The strength of Elliptic Curve Cryptography relies on the fact that given points P and Q such that P = kQ is computationally infeasible to find k, and this is the Elliptic Curve Discrete Logarithm Problem (ECDLP).

A. *Properties of Addition on ECs*

It can be shown [19] that the curve $E(\mathbb{F}_p)$ under point addition is a commutative / abelian group because of the following properties [20]:

- $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in E(\mathbb{F}_p)$, which makes the point $\mathcal{O}$ be the identity under point addition.
- $P + (-P) = \mathcal{O}$ for all $P \in E(\mathbb{F}_p)$, which makes the reflection of a point the inverse of the point.
- $P + (Q+R) = (P+Q) + R$ for all $P,Q,R \in E(\mathbb{F}_p)$, which is the associativity law.
- $P + Q = Q+P$ for all $P,Q \in E(\mathbb{F}_p)$, which is the commutative law.
- $P + Q = R$ for all $P,Q \in E(\mathbb{F}_p)$, $P + Q$ produces result $R \in E(\mathbb{F}_p)$, thus the addition operation is closed on the curve $E(\mathbb{F}_p)$.

The fact that the points on an elliptic curve form an abelian group is behind most of the interesting properties and applications utilizing ECs.

IV. Java Implementation Details

The algorithms presented here are well known and standardized. We will not present the entire implementation of these algorithms (because of space limitations), instead we will provide implementation hints on how one can implement them. For each algorithm we will provide hints on how some of the important steps of the algorithms can be implemented. Full source code is provided for free upon request.

We do not claim that our implementations outperform any other implementations. We simply provide implementation hints in Java in this paper. The source code can be used as a vehicle to deepen our understanding of the inner workings of such security algorithms.

Java's *BigInteger* object is used almost exclusively for implementing these algorithms. This object contains methods that can operate on large numbers (primes or not) needed for all cryptographic algorithms. Its *probablePrime()* method generates prime numbers of specified number of bits. Other methods allow you to add, subtract, multiply, etc. *BigInteger* objects. One of the important methods that is used frequently is the *modPow()* method which allows you to raise a number to a power and then perform the mod operation, all in this single call. Another method that is used as frequently is the *modInverse()* which allows you to perform a mod operation on the inverse of a number. Based on the *BigInteger* object, we created an *ECC_Point* object. This object allows us to perform additions of Points based on the properties of EC discussed earlier. This object enables us to perform numerical operations considering the point at infinity as well. There are two main functions is this object, one to add two points, and the second to multiply a Point by a scalar. We will be using variable names that begin with **BI** to indicated that the data type of a variable is of *BigInteger*, and **ECP** to indicate that the datatype of a variable is of *ECC_Point*; the complete object is provided in the URL mentioned above.

V. Diffie-Hellman Key Exchange Algorithm Using the Template

The Diffie-Hellman key exchange algorithm is used between two parties to exchange keys securely over a public unsecured communication line. Its strength is based on the fact that while it is easy to calculate exponentials modulo a prime number, it is very hard to calculate discrete logarithms. The Diffie-Hellman Key Exchange algorithm between two parties works as follows:

- Both users agree on a number **q** that is a large prime number.
- Both users also agree on a number **a** that is a primitive root of **q** and **a<q**.
- User A selects a private key **XA** that is **XA<q**.
- User B selects a private key **XB** that is **XB<q**.
- User A calculates her public key $\mathbf{YA = a^{XA} \bmod q}$.
- User B calculates her public key $\mathbf{YB = a^{XB} \bmod q}$.
- At this point the two users exchange their public keys.
- User A Calculates the Secret key $\mathbf{K=YB^{XA} \bmod q}$.
- User B Calculates the Secret key $\mathbf{K=YA^{XB} \bmod q}$.

Both users exchanged publicly the **q, a**, and their public keys **YA** and **YB**. However, it is very difficult for an eaves dropper to calculate the discrete logarithm and find either **XA** or **XB** knowing **YA**, **YB**, **a**, and **q**. In actuality, the two parties exchanged information in a public domain to securely calculate the shared secret key **K**, which can then be used in other symmetric algorithms.

