Research on Elliptic Curve Crypto System with Bitcoin Curves – SECP256k1, NIST256p, NIST521p and LLL

Mohammed Mujeer Ulla* and Deepak S. Sakkari

Department of Computer Science and Engineering, Presidency University, Bangalore, India
E-mail: Mohammedmujeerulla@presidencyuniversity.in; deepakssakkari@presidencyuniversity.in
**Corresponding Author*

Received 21 April 2022; Accepted 03 January 2023;
Publication 03 March 2023

Abstract

Very recent attacks like ladder leak demonstrated feasibility to recover private key with side channel attacks using just one bit of secret nonce. ECDSA nonce bias can be exploited in many ways. Some attacks on ECDSA involve complicated Fourier analysis and lattice mathematics. In this paper will enable cryptographers to identify efficient ways in which ECDSA can be cracked on curves NIST256p, SECP256k1, NIST521p and weak nonce, kind of attacks that can crack ECDSA and how to protect yourself. Initially we begin with ECDSA signature to sign a message using private key and validate the generated signature using the shared public key. Then we use a nonce or a random value to randomize the generated signature. Every time we sign, a new verifiable random nonce value is created and way in which the intruder can discover the private key if the signer leaks any one of the nonce value. Then we use Lenstra–Lenstra–Lovasz (LLL) method as a black

Journal of Cyber Security and Mobility, Vol. 12_1, 103–128.
doi: 10.13052/jcsm2245-1439.1215

box, we will try to attack signatures generated from bad nonce or bad random number generator (RAG) on NIST256p, SECP256k1 curves. The analysis is performed by considering all the three curves for implementation of Elliptic Curve Digital Signature Algorithm (ECDSA).The comparative analysis for each of the selected curves in terms of computational time is done with leak of nonce and with Lenstra–Lenstra–Lovasz method to crack ECDSA. The average computational costs to break ECDSA with curves NIST256p, NIST521p and SECP256k1 are 0.016, 0.34, 0.46 respectively which is almost to zero depicts the strength of algorithm. The average computational costs to break ECDSA with curves SECP256K1 and NIST256p using LLL are 2.9 and 3.4 respectively.

Keywords: EdDSA – Edwards curve Digital Signature Algorithm, Nonce – number only used once, RAG – Random number generator, NIST – National Institute of Standards and Technology, ISO – International Organization for Standardization, IEEE – Institute of Electrical and Electronics Engineers, ECC – Elliptic curve cryptography, IoT – Internet of Things.

1 Introduction

Due to rapid technological advancements, there has been an excessive amount of sensitive data exchanged in recent years in applications like direct online banking (or third-party applications like Google Pay or Paytm), stock market trading, and remote access to data in the healthcare, defence, automotive, retail, and many other fields. Public-key cryptosystems are used by several internet security protocols to achieve secrecy, integrity, and authentication. Elliptic Curve Digital Signature Algorithm is a public-key protocol that is frequently utilised on the internet (ECDSA). TLS, Open PGP, smart cards and digital currency like Ripple, Ethereum, and Bitcoin are a few application areas of ECDSA. ECDSA is a quick signing algorithm because of its short key size and the difficulty of the discrete logarithm problem. It is recommended by IEEE and NIST since 2000, ANSI since 1999, and ISO since 1998 because of these features [1]. A useful tool in cryptanalysis is lattice reduction. Many cryptosystems like knapsack and RSA are broken using lattice reduction. In addition computations in ECDSA-discrete logarithms and factoring composite numbers are possible using lattice reduction [2, 3]. A LLL algorithm is one of the most popular algorithms for lattice reductions by Lenstra, Lenstra and Lovasz. Many of the lattice algorithms used today are LLL variants. In this paper we focus on applying

LLL algorithm to crack ECDSA on NIST and SECP recommended curves like NIST 256p, SECP256k1 and NISP521p [2]. The paper is organized as follows Section 2 provides an theoretical principle-Elliptic curve digital signature (ECDSA) and the LLL Algorithm. Section 3 is described in three parts, A. ECDSA-Disclosing the private key, if nonce known using NIST256p, SECP256k1, NIST5, B. ECDSA-Disclosing the private key using Lenstra–Lenstra–Lovasz (LLL) method, if nonce known, C. ECDSA-Disclosing the private key using Lenstra–Lenstra–Lovasz (LLL) method, if nonce known with real-world ECDSA bugs. Section 4 demonstrates an analysis of our experimental results and Section 5 summarizes our conclusions and discusses future work.

2 Theoretical Principle

2.1 Recent Trends in Elliptic Curve Cryptography

The majority of IoT services are going to be implemented as real-time embedded systems that significantly rely on security procedures as a result embedded IoT devices must be secure. This work outlines the security issues that system designers must consider while creating safe embedded systems. Implementing public key cryptography (PKC) in embedded system is most challenging [4]. PKC, has reduced key sizes and is based on Elliptic Curve Cryptography (ECC), is effective for both private and public activities. ECC is helpful when you need to integrate security because of the IoT's proliferation of connected embedded devices. According to the comparative study, real-time embedded systems in the Internet of Things with limited resources are best suited for ECC [5, 6].

A growing number of electronic applications in today's technology, such as Internet of Things devices, require secure communication. A popular and efficient public-key cryptosystem is the elliptic curve Diffie Hellman (EC-DH) algorithm. The Diffie-Hellman Key agreement system is one of many key exchange techniques that frequently employ elliptic curves. ECC provides similar security with smaller key sizes as compared to traditional cryptosystems like RSA, which results in lower power consumption, quicker calculations, as well as lesser memory and transmission capacity (bandwidth) reserve [7, 8]. This is especially true and practical for applications like IoT devices, where CPU processing speed, power, and space are frequently limited. The Elliptic Curve Diffie-Hellman (ECDH) Key agreement algorithm, the RSA algorithm, and Diffie-Hellman are all implemented in this work includes software and hardware. Additionally, power, performance, and area analyses and comparisons are part of the proposed effort. The comparison is based on metrics collected after using the 90 nm UMC Faraday library to implement the algorithms in Synopsys. In terms of power and area, the ECDH algorithm is proven to be superior to others [9, 10].

In the Internet of Things (IoT), establishing end-to-end authentication between devices and apps is a difficult task. Existing authentication mechanisms are vulnerable to security threats and can halt the development of the IoT and the realisation of Smart Cities, Smart Homes, and Smart Infrastructure, among other IoT goals, as a result of heterogeneity in terms of the devices, topology, communication, and various security protocols used in IoT [11, 12]. The current authentication schemes and security protocols demand a two-factor authentication mechanism in order to provide end-to-end authentication between IoT devices/applications. So, this paper, explores whether one-time password (OTP)-based authentication is appropriate for the Internet of Things and suggest a scalable, effective, and reliable OTP strategy. The Lamport OTP algorithm and the lightweight Identity Based Elliptic Curve Cryptography technique are the foundations of our suggested solution. Also analyze and test the effectiveness of new scheme, despite having a lower key size and fewer infrastructures, it performs well without sacrificing security. This method is suitable for two-factor authentication between IoT devices, apps, and communications and can be deployed in real-time IoT networks [13].

