Implementation of Elliptic Curve Cryptosystem with Bitcoin Curves on SECP256k1, NIST256p, NIST521p, and LLL

Mohammed Mujeer Ulla[1,*], Preethi[2], Md. Sameeruddin Khan[1] and Deepak S. Sakkari[3]

[1]*School of Computer Science and Engineering, Presidency University, Bangalore, Karnataka, India*
[2]*Department of Information Technology, Manipal Institute of Technology, Bengaluru, Manipal Academy of Higher Education, Manipal, India*
[3]*Department of Computer Science and Engineering, Sri Krishna Institute of Technology, Bangalore*
E-mail: mohammedmujeerulla@presidencyuniversity.in; preethi.srivathsa@manipal.edu; sameerirfan70@gmail.com; deepakssakkari@presidencyuniversity.in
[*]*Corresponding Author*

Received 23 April 2023; Accepted 27 August 2023; Publication 18 November 2023

Abstract

Very recent attacks like ladder leaks demonstrated the feasibility of recovering private keys 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. 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 an ECDSA signature to sign a message using the 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 a

Journal of ICT Standardization, Vol. 11_4, 329–354.
doi: 10.13052/jicts2245-800X.1141

way in which the intruder can discover the private key if the signer leaks any one of the nonce values. Then we use Lenstra–Lenstra–Lovasz (LLL) method as a black box, we will try to attack signatures generated from bad nonce or bad random number generator (RAG) on NIST256p, SECP256k1 curves. The combination of nonce generation, post-message signing, and validation in ECDSA helps achieve Uniqueness, Authentication, Integrity, and Non-Repudiation. The analysis is performed by considering all three curves for the implementation of the Elliptic Curve Digital Signature Algorithm (ECDSA). The comparative analysis for each of the selected curves in terms of computational time is done with the leak of nonce and with the 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 zero depicting the strength of the algorithm. The average computational costs to break ECDSA with curves SECP256K1 and NIST256p using LLL are 2.9 and 3.4 respectively

Keywords: Internet of Things, ECC – elliptic curve cryptography, SEC – U.S. securities and exchange commission, IEEE – institute of electrical and electronics engineers, ISO – international organization for standardization, American national standards institute, The NIST national institute of standards and technology, American security agency, EdDSA – edwards curve digital signature algorithm nonce – number only used once, RAG – random number generator.

1 Introduction

Over recent years huge amounts of sensitive data exchanged in applications like direct online banking (or third-party applications such as Google Pay, and Paytm), stock market trading, and remote access to data in health care, defense sector, automotive sector, retail sector, and many more areas are too high due to drastic changes in the technology. Many internet security protocols rely on public-key cryptosystems to attain confidentiality, integrity, and authentication. A widely adopted public-key protocol over the internet is the Elliptic Curve Digital Signature Algorithm (ECDSA). Some of the application areas of ECDSA are TLS, Open PGP, and smart cards, which can be found in Ripple, Ethereum, and Bitcoin. Due to hardness in discrete logarithm problems, it is highly secure and due to its small key size, it is a fast signing algorithm. Due to these features, it has been recommended by IEEE and NIST since 2000, ANSI since 1999, and ISO since 1998 [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. An 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 the 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 a theoretical principal 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 Elliptic Curve Digital Signature (ECDSA)

The Elliptic curve digital signature or simply ECDSA is a public key cryptography encryption algorithm. The keys generated via ECDSA are exponentially smaller in size than keys generated by any other digital signing algorithm. For example, to have 128-bit security using RSA requires 3072 bit key size while ECC requires 256 key size. To have a 256-bit security using RSA requires a 15360-bit key size while ECC requires a 512 key size.

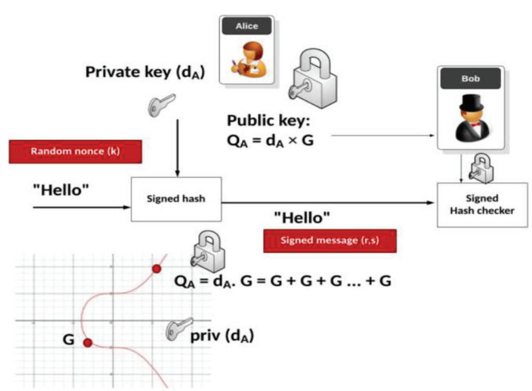

Figure 1 ECDSA.