Implementation hint: To perform the operation $\mathbf{K=YA^{XB} \bmod q}$ in Java, we can do:

BI_K =**BI**_YA.modPow(**BI**_XB, **BI**_q);

VI. Diffie-Hellman on Elliptic Curves (ECDH)

The Diffie-Hellman algorithm can also be used over Elliptic Curves between two parties to exchange keys. The algorithm works as follows:

- Both users agree on an elliptic curve E over a finite field $\mathbb{F}_q$ such that the discrete logarithm problem is hard in $E(\mathbb{F}_q)$. This implies that both users agree on the Elliptic curve (i.e. $\mathbf{y^2=x^3+ax+b \pmod q}$) which includes the values of **a** and **b**, as well as the prime **q**.
- Both users also agree on a base point $\mathbf{G} \in E(\mathbb{F}_q)$ so that its order is a large prime.
- **n** is the smallest integer such that **nG=O** (where **O** is the point at infinity).
- User A selects a private key $\mathbf{n_a < n}$.
- User A calculates his public key $\mathbf{P_a = n_aG}$.
- User B selects a private key $\mathbf{n_b < n}$.
- User B calculates his public key $\mathbf{P_b = n_bG}$.
- Both users exchange their public keys $\mathbf{P_a}$ and $\mathbf{P_b}$.
- User A calculates her Secret key $\mathbf{k_a = n_a * P_b}$.
- User B calculates her Secret key $\mathbf{k_b = n_b * P_a}$.

At this point $\mathbf{k_a = k_b}$ because $\mathbf{k_a = n_a * P_b = n_a * (n_b * G) = n_b * (n_a * G) = n_b * P_a = k_b}$.

An eavesdropper only sees the curve E, the finite field $\mathbb{F}_q$ and the points **G**, $\mathbf{n_aG}$ and $\mathbf{n_bG}$. If the eavesdropper was capable of solving discrete logarithms in $E(\mathbb{F}_q)$, she could then find $\mathbf{n_a}$ from the points **G** and $\mathbf{n_aG}$.

Implementation hint: To perform the operation $\mathbf{P_b = n_bG}$ in Java, we can do: **ECP**_Pb = **ECP**_G.multiply(**BI**_nb);

VII. Tripartite Diffie-Hellman

The Diffie-Hellman algorithm on Elliptic Curves can be extended so that three participants are involved to derive a shared key, but we will not discuss this algorithm and its implementation in this paper. This technique requires a single exchange of messages between them [21]. It uses the Weil pairing, but a better approach is to use the Tate pairing, to define the function **F**. Let **E** be the super singular curve $y^2=x^3+1$ over $\mathbf{E_q}$ where $q \equiv 2 \pmod 3$. Let **S** be a point on the curve of order **n**. The three users choose a secret number: a, b, c (mod n) respectively. They then calculate their public key as: $\mathbf{P_a}$=aS, $\mathbf{P_b}$=bS, $\mathbf{P_c}$=cS respectively, and they broadcast them. Then each of the three participants computes the shared key as follows, which is the same as $F(S,S)^{abc}$: User A: $F(P_b, P_c)^a$, User B: $F(P_a, P_c)^b$ and User C: $F(P_a, P_b)^c$.

VIII. ElGamal Digital Signatures

A Digital signature is a mathematical operation on a given data where anyone can verify the entity that signed a message. The verification process should be relatively easy to compute but very hard to forge someone else's signature. If one verifies that the signature is valid, this implies that the entity who claims to have signed the document is the one who actually performed

the signage of the document. The following solution relies on the difficulty of computing discrete logarithms over a finite field. Below is the algorithm where User A signs a message and sends it User B. User B then can verify or reject the signature.

- User A selects a generator point $\mathbf{G} \in E(\mathbb{F}_q)$.
- User A selects a secret key **d** (a random number).
- User A computes his public key $\mathbf{PU_A} = \mathbf{dG}$.

Public information: E, $\mathbb{F}_q$, **G**, $\mathbf{PU_A}$.

Message Signature

- User A signs the message **M** by first calculating its hash value: **e**=H(M)
- User A chooses a random number **k** (and keeps it secret) where gcd(k,N) = 1, and $N=\#E(\mathbb{F}_q)$
- User A calculates **P** = **k G**
- User A stores the x-coordinate of point P in **x**
- User A calculates $\mathbf{s} \equiv k^{-1}(e - d\,x) \pmod N$.
- User A transmits the signature **(e, P, s)** followed by the message **M** (User A does not try to keep the message **M** a secret!)