2.2 Elliptic Curve Digital Signature (ECDSA)

The Elliptic curve digital signature algorithm, abbreviated as ECDSA, is a public key cryptography encryption algorithm. ECDSA keys are orders of magnitude smaller in size than keys generated by any other digital signing algorithm [14]. For example to have 128 bit security using RSA requires 3072 bit key size while ECC requires 256 key sizes. To have a 256 bit security using RSA requires 15360 bit key size while ECC requires 512 key sizes.

The steps in ECDSA are as follows:

Alice computations:

(1) Alice selects his private key $= P$
(2) Alice computes his public key $=$ private key $P * G$ i.e. private key P times G
(3) Alice finds (x, y) coordinates of point $P * G$ i.e. $(x, y) = k * G$, where k is a nonce or random value

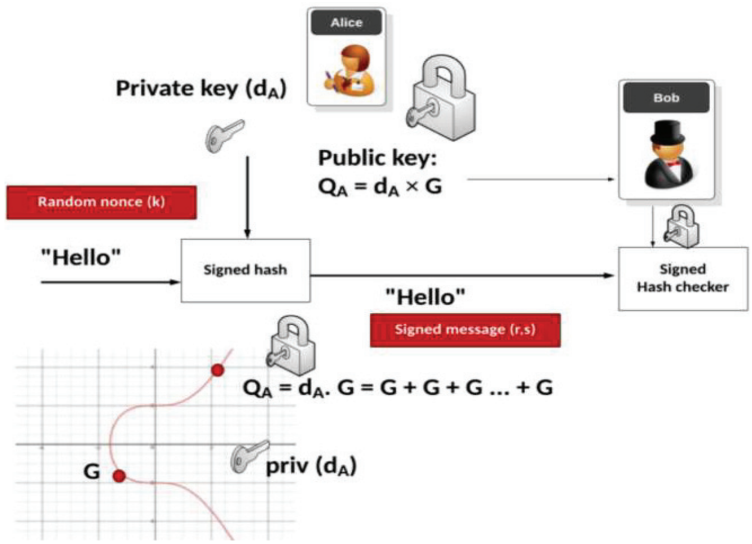

Figure 1 ECDSA.

(4) Alice finds value of r

$$r = x \; Mod \; N \tag{1}$$

(5) Alice generates the signature for the message M that has to be sent for Bob

$$k^{-1}(H(M) + r * privatekey\ P) \tag{2}$$

Bob computations:

(1) Once the Bob receives the signed message from Alice, he computes $u_1 = H(M)s^{-1}$ and $u_2 = rs^{-1}$
(2) Bob computes (x, y) coordinates using u_1, u_2 i.e., $(x, y) = u_1G + u_2(privatekey\ P * G)$
(3) Computations at Bob side

$$\frac{H(m) + r * privatekey\ P * G}{s}$$

$$\frac{H(m) * G + r * privatekey\ P * G}{k^{-1}(H(m) + r * privatekey\ P)}$$

$$\frac{\cancel{(H(m) + r * privatekey\ P)}G}{\cancel{(H(m) + r * privatekey\ P)}k^{-1}}$$

Substituting further we get $k * G$ which is same as what we had obtained in step 1 in Alice computations [3].

2.3 The LLL Algorithm

The Lenstra, Lenstra, and Lovasz (LLL) algorithm is a powerful tool for locating sufficiently orthogonal bases [15, 16]. The LLL algorithm is conceptually divided into two parts:

- Subtracting multiples of the current basis vectors from a non-basis vector (working vector).
- Choosing whether to make the working vector the next basis vector or whether it should take the place of the basis vector right before it.

Based on whether the Lovasz criterion is satisfied, this choice will be made. In general, the Lovasz requirement establishes whether the working vector is large enough to serve as the subsequent basis vector [17]. We monitor two groups of vectors:

- $\vec{v}_1; \ldots,$ an attempt to minimize the existing set of basis vectors to a roughly orthogonal set.
- $\vec{v}_1^*$, $\vec{v}_2^*$,..., the collection of orthogonal basis vectors created by the Gram-Schmidt reduction.

k, the number of the working basis vectors we are attempting to minimise, is another important parameter to monitor. Assume our basis vectors are $\vec{v}_1^*$, $\vec{v}_2^*, \ldots \vec{v}_{k-1}^*, \vec{v}_k^*, \ldots$ and we are attempting to reduce $\vec{v}_k$. We reduce this by subtracting multiples of $\vec{v}_1, \vec{v}_2, \ldots \vec{v}_{k-1}$. Now consider the vectors $\vec{v}_{k-1}$ and $\vec{v}_k$, we might need to subtract only if we have two vectors, some multiple of new vector $\vec{v}_k$ from old vector $\vec{v}_{k-1}$. This requires swapping $\vec{v}_{k-1}$ and $\vec{v}_k$. But since we have a new $k - 1$ vector we need to go through the whole process again, this time with $\vec{v}_{k-1}$ with as new working vector. The decision of whether to swap $\vec{v}_{k-1}$ with $\vec{v}_k$ and make $\vec{v}_{k-1}$ the working vector is based on whether the Lovasz condition is satisfied. In addition to basis vectors $\vec{v}_1^*$, $\vec{v}_2^*, \vec{v}_3^*, \ldots$ found from the Gram-Schmidt reduction. Let $\vec{v}_k$ be the working vector and let

$$\boldsymbol{\mu}_{k,k-1} = \frac{\vec{v_k} * \vec{v}_{k-1}^*}{\vec{v}_{k-1}^* * \vec{v}_{k-1}^*}$$

If $||\vec{v}_k^*||^2 = (\frac{3}{4} - \mu_{k,k-1}^2)$ then we are done with $\vec{v}_k$ for now and can make $\vec{v}_{k+1}^*$ the next working vector, otherwise, swap $\vec{v}_{k-1}$ and $\vec{v}_k$ and make $\vec{v}_{k-1}$ the working vector [18–20].

