Security and Performance Analysis of Elliptic Curve Crypto System using Bitcoin Curves

Mohammed Mujeer ulla, Md Sameeruddin Khan, Preethi, and Deepak S.Sakkari

Abstract—**This paper analysis an attack on crypto currency signatures generated by Elliptic Curve Digital Signature Algorithm (ECDSA). This attack is done on three different dimensions with leakage of nonce weak nonce and by using Lenstra–Lenstra–Lovasz (LLL) by considering various crypto currency curve parameters such as number of private keys, single nonce, multiple nonces, single signature, multiple signatures on varying bit sizes of length 64, 128, 256, 512 and 1024. This work demonstrates demonstrate leak-weak nonce method performs exponentially better over LLL to crack ECDSA. In addition to finding multiple vulnerabilities on ECDSA and to avoid leak-weak nonces we use T-of-N threshold signing algorithm based on GG20 where nonce k is distributed across N parties unlike having with one single party. This technique is implemented and evaluated considering group of N parties. This paper brings out the pitfalls of ECDSA, when it is used to sign crypto currency transactions.**

Keywords— Elliptic Curve Cryptography (ECC), Securities and Exchange Commission (SEC), National Institute of Standards and Technology (NIST), Edwards curve Digital Signature Algorithm (ECDSA), Nonce - number only used once.

I. Introduction

Many sensitive and confidential data shared in real-time applications like online banking system includes third-party applications such as UPI payment schemes. Due to abrupt advances in technology, trading centre and data access remotely in healthcare-based applications, IIOT, defence, retail market, and many other industries are at an unacceptably high level [1]. Public-key cryptosystem ensures confidentiality, integrity, authentication and hence used by several internet security protocols.

Manuscript received April 23, 2022; revised March 3, 2023.
This work was supported in by Presidency University - Research and Innovation Council for fifty thousand Indian rupees, Bangalore, Karnataka, India - 560064.

Mohammed Mujeer Ulla is Assistant professor at Presidency University ,Bangalore, Karnataka, India - 560064. e-mail : mujerroshan@gmail.com.

Md. Sameeruddin Khan is Dean at Presidency University, Bangalore, Karnataka, India - 560064. e-mail: sameeruddinkhan@presidencyuniversity.in.

Preethi is Assistant professor at Presidency University, Bangalore, Karnataka, India - 560064. e-mail : preethisrivathsa@gmail.com.

Deepak. S. Sakkari is Associate Professor at Presidency University, Bangalore, Karnataka, India - 560064. e-mail : sakkari@hotmail.com.

Elliptic Curve Digital Signature Algorithm (ECDSA) is commonly used public-key protocol over internet communication. Few specific areas of ECDSA are TLS, PGP, Smart card, Digital currencies like Bitcoin, Ripple and Ethereum. The listed areas find wide application of ECDSA. The ECDSA algorithm considered as highly secured due to its strong computing of discrete logarithm problem. Also it generates small key length for signing which is considers as fastest method. Hence it is highly recommended by IEEE and NIST since 2000, ANSI since 1999 and ISO since 1998 [2].

The ECDSA security is based on accurate generation of per-signature nonce which is also known as ephemeral private key. There are many ECDSA flaw's that one could use to uncover the private key, one among them is the use of repeated nonce for generating signatures [3]. In this paper we initially begin by collecting signatures from bitcoins and ethereum, then we use them to efficiently compute their private keys. In addition we find couple of bitcoin and ethereum blockchain private keys that were generated through iterated uses of nonces. We have employed two methods to attack ECDSA signatures downloaded from bitcoin and ethereum block chain. In first technique we try to attack signatures targeting leak-weak and shared nonces and in second method we use LLL algorithm to attack the signatures on NIST and SECP curves. The popular tool used in cryptanalysis is lattice reduction. This tool is used by majority of the cryptosystems like RSA/Knapsack and are broken using lattice reduction. ECDSA also uses this tool combining lattice reduction , discrete algorithm and performing factorization on composite numbers. LLL algorithm introduced by Lenstra, Lenstra and Lovasz is one among the known technique for lattice reductions. Currently, the variants of the lattice algorithms are used for lattice reduction. The main focus in this paper is on applying leak-weak nonces, shared nonces and LLL to find private keys of various length in terms of bits like 64 ,128, 256, 512, and 1024.