Signature Verification (*needs 3 Point multiplications*)

- User B stores the x-coordinate of point P in **x**
- User B calculates $\mathbf{V_1} = \mathbf{x}\,\mathbf{PU_A} + \mathbf{s}\,\mathbf{P}$
- User B calculates $\mathbf{V_2} = \mathbf{e}\,\mathbf{G}$
- User B accepts the signature as valid if and only if $\mathbf{V_1}$ is equal to $\mathbf{V_2}$.

This algorithm works because:

$\mathbf{V_1} = x\,PU_A + s\,P = x\,d\,G + s\,k\,G = x\,d\,G + k^{-1}(e - d\,x)\,k\,G = G((d\,x) + e - (d\,x)) = e\,G = \mathbf{V_2}$.

Implementation hint: To perform the operation $\mathbf{s} \equiv k^{-1}(e - d\,x) \pmod N$ in Java, we can do:

```
BI_s = BI_k.modInverse(BI_N).multiply(
BI_e.subract(BI_d.multiply(BI_x))).mod(BI_N);
```

IX. The Digital Signature Algorithm (DSA) Based on Elliptical Curves (ECDSA)

The Digital Signature Standard [15] is based on the Digital Signature Algorithm [16]. Recently, the ECDSA [22] algorithm emerged that uses Elliptic Curves. ECDSA is similar to ElGamal's signature scheme but it uses a slightly different signature verification method which makes verification of signatures faster [19]. The main difference between ECDSA and ElGamal's digital signature system is that in ElGamal's system the verification process requires three multiplications of an integer times a point, whereas in ECDSA only two multiplications of an integer times a point are required. These multiplications are the most expensive parts of these algorithms.

- User_A wants to sign a message **m**, which is an integer or most likely the hash of the document to be signed.
- User_A chooses an elliptic curve E over a finite field $\mathbb{F}_q$ where q is a prime number
- User_A chooses a base point $\boldsymbol{G} \in E(\mathbb{F}_q)$ of order $\boldsymbol{n}$; n is the smallest positive integer that nG= O, which is also the number of points on the curve.

Key Generation

- User_A select a random $\mathbf{d} \in [1...n-1]$.
- User_A computes $PU_a = dG$, which is a point on the curve. **d** is the private Key and $\mathbf{PU_a}$ is the public key of User_A.

Public Information: E, $\mathbb{F}_q$, n, G, PU_a.

Signature Generation of message m

- User_A select a random number **k**, $(1 \le k < n)$
- User_A computes P=(**x**,y)=kG and r = **x** mod n. If r is zero restart
- User_A computes e = H(m) H is SHA-1 (160 bit hash)
- User_A computes $s \equiv k^{-1}(e+dr) \bmod n$. If s is zero restart

The Signature is the pair **(r,s)**

Signature Verification

- User_B verifies that r and s are in [1...n-1]
- User_B computes e=H(m)
- User_B computes $w \equiv s^{-1} \bmod n$
- User_B computes $u_1 = ew$ and $u_2 = rw$
- User_B computes $X=(\mathbf{x},y) = u_1G + u_2PU_a$
- If X is the infinity point, User_B rejects the signature, else she computes: v = **x** mod n

User_B accepts the signature iff v is equal to r, since

r = **x** mod n from P=(**x**,y), that User A computes, and

v = **x** mod n from X=(**x**,y), that User B computes.

This algorithm works because:

$\mathbf{X} = u_1G + u_2PU_a = ewG + rwPU_a = es^{-1}G + rs^{-1}PU_a = es^{-1}G + rs^{-1}dG = s^{-1}(e+rd)G = kG = \mathbf{P}$

Implementation hint: To perform the operation $s \equiv k^{-1}(e+dr) \bmod n$ in Java, we can do:

```
BI_s = BI_k.modInverse(BI_N).multiply(
BI_e.add(BI_d.multiply(BI_x))).mod(BI_N);
```

X. ElGamal Public Key Encryption

Public Key Encryption is the mathematical operation to a message that enables only the recipient to decrypt the message. There are two families of encryption algorithms: symmetric and asymmetric. In symmetric algorithms, the same key that encrypts can also decrypt a message. In asymmetric encryption, or else public key encryption, one key encrypts and the other decrypts – one key is kept secret and the other is made public. Generally, public key crypto-systems are slower than symmetric crypto-systems. Therefore, it is common to use public key systems, such Diffie-Hellman, to negotiate and establish a key that is then used in a symmetric crypto-system.