3 Methodology

3.1 ECDSA-Disclosing the Private Key, If Nonce Known Using NIST256p, SECP256k1, NIST521

In this section let us use ECDSA, private key, nonce value and how we can possibly derive the private key if we know the nonce value that is being used to create the signature. Initially the communication between Alice and Bob begins with Alice having her private key P and public key i.e., private key P * G. The process of obtaining private key is as follows, with elliptic curve cryptography we have curve with equation of form $y^2 = x^3 + ax + b$ Mod N. All the points on the curve what we get are from 0 to N − 1 [21]. The curve itself is defined by values of a, b and large prime number N. We initially select a point on curve called as generator point G and we add M number of times with itself until we get another point on elliptic curve which we call it as private key i.e. $G + G + G + \cdots + G$, the private key is a 256 bit random value [22, 23]. The public key happens to be the (x, y) coordinates of point M * G or simply M times G. Once Alice selects her private key P and

Table 1 ECDSA: Disclosure of the private key, if known nonce (NIST-256p recommended parameters)

N=115792089210356248762697446949407573530086143415290314195533631308867097853951
a=-3
b=41058363725152142129326129780047268409114441015993725554835256314039467401291
h=1
Order:115792089210356248762697446949407573529996955224135760342422259061068512044369
G_x=48439561293906451759052585252797914202762949526041747995844080717082404635286
G_y=36134250956749795798585127919587881956611106672985015071877198253568414405109
Message 1: Journal of Cyber Security and Mobility
Sig1(R,S): 22012898957236689703754513423415920854839883197556881026747687864608172515076324787754002326060342354892015964169418601263373897491548113917969154534 2485
Private Key: 95496264190673951577435564680237507319016551826879163101576563045934039929932
The private key is found: 95496264190673951577435564680237507319016551826879163101576563045934039929932

Table 2 ECDSA: Disclosing the private key, due to weak nonce (SEC-256K1 recommended parameters)

N=115792089237316195423570985008687907853269984665640564039457584007908834671663
a=0
b=7
h=1
Order:115792089237316195423570985008687907852837564279074904382605163141518161494337
Gx=55066263022277343669578718895168534326250603453777594175500187360389116729240
Gy=32670510020758816978083085130507043184471273380659243275938904335757337482424
Message 1: Hello
Sig1(R,S): 7773499647157869072481902530128808492319852184749699500415381153024496572628597413358969903295688400058661785083988603187257053615822443330142691718421644
Random value (k): 63076811158092363886617914846091290891
Private Key: 16122978565960941408252013174486341707774481479068509615634681955105988655192
The private key is found: 16122978565960941408252013174486341707774481479068509615634681955105988655192
48588618394399226405893001917337148111899544979674835399706352006027182977592

computes the public key, she picks up a message that has to be signed with her private key. Using ECDSA, R and S values are used to create a signature for her message. Once the signed message is received by Bob he picks up R and S values along with public key of Alice to determine whether the message is signed by Alice or not.

3.2 ECDSA – Disclosing the Private Key Using Lenstra–Lenstra–Lovasz (LLL) Method, If Nonce Known

In this section we search for private key used to sign a message with ECDSA. In this method we will generate two signatures and find the private key using Lenstra–Lenstra–Lovasz (LLL) method. Despite Alice keeps her nonce

Table 3 ECDSA: Disclosing the private key, if nonce known (NIST-521P recommended parameters)

N=686479766013060971498190079908139321726943530014330540939446345918554318339765605212255964066145455497729631139148085803712198799971664381257402829111505 7151
a=-3
b=1093849038073734274511112390766805569936207598951683748994586394495953116150735016013708737573759623248592132296706313309438452531591012912142327488478985984
h=1
Order:6864797660130609714981900799081393217269435300143305409394463459185543183397655394245057746333217197532963996371363321113864768612440380340372808892707005449
Gx=2661740802050217063228768716723360960729859168756973147706671368418802944996427808491545080627771902352094241225065558662157113545570916814161637315895999846
Gy=3757180025770020463545507224491183603594455134769762486694567779615544477440556316691234405012945595621444445372894285225856667291965808101243442775783 76784
Message 1: Hello
Sig$_1$(R,S): 118912407987803730594927821196302530155934634170726309186953432271081694067943402040651927711372576729242699318709436457201951428350265909093262285263237426660261496652760999431458175285473075579506410784311048885831700513533938669254933278708539386364780181671622028749249739407959492723481836257320149389 48 579325
Random value (k): 1345073822754761250886379837 21177218254
Private Key: 5230660367685557512328488121911598620447434539858665179340194878686546186927608644279504288975234655566278162777217776403595523877155872653821143293053140344
The private key is found: 5230660367685557512328488121911598620447434539858665193401948786865461869276086442795042889752346555662781627772177764035955238771558726538211432930531 40344

secret, Eve can easily recover the secret key if Alice uses repeated nonce even for different messages. Let us assume two signatures (r, s_1) and (r, s_2) derived on messages msg_1, msg_2 respectively from same nonce k then r value will remain same for both messages as the k value is same [24]. So Eve would detect the private key as follows:

(1) $Sig_1 = k^{-1}$ (Hash(Msg_1) + xr) and $Sig_2 = k^{-1}$ (Hash (Msg_2) + xr)
(2) Sig_1–$Sig_2 = k^{-1}$ (Hash (Msg_1) –Hash (Msg_2))

(3) K (Sig_1–Sig_2) = Hash (Msg_1)–Hash (Msg_2)
(4) k = (Sig_1–Sig_2)$^{-1}$ (Hash (Msg_1)–Hash (Msg_2)) [8]

Using above formula once we have recovered the nonce k then secret key is recovered using previously described attack. If any nonce for the signature is leaked, then private key can be cracked, and complete signature scheme is broken. In addition to this if any of the nonce is repeated accidentally then accidental repetition of nonce can be easily detected by Eve and can recover the private key by breaking complete encryption scheme. Even leaking fractional parts of nonce can damage signature abruptly. Work by N.A. Howgrave-Graham, N.P. Smart showed the application of lattice attacks to crack DSA from partial leakage of nonce [25]. Further to this Nguyen and Shparlinski continued their work to obtain secret key from 160-bit DSA and then from every 100 signatures in ECDSA secret key was obtained by just knowing three bits of each nonce [26]. Further to the research Mulder et. al. performed more attacks on partial nonce leakage using Fourier transform-based attack and recovered secret keys from 384-bit ECDSA by knowing only five bits from each nonce from 4,000 signatures. Most of us would have heard Minerva attacks which involved several timing side channels were leveraged to recover partial nonce leakage and these lattice attacks. Using enough signatures they were able to obtain private key even if size of nonce was leaked. The latest attack known as Ladder leak attack which is even worse Fourier analysis attack in ECSDA one could obtain secret keys just by having 1 bit of nonce is leaked [27].

Further to it, even if one manages to keep his nonce secret, never leak any of the bits and never repeat a nonce. The work by Heninger and Breitner proved that application of lattice attacks can potentially break the signature scheme implemented using defective random number [28]. One's signature scheme is completely broken if one uses 256-bit ECDSA, if bias of 4 bits is done using 256-bit ECDSA in your nonce, despite not knowing those biased values. In our research we use LLL algorithm as a black box, we will try to attack signatures generated from bad nonce or bad RNG. Such nonce will have fixed prefix i.e. where many of the most significant bits (MSB) will remain same. This attack also works even if most significant bits (MSB) are not fixed bits. We begin with LLL algorithm with an input matrix and the algorithm will generate the output new matrix values. In this input matrix is constructed using a collection ECDSA signatures and the final output by LLL matrix will enable us to obtain ECDSA private key this is the resultant of LLL output matrix which will contain signatures of all nonce. Using obtained