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
(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 to 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

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

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

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

2.2 The LLL Algorithm

An efficient way to find reasonably orthogonal basis is the LLL algorithm, named after its inventors: Lenstra, Lenstra and Lovasz. Conceptually LLL algorithm consists of two parts:

- Reducing a non-basis vector (working vector) by subtracting multiples of the current basis vectors
- Deciding whether the working vector becomes the next basis vector or whether it should replace the basis vector immediately before it.

The Lovász condition is fulfilled if the vectors are close enough to being orthogonal, or if they are roughly ordered by length.

Lovasz condition obtained by rearranging orthogonal vectors: $(\delta - \mu^2_{\mathrm{i+1,i}})$ $\|\mathrm{b}^*_{\mathrm{i}}\|^2 = \|\mathrm{b}^*_{\mathrm{i+1}}\|^2$.

This decision is based on whether or not Lovasz condition is met. Roughly speaking Lovasz condition determines the working vector is big enough to be the next basis vector [4]. We keep track of two sets of vectors:

- $\vec{\mathrm{v}}_1$; ..., the current set of basis vectors which we are trying to reduce to a nearly orthogonal set.
- $\vec{\mathrm{v}}^*_1$, $\vec{\mathrm{v}}^*_2$, ..., the set of orthogonal basis vector produced by the Gram-Schmidt reduction.

We also n to keep track of k, this is a number of the working basis vectors we are trying to reduce. Suppose our basis vectors are $\vec{\mathrm{v}}^*_1$, $\vec{\mathrm{v}}^*_2$,... $\vec{\mathrm{v}}^*_{\mathrm{k-1}}$, $\vec{\mathrm{v}}^*_{\mathrm{k}}$,... and suppose we are working to reduce $\vec{\mathrm{v}}_{\mathrm{k}}$. We reduce by expected fashion by subtracting multiples of $\vec{\mathrm{v}}_1, \vec{\mathrm{v}}_2$,... $\vec{\mathrm{v}}_{\mathrm{k-1}}$. Now let us consider the vectors $\vec{\mathrm{v}}_{\mathrm{k-1}}$ and $\vec{\mathrm{v}}_{\mathrm{k}}$, if these were only two vectors we had we might need to subtract some multiple of new $\vec{\mathrm{v}}_{\mathrm{k}}$ from the old $\vec{\mathrm{v}}_{\mathrm{k-1}}$. This requires swapping $\vec{\mathrm{v}}_{\mathrm{k-1}}$ and $\vec{\mathrm{v}}_{\mathrm{k}}$. But since we have a new $\mathrm{k}-1$ vector we need to go through the whole process again, this time with $\vec{\mathrm{v}}_{\mathrm{k-1}}$ with as new working vector. The decision of whether to swap $\vec{\mathrm{v}}_{\mathrm{k-1}}$ and $\vec{\mathrm{v}}_{\mathrm{k}}$ and make $\vec{\mathrm{v}}_{\mathrm{k-1}}$ the working vector is based on whether the Lovasz condition is satisfied. In addition to basis vectors $\vec{\mathrm{v}}^*_1$, $\vec{\mathrm{v}}^*_2$, $\vec{\mathrm{v}}^*_3$,... found from the Gram-Schmidt reduction. Let $\vec{\mathrm{v}}_{\mathrm{k}}$ 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 $||\vec{\mathrm{v}}^*_{\mathrm{k}}||^2 = (\frac{3}{4} - \mu^2_{\mathrm{k,k-1}})$ then we are done with $\vec{\mathrm{v}}_{\mathrm{k}}$ for now and can make $\vec{\mathrm{v}}^*_{\mathrm{k+1}}$ the next working vector, otherwise, swap $\vec{\mathrm{v}}_{\mathrm{k-1}}$ and $\vec{\mathrm{v}}_{\mathrm{k}}$ and make $\vec{\mathrm{v}}_{\mathrm{k-1}}$ the working vector [5]. 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 $\vec{\mathrm{v}}_1$, as first Gram-Schmidt vector $\vec{\mathrm{v}}^*_1$

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

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

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

$$\approx (1.133, -6.155)$$

We have:

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

Now we use $\vec{v}_1$ to reduce $\vec{v}_2$:

$$\vec{v}_2 = (1648, 297) - \frac{(1648, 297) \cdot (201, 37)}{(201, 37) \cdot (201, 37)}(201, 37)$$
$$\vec{v}_2 = (1648, 297) - 8(201, 37)$$
$$\vec{v}_2 = (40, 1)$$