Here is the ElGamal's encryption algorithm where User A encrypts a message and sends it to User B. Then only User B is able to decrypt the recover the original message.

Initialization

- User B chooses a generator point $\mathbf{G} \in E(\mathbb{F}_q)$.

- User B chooses a secret integer **d.** (**d** is user's B private key).
- User B computes $\mathbf{PU_B} = dG$. ($\mathbf{PU_B}$ is user's B public key).

Public Information: E, $\mathbb{F}_q$, G, PU_B

Encryption

- User A expresses her message $M \in E(\mathbb{F}_q)$.
- User A chooses a secret session random integer **k**. It is important that a different k is used each time.
- User A computes $\mathbf{M_1} = kG$
- User A computes $\mathbf{M_2} = M + kPU_B$
- User A sends to User B: (M_1, M_2)

Decryption

- User B computes and recovers $M = M_2 - dM_1$

This algorithm works because:

$M_2 - dM_1 = (M+kPU_B) - d(kG) = M + kdG - kdG = M$

Implementation hint: To perform the operation $M=M_2-dM_1$ in Java, we can do:

ECP_M = **ECP**_M2.subract(**ECP**_M1.multiply(**BI**_d)).mod(**BI**_q);

XI. Massey-Omura Encryption

In Massey-Omura encryption scheme, both users agree on an elliptic curve E over a finite field $\mathbb{F}_q$ such that the discrete log problem is hard in $E(\mathbb{F}_q)$. This algorithm works as follows. User A places a lock on the message M and sends it to User B. User B places another lock on the message and sends it back to User A. User A then removes his lock (leaving the message with only User B's lock on, and sends it back to User B. User B then removes his lock to retrieve the message that User A sent him. Below is the algorithm. It is important to notice here that "removing a lock" requires the multiplication of a point with one's inverse private key. To do this, both users need to know $\mathbf{N}=\#E(\mathbb{F}_q)$. If **d** is the private key, the **d'** is the inverse of **d** in $\mathbb{F}_q$ which **d*d'** = 1.

- User A wants to encrypt a message $M \in E(\mathbb{F}_q)$.
- User A chooses a secret integer $\mathbf{d_a}$ with $\gcd(d_a,N) = 1$.
- User A computes $\mathbf{M_1} = d_aM$ and sends it to User B.
- User B chooses a secret integer $\mathbf{d_b}$, with $\gcd(d_b,N) = 1$.
- User B computes $\mathbf{M_2} = d_bM_1$ and sends it back to User A.
- User A computes $d_a^{-1} \in \mathbb{Z}_n$
- User A calculates $\mathbf{M_3} = d_a^{-1} M_2$ and sends it to User B.
- User B computes $\mathbf{d_b^{-1}} \in \mathbb{Z}_n$
- User B calculates $\mathbf{M_4} = d_b^{-1} M_3$.

$M_4 = M$ because: $M_4 = d_b^{-1} M_3 = d_b^{-1} d_a^{-1} M_2 = d_b^{-1} d_a^{-1} d_bM_1 = d_b^{-1} d_a^{-1} d_b d_aM = M$

Implementation hint: To find the inverse of a number in $\mathbb{F}_q$ of order N, we perform the following (finding d_a^{-1} knowing d_a):

BI_daInv = **BI**_da.modInverse(**BI**_N);

XII. Elliptic Curve Integrated Encryption Scheme (ECIES)

ECIES is another encryption scheme. ECIES is the most extended encryption scheme utilizing ECs. It is defined in ANSI X9.63 [23], IEEE 1363a [24], ISO/IEC 18033-2 [25] and SECG SEC 1 [26]. In [27] the authors compare the different ECIES versions that are implemented by different security organizations and standards. Both users agree on an elliptic curve E over a finite field $\mathbb{F}_q$ so that the discrete log problem is hard for $E(\mathbb{F}_q)$. Both users also agree on a point $\mathbf{G} \in E(\mathbb{F}_q)$. $\mathbf{N}=\#E(\mathbb{F}_q)$ is the number of points on the curve; the order of the curve. We need 2 different hash functions H_1 and H_2 and a symmetric encryption algorithm. An advantage of ECIES over ElGamal's and the Massey-Omura public key encryption methods is that the message M is not represented as a point on the curve, on the contrast it is any sequence of bytes. Here User B wants to send an encrypted message M to user A.