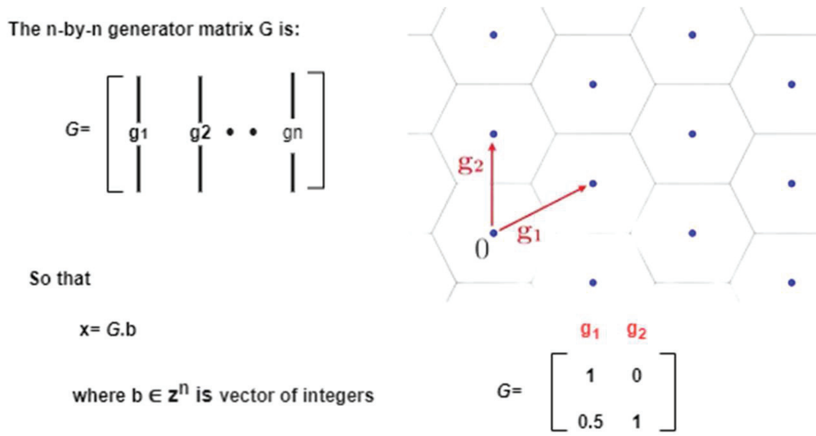

Figure 2 Lattice: linear code over real numbers with N × N generator matrix.

nonce we make use of basic attack described earlier to recover the private key. A LLL basis reduction algorithm is used to approximate the shortest vector in higher dimensional space in polynomial time. It also has applications in cracking many cryptography algorithms, integer programming and number theory because of its accuracy and performance [29, 30]. A lattice λ is an additive subgroup of real numbers and is represented by a basis vector g_1, g_2,… g_n in N-dimensional space. A lattice point X is a linear combination of integral basis vectors.: $X = g_1\ b_1 + g_2\ b_2 + \cdots + g_n\ b_n$ where the b_i are integers. Figure 2 depicts a two-dimensional lattice with two generator vectors, g1 and g2. We arrange the generator vectors and columns so that a lattice point X equals the generator matrix G times B, where B is an integer vector and b_z^n is an integer vector. In Figure 3, we take B to be an integer vector [0, 0], then X equals G times B, and thus the lattice point is 0. In Figure 4 we take B is equal to integer vector [3,−1] then X is equals G times B and therefore we get the lattice point as [3,0.5]. Basis reduction is a technique for reducing the basis B of a given lattice L to a smaller basis B0 without changing the lattice L. Figure 5 depicts a two-dimensional lattice with two different bases. The basis determinant is shaded, and the right basis is reduced and orthogonal [31, 32]. The following are the steps to change the basis while keeping the same lattice.

(1) Firstly, swap the two vectors in the basis.
(2) We use $-b_i$ for a vector $b_i \in B$
(3) We combine additional basis vectors linearly. to $b_i \in B$ vector.

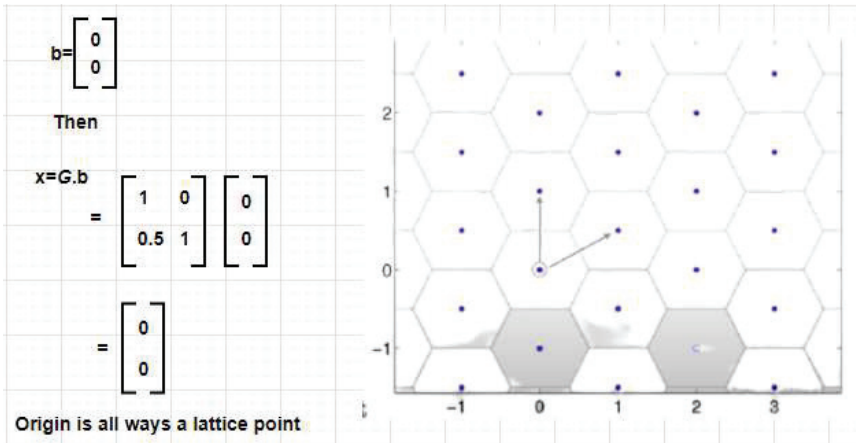

Figure 3 Example 1-Integers to lattice.

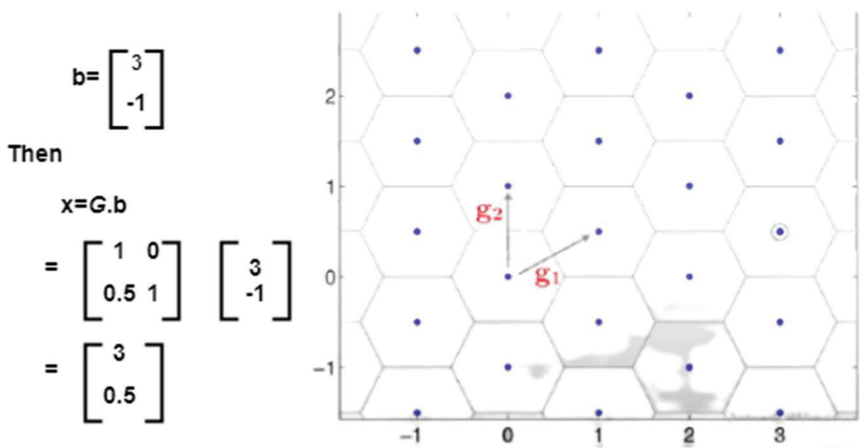

Figure 4 Example 2-Integers to lattice.

In the lattice L, any vector v is represented by

$$v = \sum_{i=0}^{m} z_i b_i$$

We obtain a new basis vector after induction b_j, where

$$b_j = b_j + \sum_{i!=j} y_i b_i,\ y_i Z$$

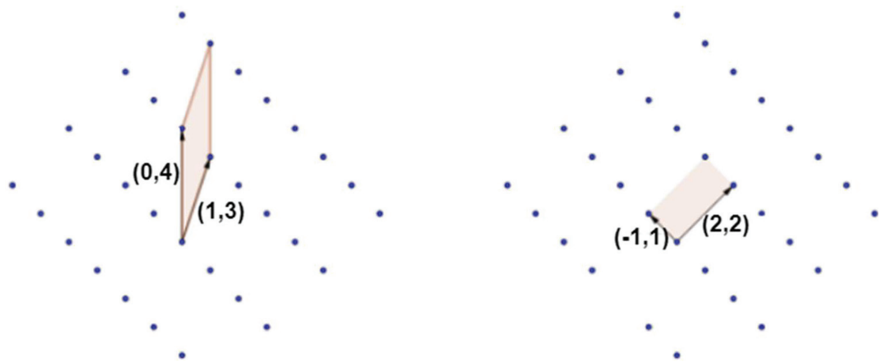

Figure 5 A two dimension lattice with two different basis.

A new basis for a lattice L is represented as

$$\mathrm{v} = \sum_{\mathrm{i!=j}} \mathrm{z_i b_i} + \mathrm{z_j}\left(\mathrm{b_j} + \sum_{\mathrm{i!=j}} \mathrm{y_i b_i}\right)$$

As a result, despite changing the basis lattice, the result is the same.