We have

$$\vec{v}_1 = (201, 37) \text{ and } \vec{v}_2 = (40, 1)$$
$$\vec{v}_1^* = (201, 37) \text{ and } \vec{v}_2^* = (1.133, -6.155)$$

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

$$||\vec{v}_1^*||^2 = 41770||\vec{v}_2^*||^2 = 39.16$$
$$\mu_{2,1} = \frac{(40, 1) \cdot (201, 37)}{(201, 37) \cdot (201, 37)} = 0.193$$
$$\frac{3}{4} - \mu_{2,1}^2 \approx 0.713$$

So, $||\vec{v}_2^*||2! \geq (\frac{3}{4} - \mu_{2,1}^2)||\vec{v}_1^*||2$ and we should d swap, making $\vec{v}_1 = (40, 1)$ and $\vec{v}_2 = (201, 37)$

Step 2: We have $\vec{v}_1 = (40, 1)$ and $\vec{v}_2 = (201, 37)$ and $\vec{v}_1^* = (40, 1)$. Now apply the Gram-Schmidt reduction, using $\vec{v}_1^* = \vec{v}_1$

$$\vec{v}_2 = (201, 37) - \frac{(201, 37) \cdot (40, 1)}{(40, 1) \cdot (40, 1)}(40, 1) \approx (-0.799, 31.95)$$

We have $\vec{v}_1 = (40, 1)$ and $\vec{v}_2 = (201, 37)$ and $\vec{v}_1^* = (40, 1)$ 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 $\vec{v}_1 = (40, 1)$ and $\vec{v}_2 = (1, 32)\vec{v}_1^* = (40, 1)$ 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

$$||\vec{v}_1^*||^2 = 1601||\vec{v}_2^*||^2 = 1021.7$$

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

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

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

Step 3: We have $\vec{v}_1 = (1, 32)$, $\vec{v}_2 = (40, 1)$ and $\vec{v}_1^* = (1, 3)$
Now apply the Gram-Schmidt reduction, using $\vec{v}_1^* = \vec{v}_1$

$$\vec{v}_2 = (40, 1) - \frac{(40, 1) \cdot (1, 32)}{(1, 32) \cdot (1, 3)}(1, 32) \approx (39.93, -1.25)$$

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 $\vec{v}_1 1$ to reduce $\vec{v}_2$

$$\vec{v}_2 = (40, 1) - \frac{(40, 1) \cdot (1, 3)}{(1, 32) \cdot (1, 3)}(1, 32) \approx (39.93, -1.25)$$

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

$$||\vec{v}_1^*||^2 = 1025 \text{ and } ||\vec{v}_2^*||^2 = 1595.94$$

$$\mu_{2,1} = \frac{(40,1)\cdot(1,3)}{(1,32)\cdot(1,3)} = 0.070$$

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

So, $||\vec{v}_2^*||^2 = (\frac{3}{4} - \mu_{2,1}^2)||\vec{v}_2^*||^2$ 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.

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 [6]. 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. 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 computes her 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.

Steps to disclose the ECDSA private key, if nonce is known using NIST256p:

(1) As the first step Alice generates private key
(2) In the next step Alice generates public key = private key*G
(3) Alice has message (M) to be sent to Bob
(4) Alice generates a nonce value or random value (k) and then she finds values of r and s needed for ECDSA r = k * G and s = k^{-1} (H(M) + r $*$ private key)
(5) The signature for generated message is (r,s)

Let us assume that Alice has leaked her nonce value k to Bob, and the steps followed used by Eve to recover private key of Alice if he knows (r,s) and k is as follows:

(1) Bob has received r = k * G
(2) Bob has received

$$\mathrm{s} = \mathrm{k}^{-1}(\mathrm{H}(\mathrm{M}) + \mathrm{r} * \text{private key}) \tag{3}$$

(3) Using Equation (3) we get

$$\mathrm{s} * \mathrm{k} = \mathrm{H}(\mathrm{M}) + \mathrm{r} * \text{privatekey} \tag{4}$$

(4) Using Equation (4) we get

$$\mathrm{r}.\text{privatekey} = \mathrm{s} * \mathrm{k} - \mathrm{H}(\mathrm{M}) \tag{5}$$

(5) Using Equation (5) we get

$$\text{Private key} = \mathrm{r}^{-1}(\mathrm{s} * \mathrm{k} - \mathrm{H}(\mathrm{M}))\mathrm{MOD\ N}) \tag{6}$$

Table 1 – shows ECDSA: Disclosing the private key if nonce is known using NIST-256P recommended parameters. Table 2 shows ECDSA: Disclosing the private key if nonce is known using SEC-256K1recommended parameters. Table 3 shows ECDSA: disclosing the private key if nonce is known using NIST-521P recommended parameters.