Initialization

- User A chooses a secret integer **d**. (this is the secret key).
- User A computes $PU_A = dG$. (this is the public key).

Public Information: (E, $\mathbb{F}_q$, N, G, PU_A)

Encryption

- User B chooses a random number **k** where: $1 \le k \le N - 1$
- User B computes $\mathbf{R} = kG$
- User B computes $\mathbf{Z} = kPU_A$
- User B computes the hash of R and Z: $\mathbf{R_h}=H_1(R)$ and $\mathbf{Z_h}=H_1(Z)$.
- User B encrypts message M using a symmetric algorithm and the key R_h : $\mathbf{C}=E_{Rh}(M)$
- User B computes the hash of C and Z_h: $\mathbf{t_1}=H_2(C)$ and $\mathbf{t_2}=H_2(Z_h)$
- User B sends to User A: (R, C, t_1, t_2)

Decryption

- User A computes $\mathbf{Z} = dR$
- User A computes $\mathbf{R_h}'=H_1(R)$ and $\mathbf{Z_h}'=H_1(Z)$
- User A computes $\mathbf{t_1}'=H_2(C)$ and $\mathbf{t_2}'=H_2(\mathbf{Z_h}')$
- If t_1 is not equal to t_1' or t_2 is not equal to t_2', reject the cipher-text C and don't even try to decrypt it, else continue.
- User A computes $M = D_{Rh'}(C)$. (Note here that $R_h' = R_h$).

For this algorithm to work, the Z values computed by both users must be the same, and they are because:

(User A computes): $Z = dR = dkG$

(User B computes): $Z = kPU_A = kdG = dkG$

Implementation hint: To compute the Z point, user B can do the following: **ECP**_Z = **ECP**_PU_A.multiply(k);

Conclusion

We presented the underlying number theory of Elliptic Curve Cryptography in a simple to understand way. We explained several algorithms and the mathematical reasons why they work. We also explained how these algorithms can be implemented using the java language and platform. We provide the complete source code of these implementations for researchers and students to experiment with and discover the beauty and elegancy of Elliptic Curve Cryptography.

REFERENCES

[1] U.S. Department of Commerce/National Institute of Standards and Technology (NIST). Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography. Special Publication 800-56A Revision 2, May 13 2013. http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.pdf (retrieved Feb. 2015).

[2] Whitfield Diffie and Martin E. Hellman. "New Directions in Cryptography". IEEE Transactions on Information Theory Vol. 22 (6). p644–654, Nov. 1976.

[3] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Communications of the ACM, Vol.21, Issue 2, pp120-126, Feb. 1978.

[4] Andrew Burnett, Keith Winters, and Tom Dowling, "A Java implementation of an elliptic curve Cryptosystem". Proceedings of the inaugural conference on the Principles and Practice of programming, 2002 and Proceedings of the second workshop on Intermediate representation engineering for virtual machines, (PPPJ '02/IRE '02), 2002 Pages 83 - 88 National University of Ireland Maynooth, County Kildare, Ireland, Ireland ISBN: 0901519871

[5] V. Gayoso Martınez, L. Hernandez Encinas, and C. Sanchez Avila. "A Java Implementation of the Elliptic Curve Integrated Encryption Scheme." The 2010 International Conference on Security and Management (SAM'10), 2010

[6] Johann Großschadl, Dan Page, and Stefan Tillich. "Efficient Java Implementation of Elliptic Curve Cryptography for J2ME-Enabled Mobile Devices." Information Security Theory and Practice. Security, Privacy and Trust in Computing Systems and Ambient Intelligent Ecosystems Lecture Notes in Computer Science Volume 7322, 2012, pp 189-207

[7] V. Gayoso Martınez and L. Hernandez Encinas. "Implementing ECC with Java Standard Edition 7". International Journal of Computer Science and Artificial Intelligence Vol. 3 Iss. 4 pp134-142, Dec 2013

[8] Joppe W. Bos, J. Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, and Eric Wustrow. "Elliptic Curve Cryptography in Practice" In Proc. Of the 18th conf. on Financial Cryptography and Data Security (FC 2014) Christ Church Barbados. March 3-7, 2014. Springer Nov. 8 2014.