A Lenstra-Lenstra-Lovasz (LLL) algorithm estimates the shortest vector problem; it runs in polynomial time and finds an approximation to the correct answer within an exponential factor. It is a useful method for solving integer linear programming, factoring polynomials over integers, and breaking cryptosystems [33]. Let $\mathrm{b_1}$, $\mathrm{b_2}$,..., $\mathrm{b_n}$ be a basis for a N-dimensional lattice L, and $\mathrm{b_1^*}$, $\mathrm{b_2^*}$,... $\mathrm{b_n^*}$ be the orthogonal basis and we have

$$\mathrm{u_{i,k}} = \frac{\mathrm{b_k^* b_i}}{\mathrm{b_i * b_i}} \tag{3}$$

The reduced basis of LLL is $\mathrm{b_1}, \mathrm{b_2}, \ldots, \mathrm{b_n}$ if following two conditions are met:

(1) $\forall_i \neq k, u_i, k\frac{1}{2}$.
(2) for each i, $||b_{i+1}^* + u_{i,i+1} b_i^*||^2 \geq \frac{3}{4}||b_i^*||^2$

The constant values between $\frac{1}{4}$ and 1, can ascertain that the algorithm will terminate in polynomial time. The constant chosen here $\frac{3}{4}$ is for simplicity of paper. The second condition highlights the ordering of the basis. Given a basis $\mathrm{b_1}, \mathrm{b_2}, \ldots, \mathrm{b_n}$ in N-dimension space.

The LLL works to get the reduced basis as shown below:

Algorithm 1 LLL algorithm

Input: $b_1, b_2, \ldots, b_n$

Continue both the steps until LLL reduced basis is found

```
Step 1: Gram-Schmidt Orthogonalization
for i = 1 to n do
    for k = i1 to 1 do
      m ← closest integer of u_{k,i}
      b_i, b_i m b_k
    end for
end for
Step 2: Examine the II condition, if true then swap
for i = 1 to n1 do
    if ||b_{i+1} + u_{i,i+1} b_i||^2 ≥ (3/4)||b_i||^2 then
      swap b_{i+1} and b_i
      go to step 1
    end if
end for
```

To perform the attack we use ECDSA and LLL library in python. We chose ECDSA library as it allows us to input our own nonce's. There by allowing us to input nonce's from bad RNG's to validate our attack. This attack is performed on NIST P-256 elliptic curve. We begin by giving input as two signatures obtained from 128-bit nonce's. First signatures are generated then we create the input matrix to LLL algorithm. In the above matrix N is the order of NIST P-256, The upper bound limit set for our nonce's is B (both the nonce's used in our research study are of same 128 bits size), m_1 and m_2 are two input messages and (r_1, s_1) and (r_2, s_2) are the generated signatures for the input message. Once the matrix is ready it is given as input to LLL algorithm, which will output the new matrix. The output matrix will have one of the nonce utilized to obtain two signatures. As discussed earlier the procedure to recover private key after obtaining nonce k. We usually compute r^{-1}(ks–H(m)). Every attacker has an access to public key corresponding to this signature. Therefore one could easily ascertain whether we have found the corresponding private key or not by just computing its corresponding public key and compare it with public key already available. A drawback with this method is there is a noticeable failure rate for this kind

Table 4 ECDSA: Disclosing the private key using Lenstra–Lenstra–Lovász (LLL) method, with bad nonce

N=115792089210356248762697446949407573530086143415290314195533631308867097853951
a=3
b=41058363725152142129326129780047268409114441015993725554835256314039467401291
h=1
Order:115792089210356248762697446949407573529996955224135760342422259061068512044369
Gx=48439561293906451759052585252797914202762949526041747995844080717082404635286
Gy=36134250956749795798585127919587881956611106672985015071877198253568414405109
Message 1: Hello **Message 2:** Goodbye
Sig1(R,S):6843699916116213566632831575027916680919552821044868158888556087460606638197164497954281106906978594318856699527613005888505797620296329843674872597395472
Sig 2(R,S):59396660104252040522208448410403058790061382347675159895630126540572795945686824261114452357845949316447297212726196377981818640200148557183227464803947630
Random value (k1): 544079690520661127105791673855 32488796 **Random value (k2):** 1394947286662891185439150023375 93135844
Private Key: 48588618394399226405893001917337148111899544979674835399706352006027182977592
The private key is found: 48588618394399226405893001917337148111899544979674835399706352006027182977592

of attack; the failure rate can be decreased if we perform the same attack with more and more signatures. Table 4 – shows ECDSA: Disclosing the private key if nonce is known on NIST-256P recommended parameters using LENSTRA–LENSTRA–LOVASZ (LLL) method.

3.3 ECDSA – Disclosing the Private Key Using Lenstra–Lenstra–Lovasz (LLL) Method, If Nonce Known with Real-world ECDSA Bugs

A recent real-time problem affects Yubi keys' randomness generation, where poor randomness causes the same value to be fixed to almost 80 bits of nonce. These real-world issues are much easier to attack than the ones used in the preceding section. While in section A we are unsure of the fixed 80-bit values, we are aware that the fixed 128-bit values were all set to zeros. In this method, we assume that every collection of received signatures has a nonce with an exact length of 80 bits. Additionally, we believe that the 80 fixed bits are the most important bits. (Even if they are not the most significant bits, the attack may still be carried out by simply performing a left shift on one bit at a time, which is equivalent to multiplying the signature by two.) The 80 most significant bits of the differences between any two nonces in this case will all be zeros because we don't know what these 80 bits are. The same lattice attack as described in section B is used, with the exception that our signature values are subtracted. We will construct the matrix below using a set of n signatures and messages, which will then be used as input by the LLL algorithm to produce a new output matrix. The variance between the nonces for signatures 1 and n is the LLL algorithm's output matrix, which is designated as k_1–k_n. Instead of having an entire row filled with nonces, we really have a row with the difference between each nonce and the n^{th} nonce in this case since we distinguished the n^{th} value from each element in the matrix.

$$\begin{pmatrix} [N] & 0 & \dots & 0 & 0 & 0 \\ 0 & [N] & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & [N] & 0 & 0 \\ [r_1 s_1^{-1} - r_n s_n^{-1}] & [r_2 s_2^{-1} - r_n s_n^{-1}] & \dots & [r_{n-1} s_{n-1}^{-1} - r_n s_n^{-1}] & [B/N] & 0 \\ [m_1 s_1^{-1} - m_n s_n^{-1}] & [m_2 s_2^{-1} - m_n s_n^{-1}] & \dots & [m_{n-1} s_{n-1}^{-1} - m_n s_n^{-1}] & 0 & [B] \end{pmatrix}$$

LLL Input Matrix with unknown Nonce Bias

If generated signatures are made from nonces with 80 fixed bits, the secret key can be easily extracted from only five signatures. We constructed the aforementioned matrix with $n = 6$ to lower the mistake rate. The 80 fixed bits used for generation are scarce in the real world. When used with 256 bit elliptic curves, this type of attack is far more resilient and still effective even when 4 bits of the nonce are fixed. The implementation does not become difficult; rather, the attacker merely needs to increase the size of the lattice, or the value of n, and repeat the attack. This method will lengthen the algorithm's execution time without increasing its complexity. The value of N in our experiments represents the total number of signatures needed to recover the secret key, and it was calculated experimentally by trying to launch an attack using a different number of signatures on a different number of fixed bits. The value of N = 2 when the nonce's first 128 bits were fixed to 0, and the value of N = 3 when the first 128 bits are fixed but we are unsure of their fixed values. N = 5 when the nonce had 80 fixed bits chosen at random.