The paper is organized as follows into various sections. Section II gives a theoretical proof of Elliptic Curve Digital Signature (ECDSA) and the LLL Algorithm. Section III describes three divisions, firstly- ECDSA: Disclosing the private key, due to leak-weak and shared nonce using NIST and SECP curve, secondly-ECDSA: Disclosing the private key using Lenstra–Lenstra–Lovasz (LLL) method, if nonce known, Finally-ECDSA-Disclosing the private key using Lenstra–Lenstra–Lovasz (LLL) method, if nonce known with real-world ECDSA bugs, also discusses-ECDSA: T-of-N threshold digital signing algorithm based on GG20 cryp-tology. Section IV provides complete details of result analysis and section V concludes the work with future enhancement.

II. Theoretical Principle

A. Elliptic curve digital signature (ECDSA):

The Elliptic-Curve Digital Signature (ECDSA) is a public-key used in digital signing algorithm. The keys generated using ECDSA are exponentially smaller in length compared to the keys generated by any other digital signing algorithm. The ECDSA public domain parameters include an include an elliptic curve E on a field F which is deterministic with a base point G of order$N$, where N is a very large integer. The signature verification using public key is performed by Q= dG is considered as a point on E, where d is a private key such that $d = d$ modulo N. The elliptic curve public keys are normally manifested in compressed way by just furnishing the x coordinate of intersecting point on elliptic curve and single parity bit of y value. In order to sign a message hash h, the sender selects empirical private key, also known as nonce (k), where $k = k$ modulo N, with this we have publicly verifiable point k.G on the curve. Then the sender computes $(x_r, y_r) = kG$ MOD N and outputs the signature (r,s) where $r = x_r$ and $s = k^{-1}(h+dr)$ MOD N. At the other end, receiver checks the authenticity for the message hash using public key Q, by computing $(x'_r, y'_r) = hs^{-1}G+rs^{-1}Q$, this is pictorially shown in fig.1.

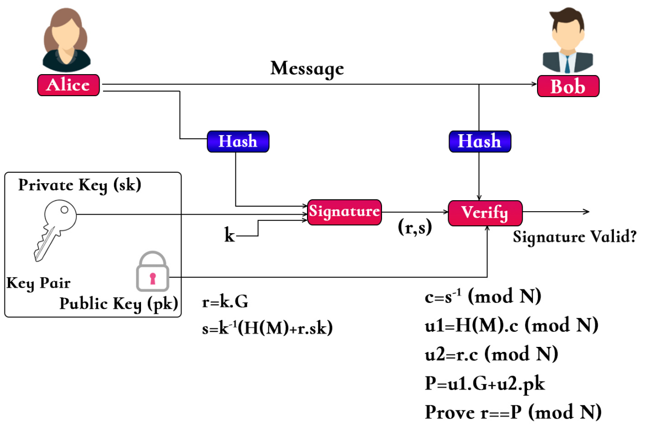

Fig 1. ECDSA

B. The LLL Algorithm :

There are several algorithms to find orthogonal basis. The best and efficient technique is LLL algorithm. This algorithm was invented by great researchers: Lenstra, Lenstra and Lovasz. LLL algorithm mainly consists of two conceptual parts:

- The working vector can become the successor of basis vector or the working vector as it is replaced as basis vector. This decision can be taken by reducing a non-basis vector/working vector by reducing multiples of the currently used basis vectors.
- The decision is based on satisfying Lovasz condition. Lovasz condition checks the size of the working vector. If it is large in bit size, the subsequent basis vector are generated easily [4], here we maintain track of two sets of vectors:
- $\vec{v_1}$,,......, the current set of basis vectors, which we are attempting to minimize to a set that is approximately orthogonal
- $\vec{v_1}^*$, $\vec{v_2}^*$,......, the set of orthogonal basis vector produced by the Gram-Schmidt reduction.