[9] Joseph. H. Silverman. "The Arithmetic of Elliptic Curves (2nd ed.), Springer-Verlag, New York, ISBN: 978-0-387-09493-9, 2009.

[10] Hankerson, Darrel, Menezes, Alfred J., Vanstone, Scott. "Guide to Elliptic Curve Cryptography". Springer-Verlag, New York, NY, ISBN: 978-0-387-95273-4, USA, 2004.

[11] René Schoof. "Elliptic Curves over Finite Fields and the Computation of Square Roots mod p". Mathematics of Computation 44, (1985), p483-494.

[12] Victor S. Miller. "Use of Elliptic Curves in Cryptography". Advances in Cryptology – CRYPTO 1985 Proceedings. Lecture Notes in Computer Science Vol. 218 pp417-426, Springer 1986.

[13] I. F. Blake, G. Seroussi, and N. P. Smart. "Elliptic Curves in Cryptography". London Mathematical Society Lecture Note Series Vol. 265. Cambridge University Press, Cambridge, 1999 ISBN 0521-653746. Reprinted in 2000.

[14] D. Stebila and J. Green. "Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer". RFC 5656, Dec. 2009. http://tools.ietf.org/html/rfc5656 Retrieved Feb. 2015.

[15] Digital Signature Standard (DSS), Federal Information Processing Standards Publications FIPS 186-4 July 2013 National Institute of Standards and Technology (NIST) doi:10.6028/NIST.FIPS.186-4 U.S. Department of Commerce http://csrc.nist.gov/publications/PubsFIPS.html , http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf. Retrieved Feb. 2015.

[16] Digital Signature Algorithm US patent 5231668A Jul. 27 1993 Inventor David W. Kravitz. http://www.google.com/patents/US5231668 Retrieved Feb. 2015.

[17] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System. Oct. 2008 https://bitcoin.org/bitcoin.pdf, Retrieved Feb 2015.

[18] Douglas R. Stinson "Cryptography Theory and Practice 3rd Edition Discrete Mathematics and its applications". Series editor Kenneth H. Rosen. Publisher: Chapman & Hall/CRC Taylor & Francis Group Boca Raton London New York, ISBN13: 978-1-58488-508-5, 2006.

[19] "ELLIPTIC CURVES" NUMBER THEORYAND CRYPTOGRAPHY SECOND EDITION, Discrete Mathematics and its Applications" Series Editor Kenneth H. Rosen 2008 by Taylor & Francis Group, LLC. Chapman & Hall/CRC. Taylor & Francis Group 6000 Broken Sound Parkway NW, Suite 300. Boca Raton, FL 33487-2742. Chapman & Hall/CRC is an imprint of Taylor & Francis Group, an Informa business. ISBN: 13: 978-1-4200-7146-7

[20] William Stallings, "Cryptography and Network Security: Principles and Practice" fourth edition. ISBN: 0-13-187316-4 Pearson / Prentice Hall Upper Saddle River, NJ, 2006

[21] Antoine, Joux. "A one round protocol for tripartite Diffie-Hellman". In Algorithmic Number Theory, Ed. W. Bosma, Vol. 1838 of Lecture Notes in Computer. Sci., pp 385–393. Springer- Verlag, Berlin, 2000.

[22] The FIPS 186-4 Elliptic Curve Digital Signature Algorithm Validation System (ECDSA2VS) Updated March 18 2014. National Institute of Standards and Technology (NIST). Information Technology Laboratory. http://csrc.nist.gov/groups/STM/cavp/documents/dss2/ecdsa2vs.pdf Retrieved Feb. 2015.

[23] American National Standards Institute. "Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography". ANSI X9.63, 2001.

[24] Institute of Electrical and Electronics Engineers. Standard Specifications for Public Key Cryptography - Amendment 1: Additional Techniques. IEEE 1363a, 2004.

[25] International Organization for Standardization / International Electrotechnical Commission. Information Technology - Security Techniques - Encryption Algorithms - Part 2: Asymmetric Ciphers. ISO/IEC 18033-2, 2006.

[26] Standards for Efficient Cryptography Group. Recommended Elliptic Curve Domain Parameters. SECG SEC 1 version 2.0, 2009.

[27] V. Gayoso Martínez, F. Hernández Álvarez, L. Hernández Encinas. "Analysis of ECIES and Other Cryptosystems Based on Elliptic Curves". Journal of Information Assurance and Security 6(4) : 285-293 (2011).