One can recover the secret key using below formulations:

(1) $Sig_1 = k_1^{-1}(Msg_1 + xr_1)$ and $Sig_n = k_n^{-1}(Msg_n + xr_n)$

(2) $Sig_1 k_1 = Msg_1 + xr_1$ and $Sig_n k_n = Msg_n + xr_n$

(3) $k_1 = Sig_1^{-1}(Msg_1 + xr_1)$ and $k_n = Sig_n^{-1}(Msg_n + xr_n)$

(4) $k_1 - k_n = Sig_1^{-1}(Msg_1 + xr_1) - Sig_n^{-1}(Msg_n + xr_n)$

(5) $Sig_1 Sig_n(k_1 - k_n) = Sig_n(Msg_1 + xr_1) - Sig_1(Msg_n + xr_n)$

(6) $Sig_1 Sig_n(k_1 - k_n) = xSig_n r_1 - xSig_1 r_n + Sig_n Msg_1 - Sig_1 Msg_n$

(7) $x(Sig_1 r_n - Sig_n r_1) = Sig_n Msg_1 - Sig_1 Msg_n - Sig_1 Sig_n(k_1 - k_n)$

(8) Secret key $x = (r_n Sig_1 - r_1 Sig_n)^{-1}(Sig_n Msg_1 - Sig_1 Msg_1 Sig_1 Sig_n (k_1 - k_n))$

4 Performance Analysis

In this section, the experimental analysis of running time of algorithm to crack ECDSA using selected NIST and SECP curves are presented. Table 6 shows time to crack ECDSA algorithm with leak of nonce and ECDSA with LLL algorithm. Each algorithm is executed on five different intervals of time with different curves and average execution times to crack the algorithm are recorded. Among all the three curves NIST256p require less time to crack and ECDSA with LLL among SECP256k1 and NIST256p, SECP256k1 require

Table 5 Features of nodes used in our research

Type of Node	Processor	CPU Type	CPU Speed	RAM	Operating System
HP LAPTOP	Intel Core i3	64 bits	1.99 GHz	4GB	Windows 10
Raspberry pi	ARM CPU	64 bits	1.2 GHz	1GB	Rasbian 5.10

Table 6 Elliptic curves average execution time in seconds to crack ECDSA

	Average Execution Time (Seconds)					
ECDSA With Curves	T1	T2	T3	T4	T5	Avg
NIST256p	0.004	0.005	0.060	0.004	0.007	0.016
NIST521p	0.020	0.033	0.017	0.024	0.076	0.034
SECP256k1	0.060	0.016	0.017	0.073	0.068	0.046
LLL with SECP256k1	2.80	3.00	3.17	2.68	2.98	2.926
LLL with NIST256p	3.20	3.64	3.13	3.70	3.39	3.412

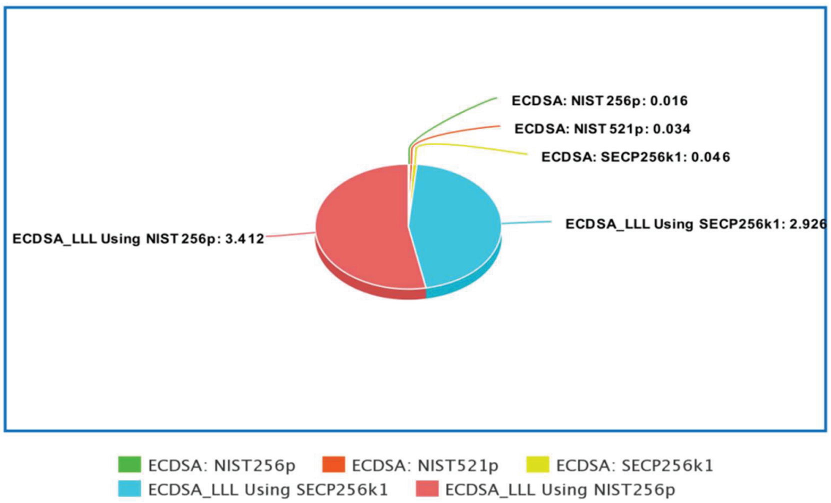

Figure 6 Average execution time to crack ECDSA (seconds).

less time to crack. Figure 6 demonstrates average execution time to crack ECDSA.

5 Conclusions

In this paper, curves recommended by various standards are selected and examined. Each curve applied on ECDSA algorithm is cracked in two ways if nonce is leaked and another way is by performing lattice attacks using Lenstra-Lenstra-Lovasz (LLL) algorithm if random number generator generates bad nonce. The comparative table shows the computation time taken by each curve when these two algorithms are used. From this analysis it is clear the computation times of curves increases when field size increases. Therefore ECDSA is fragile and we recommend use of EdDSA where nonce's are generated safely without use of RNG. Further NIST has standardized use of EdDSA with Curve25519 to overcome side channel attacks. Use of ECDSA should be done with caution such as nonce used for ECDSA signatures are never repeated, never revealed (even partially), and generated safely. Finally we come to a conclusion that elliptic curve cryptography using the NIST256p, SECP256k1, NIST521p curves and weak nonce are not safe for the transactions that are confidential and are to be kept secured down the line.

Appendix

LLL is used on the basis of (201, 37) and (1648, 297). We choose one of these as our initial basis vector before reducing the second vector to a candidate basis vector.

LLL Example: Applying LLL to the basis spanned by (201, 37) and (1648, 297). We begin by choosing one of these as our first basis vector, then using it to reduce the second vector to a candidate basis vector.

Step 1: Let us consider our first lattice basis vector $\vec{v}_1$ as first Gram-Schmidt vector $\vec{v}_1^*$

$$\vec{v}_1 = (201, 37).\ \vec{v}_2 = (1648, 297) \text{ and } \vec{v}_1^* = (201, 37)$$

Applying Gram-Schmidt reduction to reduce vector $\vec{v}_2$:

$$\vec{v}_2 = (1648, 297) - \frac{(1648, 297) \cdot (201, 37)}{(201, 37) \cdot (201, 37)}(201, 37) \approx (1.133, -6.155)$$

We have

$\vec{v}_1 = (201, 37)$, $\vec{v}_2 = (1648, 297)$,
$\vec{v}_1^* = (201, 37)$ and $\vec{v}_2^* = (1.133, -6.155)$

We have: $\vec{v}_1 = (40, 1)$, $\vec{v}_2 = (201, 37)$, $\vec{v}_1^* = (40, 1)$ and $\vec{v}_2^* = (-0.799, 31.956)$.