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 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

Table 1 ECDSA: Disclosing the private key, if nonce known (NIST-256P recommended parameters)

N=115792089210356248762697446949407573530086143415290314195533631308867097853951
a=-3
b=41058363725152142129326129780047268409114441015993725554835256314039467401291
h=1
Order:115792089210356248762697446949407573529996955224135760342422259061068512044369
G_x=48439561293906451759052585252797914202762949526041747995844080717082404635286
G_y=36134250956749795798585127919587881956611106672985015071877198253568414405109
Message 1: Hello
Sig1(R,S):878646081725150763247877540023260603423548920159641694186012633738974915481139179691545342485220128989572366897037545134234159208548398831975568810267476
PrivateKey:18268791631015765630459340399299329549626419067395157743556468023750731901655
Theprivatekeyisfound:18268791631015765630459340399299329549626419067395157743556468023750731901655

Table 2 ECDSA: Disclosing the private key, if nonce known (SEC-256K1 recommended parameters)

N=115792089237316195423570985008687907853269984665640564039457584007908834671663
a=0
b=7
h=1
Order:115792089237316195423570985008687907852837564279074904382605163141518161494337
G_x=55066263022277343669578718895168534326250603453777594175500187360389116729240
G_y=32670510020758816978083085130507043184471273380659243275938904335757337482424
$\mathbf{Message_1}$: Hello
$\mathbf{Sig_1}$(R,S):7773499647157869072481902530128808492319852184749699500415381153024496572628597413358969903295688400058661785083988603187257053615822443330142691718421644
Random value (k):63076811158092363886617914846091290891
PrivateKey:16122978565960941408252013174486341707774481479068509615634681955105988655192
Theprivatekeyisfound:16122978565960941408252013174486341707774481479068509615634681955105988655192

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

N=6864797660130609714981900799081393217269435300143305409394463459185543 183397656052122559640661454554977296311391480858037121987999716643812574028291115057151
a=-3
b=1093849038073734274511112390766805569936207598951683748994586394495953 116150735016013708737573759623248592132296706313309438452531591012912142327488478985984
h=1
Order:686479766013060971498190079908139321726943530014330540939446345918554318339765539424505774633321719753296399637136332111386476861244038034 0372808892707005449
Gx=2661740802050217063228768716723360960729859168756973147706671368418802944996427808491545080627771902352094241225065558662157113545570916814161637315895999846
Gy=3757180025770020463545507224491183603594455134769762486694567779615544477440556316691234405012945595621444445372894285225856667291965808101243442775783767841
Message 1: Hello
$\mathbf{Sig_1}$(R,S):11891240798780373059492782119630253015593463417072630918695343227108169406794340204065192771137257672924269931870943645720195142835026590909326228526323742666026149665276099943145817528547307557950641078431104888583170051353393866925493327870853938636478018167162202874924973940795949272348183625732014938948579325
Random value (k):1345073822754761250886379837 21177218254
PrivateKey:52306603676855575123284881219115986204474345398586651793401948786865461869276086442795042889752346555662781627772177764035955238771 558726538211432930531403 44
Theprivatekeyisfound:523066036768555751232848812191159862044743453985866519340194878686546186927608644279504288975234655566278162777217776403595 5238771558726538211432930531403 44

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 [7]. 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 [9]. 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 [10]. 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.

A nonce is considered a secret value, and its leakage can potentially compromise the security of the system. If an attacker gains knowledge of the nonce, they might be able to launch various attacks depending on the specific cryptographic protocol in use. Here are a few possible ways a nonce can be leaked:

1. Side-Channel Attacks: Attackers can exploit side-channel information, such as timing or power consumption, to infer the nonce value. For instance, variations in execution times during cryptographic operations might provide clues about the nonce.
2. Fault Attacks: In some cases, attackers may intentionally introduce faults into a cryptographic operation to cause errors that leak information about the nonce when analysed.
3. Weak Random Number Generation: If the nonce is generated using a weak random number generator, it might be possible for an attacker to predict or guess the nonce value.
4. Software Vulnerabilities: Vulnerabilities in the software or implementation of cryptographic algorithms can sometimes expose the nonce unintentionally. For example, buffer overflows or memory leaks might reveal sensitive data like nonces.

5. Protocol Vulnerabilities: Flaws in the design or implementation of the cryptographic protocol itself could lead to nonce leakage. If the protocol does not adequately protect the nonce, an attacker might be able to intercept or manipulate it.
6. Physical Attacks: If an attacker gains physical access to the device or system where cryptographic operations are taking place, they might be able to observe the nonce value directly from memory or other hardware components.
7. Interception of Communication: If the nonce is transmitted over an insecure channel or not properly protected during communication, an attacker who intercepts the communication might learn the nonce value.
8. Malware or Spyware: Malicious software running on a system could potentially capture or exfiltrate nonce values if it manages to compromise the security of the system.

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 [11]. 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 RAG. A "bad nonce" or "weak nonce" occurs when the same nonce is used for multiple signatures with the same private key. If an attacker can observe such repeated nonces, they might be able to recover the private key. Additionally, if a nonce is generated in a predictable or biased manner, it can also lead to vulnerabilities. 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 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. 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 an integral, linear combinations of basis vector: $X = g_1 b_1 + g_2 b_2 + \cdots + g_n b_n$ where the bi are integers. Figure 2 demonstrates a two dimensional lattice having two generator vectors g_1 and g_2. We put the generator vectors and columns so that a lattice point X is equal to the generator matrix G times B where B is a vector of integers where bz^n is a vector or integers. In Figure 3 we take B is equal to integer vector [0, 0] then X is equals G times B and therefore we get the lattice point as 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 of decreasing the basis B of a given lattice L to a smaller basis B 0 without changing the lattice L. Figure 5 depicts a lattice having two different basis in two dimensions. The determinant of the basis is shaded and the right basis is reduced and orthogonal. Steps to change the basis but to retain the same lattice are as follows:

(1) Exchange the two vectors in the basis.
(2) We use $-b_i$ for a vector $b_i \in B$
(3) We add a linear combination of other basis vectors to $b_i \in B$ vector.

Any vector v in lattice L, is represented as

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

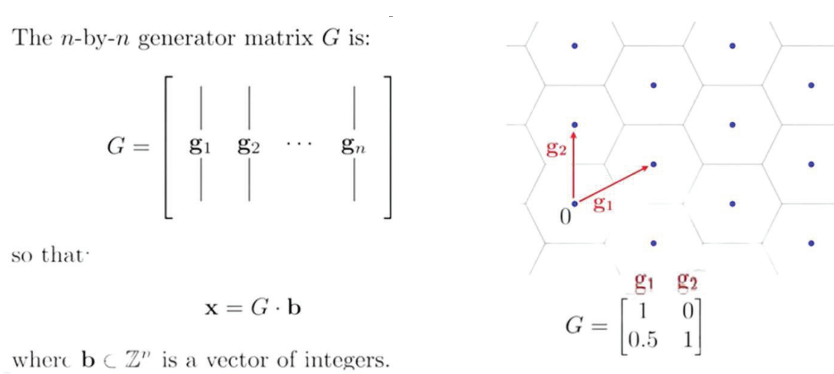

Figure 2 Lattice: linear code over real numbers with generator matrix $N \times N$.

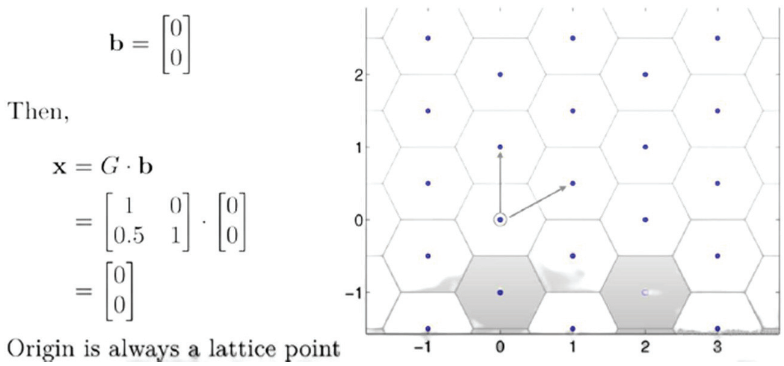

Figure 3 Example 1-Integers to lattice.