Using $\vec{v}_1$ to reduce $\vec{v}_2$

$$\vec{v}_2 = (201, 37) - \left\lfloor \frac{(201, 37) \cdot (40, 1)}{(40, 1) \cdot (401, 1)} \right\rceil (40, 1) = (1, 32)$$

We have: $\vec{v}_1 = (40, 1)$, $\vec{v}_2 = (1, 32)$, $\vec{v}_1^* = (40, 1)$ and $\vec{v}_2^* = (-0.799, 31.956)$.

Next, we find the magnitude of Gram-Schmidt basis vector $\|\vec{v}_1^{*2}\|$ and $\|\vec{v}_2^{*2}\|$ and check the Lavasz condition. $\|\vec{v}_1^{*2}\| = 1601$, $\|\vec{v}_2^{*2}\| = 1021.76$

$$\mu_{2,1} = \left(\frac{(1,32)\cdot(40,1)}{(40,1)\cdot(40,1)} = 0.193\right)$$

$$\left(\frac{3}{4} - \mu_{2,1}^2 \approx 0.748\right)$$

So, $\|\vec{v}_2^*\|^2 \ngeq (\frac{3}{4} - \mu_{2,1}^2)\|\vec{v}_1^*\|^2$ and we should swap, making $\vec{v}_1 = (1,32)$ and $\vec{v}_2 = (40,1)$.

Step 3:

We have:

$$\vec{v}_1 = (1,32).\ \vec{v}_2 = (40,1) \text{ and } \vec{v}_1^* = (1,32).$$

Now apply the Gram-Schmidt reduction, using $\vec{v}_1^* = \vec{v}_1$

$$\vec{v}_2 = (40,1) - \frac{(40,1)\cdot(1,32)}{(1,32)\cdot(1,32)}(1,32) \approx (39.93,-1.25)$$

We have:

$$\vec{v}_1 = (1,32),\ \vec{v}_2 = (40,1),\ \vec{v}_1^* = (1,32) \text{ and } \vec{v}_2^* = (39.93,-1.25).$$

Using $\vec{v}_1$ to reduce $\vec{v}_2$

$$\vec{v}_2 = (40,1) - \left\lfloor \frac{(40,1)\cdot(1,32)}{(1,32)\cdot(1,32)} \right\rceil (1,32)$$

$$\vec{v}_2 = (40,1) - 0(1,32)$$
$$\vec{v}_2 = (40,1)$$

Next, we find the magnitude of Gram-Schmidt basis vector $\|\vec{v}_1^{*2}\|$ and $\|\vec{v}_2^{*2}\|$ and check the Lavasz condition. $\|\vec{v}_1^{*2}\| = 1025$, $\|\vec{v}_2^{*2}\| = 1595.94$

$$\mu_{2,1} = \left(\frac{(40,1)\cdot(1,32)}{(1,32)\cdot(1,32)} = 0.070\right)$$

$$\left(\frac{3}{4} - \mu_{2,1}^2 \approx 0.745\right)$$

So, $\|\vec{v}_2^*\|^2 \geq (\frac{3}{4} - \mu_{2,1}^2)\|\vec{v}_1^*\|^2$ and we can move on to the next basis vector. $\vec{v}_1 = (1,32)$ and $\vec{v}_2 = (40,1)$ correspond to reasonably orthogonal set of basis vectors.

Acknowledgement

The authors would like to acknowledge the support provided by Presidency University – Bengaluru, India.

References

[1] Chintan Patel, Nishant Doshi, "Secure Light Weight Key Exchange Using ECC For User Gateway Paradigm" 2021 IEEE Transactions on Computer DOI: 10.1109/TC.2020.3026027 Access Pages: 1–1.

[2] Dimitrios Poulakis "New lattice attacks on DSA schemes Journal of Mathematical Cryptology 2016 IEEE Open Access Pages: 70025–70034 DOI: 10.1515/jmc-2014-0027 Volume 10 Issue 2".

[3] Badis Hammi, Achraf Fayad, Rida Khatoun, Sherali Zeadally and Youcef Begriche 2020 "A Lightweight ECC-Based Authentication Scheme for Internet of Things (IoT)" February 2020 IEEE Systems Journal 2020 Pages: 3440–3450 DOI: 10.1109/JSYST.2020.2970167 Volume: 14.

[4] Xiaoqiang Zhang And Xuesong Wang "Digital Image Encryption Algorithm Based on Elliptic Curve Public Cryptosystem IEEE November 2018 Access Pages: 70025–70034 ISSN: 2169-3536 Volume: 6".

[5] Mohammad Ayoub Khan, Mohammed Tabrez Quasim, Norah Saleh Alghamdi, Mohammad Yahiya Khan "A Secure Framework for Authentication and Encryption Using Improved ECC for IoT-Based Medical Sensor Data" IEEE Access Pages: 52018–52027 ISSN: 2169-3536 Volume: 8.

[6] Debiao He and Sherali Zeadally, "An Analysis of RFID Authentication Schemes for Internet of Things in Healthcare Environment Using Elliptic Curve Cryptography", September 2014 IEEE internet of things journal, Electronic ISSN: 2327-4662 DOI: 10.1109/JIOT.2014.2360121 vol. 2, no. 1.

[7] Sahil Garg, Kuljeet Kaur, Georges Kaddoum, and Kim-Kwang Raymond Choo, "Toward Secure and Provable Authentication for Internet of Things: Realizing Industry 4.0", September 2019 IEEE internet of things journal, Electronic ISSN: 2327-4662 DOI: 10.1109/JIOT.2019.2942271.

[8] Maxim Chernyshev, Zubair Baig, Oladayo Bello, and Sherali Zeadally "Internet of Things (IoT): Research, Simulators, and Testbeds", December 2017 IEEE Internet of Things Journal, Vol. 5, No. 3, June 2018 Electronic ISSN: 2327-4662 DOI: 10.1109/JIOT.2017.2786639.

[9] Xiaoqiang Zhang And Xuesong Wang "Digital Image Encryption Algorithm Based on Elliptic Curve Public Cryptosystem" 09 November 2018 IEEE Open access Electronic ISSN: 2169-3536 DOI: 10.1109/ACCESS .2018.2879844.

[10] Anum Sajjad, Mehreen Afzal, Mian Muhammad Waseem Iqbal, Haider Abbas, Rabia Latif, and Rana Aamir Raza "Kleptographic Attack on Elliptic Curve Based Cryptographic Protocols" 29 July 2020 IEEE Open access Electronic ISSN: 2169-3536 DOI: 10.1109/ACCESS.2 020.3012823.

[11] Patrick Longa; Ali Miri "Fast and Flexible Elliptic Curve Point Arithmetic over Prime Fields" IEEE Transactions on Computers, Volume: 57, Issue: 3, March 2008 Print ISSN: 0018-9340 DOI: 10.1109/ TC.2007.70815.