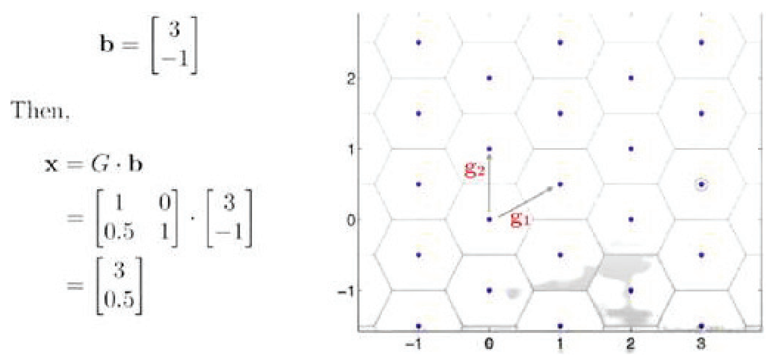

Figure 4 Example 2-Integers to lattice.

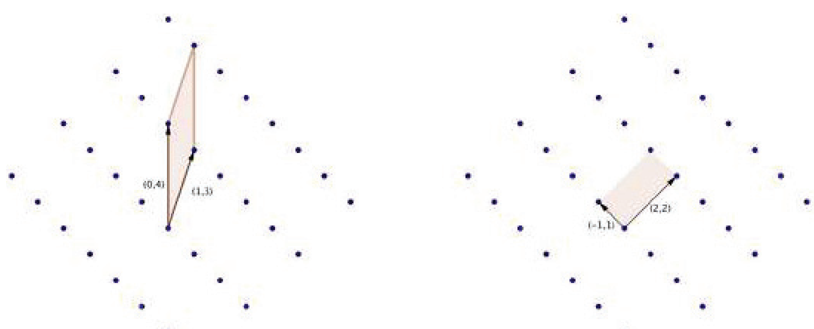

Figure 5 A two dimension lattice with two different basis.

Once inducted, we obtain new basis vector b j, where

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

A lattice L with new basis is represented as

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

Thus, despite changing the basis lattice remains same.

A Lenstra-Lenstra-Lovasz (LLL) algorithm is an estimation of the shortest vector problem; it runs in polynomial time and finds an approximation within an exponential factor of the correct answer. It is a practical method with enough accuracy in solving integer linear programming, factorizing polynomials over integers and breaking cryptosystems. 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

$$\mathrm{u_{i,k} = \frac{b_k^* b_i}{b_i * b_i}} \tag{7}$$

The reduced basis of LLL is $\mathrm{b_1}$, $\mathrm{b_2}$,. . . , $\mathrm{b_n}$ 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 $\mathrm{b_1}$, $\mathrm{b_2}$,. . . , b_n in N-dimension space. The LLL works to get the reduced basis as shown below:

Algorithm 1: LLL Algorithm

Input: $\mathrm{b_1}$, $\mathrm{b_2}$,. . . ,$\mathrm{b_n}$

Continue both the steps until LLL reduced basis is found

Step 1: Gram-Schmidt Orthogonalization

For i = 1 to n do

For k = i–1 to 1 do

m←Closest integer of $\mathrm{u_{k,i}}$

$b_i \leftarrow b_i - m\, b_k$

End for

End for

Step 2: Check Condition 2, and swap

For $i = 1$ to $n-1$ do

If $||b^*_{i+1} + u_{i,i+1} b^*_i||^2 < \frac{3}{4}||b^*_i||^2$ 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 RAG'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.

$$\begin{matrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ r_1\, s_1^{-1} & 1\, r_2\, s_2^{-1} & \frac{B}{N} & 0 \\ m_1\, s_1^{-1} & m_2\, s_2^{-1} & 0 & B \end{matrix}$$

Input Matrix To LLL Algorithm When The Nonce Bias Is Unknown

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 black box 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 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.

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

N=115792089210356248762697446949407573530086143415290314195533631308867097853951
a=-3
b=41058363725152142129326129780047268409114441015993725554835256314039467401291
h=1
Order:115792089210356248762697446949407573529996955224135760342422259061068512044369
Gx=484395612939064517590525852527979142027629495260417479958440807170824046352 86
Gy=361342509567497957985851279195878819566111066729850150718771982535684144 05109
Message 1: Hello
Message 2: Goodbye
Sig 1(R,S):
68436999161162135666328315750279166809195528210448681588885560874606066381971644979542811069069785943188566995276130058885057976202963298436748725973954 72
Sig 2(R,S):
5939666010425204052220844841040305879006138234767515989563012654052795945686824261114452357845949316447297212726196377981816402001485571832274648039476 30
Random value (k1):5440796905206611271057916738553248879 6
Random value (k2):1394947286662891185439150023375931358 44
Private Key:
48588618394399226405893001917337148111899544979674835399706352006027182977592
The private key is found:
48588618394399226405893001917337148111899544979674835399706352006027182977592

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 bug is randomness generated in Yubi keys, in which a bad randomness lead to same value fixed to nearly 80 bits of nonce. Such real world bugs can be easily attacked than the attack that was performed in previous section. In section we are not sure about what the fixed 80-bit values are, whereas in section B we aware that all the fixed 128 bits were all set with zeros. In this technique we assume that all the received collection of signatures whose nonce have 80 fixed bits. We also assume that these fixed 80 bits are most significant bits. (Even if they are not most significant bits still the attack is feasible by just doing left shift one bit at a time which is equivalent of saying multiplying the signature by 2). Here we are not aware what these 80 bits are, by subtracting any two nonce's, the 80 most significant bits of their differences will all be zeros. We apply the same lattice attack as explained in section B except our signature values subtracted. With a set of n signatures and messages we will build the below matrix and is given as input to LLL algorithm and which will in turn generate a new output matrix. The output matrix of LLL algorithm is k_1–k_n, i.e. the variance between the nonce's for signatures 1 and n. In this we differentiated nth value from every entry in matrix, in lieu of having a complete row full of nonce's we literally have a row with the variance between every nonce and the n^{th} nonce.

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

Input Matrix To LLL Algorithm When The Nonce Bias Is Unknown

One can recover the secret key using below formulations:

(1) $\mathrm{Sig_1} = k_1^{-1}(\mathrm{Msg_1} + xr_1)$ and $\mathrm{Sig_n} = k_n^{-1}(\mathrm{Msg_n} + xr_n)$
(2) $\mathrm{Sig_1}k_1 = \mathrm{Msg_1} + xr_1$ and $\mathrm{Sig_n}k_n = \mathrm{Msg_n} + xr_n$
(3) $k_1 = \mathrm{Sig_1^{-1}}(\mathrm{Msg_1} + xr_1)$ and $k_n = \mathrm{Sig_n^{-1}}(\mathrm{Msg_n} + xr_n)$
(4) $k_1 - k_n = \mathrm{Sig_1^{-1}}(\mathrm{Msg_1} + xr_1) - \mathrm{Sig_n^{-1}}(\mathrm{Msg_n} + xr_n)$
(5) $\mathrm{Sig_1Sig_n}(k_1 - k_n) = \mathrm{Sig_n}(\mathrm{Msg_1} + xr_1) - \mathrm{Sig_1}(\mathrm{Msg_n} + xr_n)$
(6) $\mathrm{Sig_1Sig_n}(k_1 - k_n) = x\mathrm{Sig_n}r_1 - x\mathrm{Sig_1}r_n + \mathrm{Sig_n Msg_1} - \mathrm{Sig_1 Msg_n}$

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

The secret key can be easily recovered from only five signatures if generated signatures are produced from nonce's with 80 fixed bits. To reduce the error rate we build the above matrix with n = 6. In real world the generating 80 fixed bits are sparse. Such kind of attack is much more robust when applied with 256 bit elliptic curves, this attack works well even when 4 bits of nonce are fixed. Implementation does not become complicated rather one needs to only increase the dimension of lattice i.e., attacker has to only escalate the value of n and repeat the attack. This technique will increase the running time of algorithm but not the complexity. In our experiments the value of N is total number of signatures required to retrieve secret key and are derived experimentally by trying to attack with dissimilar number of signatures on different amount of fixed bits. The value of N = 2 when nonce had the first 128 bits fixed to 0, the value of N = 3 when 128 bits are fixed and we do not know to what values they are fixed. The value of N = 5 when the nonce had 80 randomly fixed bits.