[12] P. K. Gupta, B. T. Maharaj, and R. Malekian, "A novel and secure IoT based cloud centric architecture to perform predictive analysis of user's activities in sustainable health centres," Multimedia Tools Appl., vol. 76, no. 18, pp. 18489–18512, Sep. 2017 doi.org/10.1007/s11042-0 16-4050-6.

[13] G. Rathee, A. Sharma, H. Saini, R. Kumar, and R. Iqbal, "A hybrid framework for multimedia data processing in IoT-healthcare using blockchain technology," Multimedia Tools Appl., 2019. Electronic ISSN 1573-7721 https://doi.org/10.1007/s11042-019-07835-3.

[14] A. H. El Zouka and M. M. Hosni, "Secure IoT communications for smart healthcare monitoring system," in Internet of Things. Amsterdam, The Netherlands: Elsevier, 2019. doi.org/10.1016/j.iot.2019.01.003.

[15] Kendall Ananyi, Hamad Alrimeih, and Daler Rakhmatov "Flexible Hardware Processor for Elliptic Curve Cryptography Over NIST Prime Fields" IEEE Transactions on Very Large Scale Integration (VLSI) Systems Volume: 17, Issue: 8, August 2009 Print ISSN: 1063-8210 DOI: 10.1109/TVLSI.2009.2019415.

[16] Bijan Ansari and M. Anwar HasanHigh-Performance Architecture of Elliptic Curve Scalar Multiplication IEEE Transactions on Computers (Volume: 57, Issue: 11, November 2008, DOI: 10.1109/TC.2008.133 Print ISSN: 0018-9340.

[17] Nizar Ouni and Ridha Bouallegue "Performance And Complexity Analysis of Reduced Iterations LLL Algorithm" International Journal of Computer Networks Communications (IJCNC) May 2016 Vol. 8".

[18] Yunju Park and Jaehyen “Analysis of the upper bound on the complexity of LLL Algorithm”, Journal of the Korean Society for Industrial and Applied Mathematics 2016 Vol. 20, No. 2, 107–121.

[19] Michael Brengel and Christian Rossow “Identifying Key Leakage of Bitcoin Users International Symposium on Research in Attacks, Intrusions, and Defenses 2018 Open Access LNSC”, ISBN: 978-3-030-00470-5 volume 11050.

[20] Dan Boneh Ramarathnam Venkatesan “Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes” Lecture Notes in Computer Science- Annual International Cryptology Conference 2001, volume 1109, pp. 129–142.

[21] Joachim Breitner and Nadia Heninger “Biased Nonce Sense: Lattice Attacks against Weak ECDSA Signatures in Cryptocurrencies” Lecture Notes in Computer Science 2019 Springer International Publishing – Financial Cryptography and Data Security.

[22] Jack Doerner, Yashvanth Kondi, Eysa Lee and abhi shelat “Threshold ECDSA from ECDSA Assumptions:The Multiparty Case” May 2019 IEEE Symposium on Security and Privacy DOI: 10.1109/SP37 863.2019.

[23] S. Tyagi, A. Agarwal, and P. Maheshwari, “A conceptual framework for IoT-based healthcare system using cloud computing,” in Proc. 6th Int. Conf.-Cloud Syst. Big Data Eng. (Conuence), Noida, India, Jan. 2016, pp. 503–507. Electronic ISBN: 978-1-4673-8203-8 DOI: 10.1109/CO NFLUENCE.2016.7508172.

[24] M. Wen, J. Lei, J. Li, Y. Wang, and K. Chen, “Efficient user access control mechanism for wireless multimedia sensor networks,” J. Comput. Inf. Syst., vol. 7, no. 9, pp. 3325–3332, 2011.

[25] Javed R. Shaikh, Maria Nenova, Georgi Iliev and Zlatka Valkova-Jarvis “Analysis of Standard Elliptic Curves for the Implementation of Elliptic Curve Cryptography in Resource-Constrained E-commerce Applications” 2017 IEEE-COMCAS ISBN: 978-1-5386-3169-0.

[26] Shen Guicheng, Yu Zhen “Application of Elliptic Curve Cryptography in Node Authentication of Internet of Things IEEE-IIHMSP” ISBN: 978-0-7695-5120-3 DOI: 10.1109/IIH-MSP.2013.118.

[27] Ravi Kishore Kodali and Ashwitha Naikoti “ECDH based Security Model for IoT using ESP 8266” 2016 IEEE- ICCICCT DOI: 10.110 9/ICCICCT.2016.7988026.

[28] Deepak S. Sakkari Mohammed Mujeer Ulla “Review on Insight into Elliptic Curve Cryptography” 2022 Modern Approaches in Machine

Learning Cognitive Science: A Walkthrough DOI: 10.1007/978-3-030-96634-88.

[29] Deepak S. Sakkari Mohammed Mujeer Ulla "Design and Implementation of Identifying Points on Elliptic Curve Efficiently Using Java" 2022 Modern Approaches in Machine Learning Cognitive Science: A Walkthrough DOI: 10.1007/978-3-030-96634-88.

[30] Deepak S. Sakkari Mohammed Mujeer ulla "Design and Implementation of Elliptic Curve Digital Signature Using Bit Coin Curves Secp256K1 and Secp384R1 for Base10 and Base16 Using Java" 2022 Innovation in Electrical Power Engineering, Communication, and Computing Technology DOI: 10.1007/978-981-16-7076-328.

[31] Nissa Mehibel, M'hamed Hamadouche "A new approach of elliptic curve Diffie-Hellman key exchange" 2017 5th International Conference on Electrical Engineering - Boumerdes (ICEE-B) Electronic ISBN: 978-1-5386-0686-5 DOI: 10.1109/ICEE-B.2017.8192159.

[32] Amit Dua, Akash Dutta, "A Study of Applications Based on Elliptic Curve Cryptography", Proceedings of the Third International Conference on Trends in Electronics and Informatics (ICOEI 2019) IEEE Xplore Electronic ISBN: 978-1-5386-9439-8 DOI: 10.1109/ICOEI.2019.8862708.

[33] Leonidas Deligiannidis, "Elliptic curve cryptography in Java" 2015 IEEE International Conference on Intelligence and Security Informatics (ISI) Electronic ISBN: 978-1-4799-9889-0 DOI: 10.1109/ISI.2015.7165975.

Biographies

Mohammed Mujeer Ulla, currently working as Assistant Professor in Department of computer science and engineering since 2017 and is pursuing his Ph.D. from presidency University. He is an alumni of R.V college of engineering – Bangalore in his UG and PG. He has many papers to his credit in reputed international and national conferences journals. His areas of expertise include internet of Things, Wireless sensor network.

Deepak. S. Sakkari, currently working as Assistant Professor in the Department of Computer Science and Engineering, Presidency University, Bangalore. He received his B.E in Instrumentation and Electronics from Siddganga Institute of Technology, Bangalore University, M.Tech in Information Technology from AAIDU, Allahabad and PhD in Computer Science Engineering from JNTUH, Hyderabad. He published many paper in Scopus indexed and SCI journals with Google scholar 9 citations. His research area includes Wireless Sensor Networks.