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

Table 5 Features of nodes used in our research

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

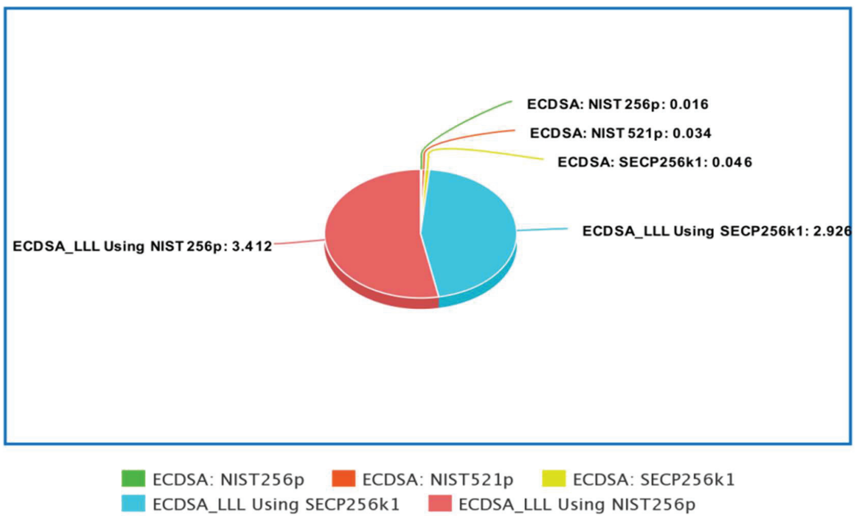

Figure 6 Average execution time to crack ECDSA (Seconds).

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 RAG. 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.

Acknowledgement

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

References

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

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

[3] Mohammad Ayoub Khan, Mohammed Tabrez Quasim, Norah Saleh Alghamdi, Mohammad Yahiya Khan. 2020 "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.

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

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

[6] Dan Boneh & Ramarathnam Venkatesan 2001 "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, volume 1109, pp. 129–142.

[7] Badis Hammi, Achraf Fayad, Rida Khatoun, Sherali Zeadally and Youcef Begriche 2020 "A Lightweight ECC-Based Authentication Scheme for Internet of Things (IoT)" IEEE Systems Journal Pages: 3440–3450 DOI: 10.1109/JSYST.2020.2970167, Volume: 14.".
[8] Joachim Breitner and Nadia Heninger 2019 "Biased Nonce Sense: Lattice Attacks against Weak ECDSA Signatures in Cryptocurrencies" Lecture Notes in Computer Science Springer International Publishing – Financial Cryptography and Data Security.
[9] Javed R. Shaikh, Maria Nenova, Georgi Iliev and Zlatka Valkova-Jarvis 2017 "Analysis of Standard Elliptic Curves for the Implementation of Elliptic Curve Cryptography in Resource-Constrained E-commerce Applications" IEEE-COMCAS ISBN:978-1-5386-3169-0.".
[10] Shen Guicheng, Yu Zhen 2013 "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.".
[11] Ravi Kishore Kodali and Ashwitha Naikoti 2016 "ECDH based Security Model for IoT using ESP 8266" IEEE – ICCICCT DOI: 10.1109/ICCI CCT.2016.7988026".

Biographies

Mohammed Mujeer Ulla, currently working as Assistant Professor- Selection Grade in School of computer science and engineering since 2017. He is an alumnus of R.V college of engineering- Bangalore in his UG and PG. And received the philosophy of doctorate degree in Computer Science and Engineering from Presidency University, Bangalore, respectively. He has many papers to her credit in reputable international journals, national journals, and conferences. He has been serving as a reviewer for highly respected journals. His areas of expertise include internet of Things, Wireless sensor network.

Preethi, received the bachelor's degree in computer science and engineering from VTU, Karnataka in 2008, the master's degree in computer science and engineering from VTU, Karnataka 2013, and the philosophy of doctorate degree in Computer Science and Engineering from Presidency University, Bangalore in 2022, respectively. She is having total 15 years of Teaching experience. She is currently working as an Assistant Professor-Senior Scale, Manipal Institute of Technology, Bengaluru, Manipal Academy of Higher Education, Manipal, India. Her research areas include the Internet of things, Computer Architecture and cryptography. She has many papers to her credit in reputed international journals, national journals and conferences. She has been serving as a reviewer for highly-respected journals.

Md. Sameeruddin Khan, currently working as Professor and Dean in the School of Computer Science and Engineering, Presidency University, Bangalore. He received his B.E in from Gulbarga University, Gulbarga. M.Tech in in Computer Science and Engineering from Visveswaraih Technological University, Belgaum. Doctor of Philosophy in Computer Science and Engineering from Rayalaseema University, Kurnool, Andhra Pradesh.

Deepak. S. Sakkari, currently working as Professor in the Department of Computer Science and Engineering, Sri Krishna Institute of Technology, 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.