We also need to keep track of k, K is the number of the working basis vectors we are attempting to minimize, is another important parameter to monitor. Suppose our basis vectors are $\vec{v_1}$,$\vec{v_2}$,......,$\vec{v}_{k-1}$,$\vec{v_K}$,.... and we are working to reduce $\vec{v_K}$. We do this by subtracting multiples of $\vec{v_1}$, $\vec{v_2}$,......,$\vec{v}_{k-1}$. Now let us consider the vectors $\vec{v}_{k-1}$ and $\vec{v_K}$, if these were only two vectors we had we might need to subtract some multiples of new $\vec{v_K}$ from the old $\vec{v}_{k-1}$. 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 $\vec{v}_{k-1}$ as new working vector. The decision of whether to swap $\vec{v}_{k-1}$ with $\vec{v_K}$ and to make $\vec{v}_{k-1}$ the working vector is based on whether the Lovasz condition is satisfied or not. In addition to basis vectors $\vec{v_1}$, $\vec{v_2}$,......,$\vec{v}_{k-1}$,$\vec{v_K}$ for the lattice we also have the orthogonal basis $\vec{v_1}^*$, $\vec{v_2}^*$, $\vec{v_3}^*$,.... found from the Gram-Schmidt reduction. Let $\vec{v_K}$ be the working vector and

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

If $||\vec{v_k}^*||^2 \geq \left(\frac{3}{4} - \mu^2_{k,k\,-1}\right)$, then we have completed the comparison with $\vec{v_k}$ and can make $\vec{v^*_{k+1}}$ the next working vector, otherwise, we swap $\vec{v^*_{k-1}}$ and $\vec{v_K}$ and make $\vec{v^*_{k-1}}$ as the working vector. A toy example demonstrating above steps in LLL algorithm is shown in appendix.

III. Methodology

C. Using NIST and SECP Curves, Disclosing ECDSA Private Key with Leak, Weak, and Shared Nonce:

In this section we uncover the ECDSA-private keys by applying elementary attacks. Firstly if attacker gets to know the nonce k for every message used to generate ECDSA signature then it is easy to compute the secret key d by having d= $(sk - h)$ r^{-1} MOD N. Secondly if two different messages signed by h_1 , h_2 with same nonce k and secret key, then it is trivial to compute secret key. For instance let (r_1, s_1) and (r_2, s_2) be the signatures generated on two message hash h_1 and h_2 respectively then one can compute secret key k= $(h_1 - h_2)(s_1 - s_2)^{-1}$ MOD N. In addition to above said attack we apply, two optimizations to LLL algorithm. In first optimization , most significant bits known in which b_i is assumed to be positive value and we gradually raise the bias by centering the b_i by having the equation $x_i^{'} t_i y + a_m + B \cong 0$ MOD p which the solution $x_i^{'} = b_i B$. In second optimization we decrease the lattice dimension by one by eliminating y variable from $x_1 t_1 y + a_1 \cong 0$ MOD p so that we have $m - 1$ equations with m unknowns all bounded.

Case 1: Disclosing the private key, due to leak of nonce on NIST curves

As discussed earlier with ECDSA a message hash is signed with its private key and the authenticity of signed message hash is proved with the public key. Each time we sign a message we randomize the nonce value to generate a different verifiable signature each time. Over all the signer has to reveal only the message signature with its public key and not the nonce value. If the signer reveals the nonce then intruder/third party easily finds the private key. Fig.2 demonstrates revealing the private key as a result of a nonce leak. Initially both sender

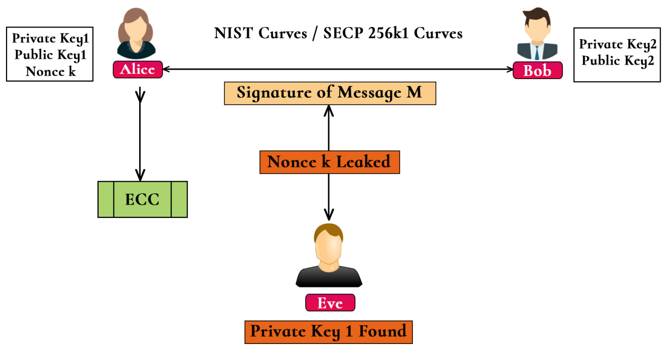

Fig 2. Leaked Nonce

and receiver agree on global parameters for ECDSA message exchanges, steps to disclose the ECDSA private key, due to leak of nonce using NIST curves are as follows:

- Sender selects his private key d and nonce value k to sign the message hash,
- Sender computes his public key = d * G, which is a verifiable point on curve by any other parties who mutually commit on ECDSA,
- The intersecting point on elliptic curve E is the signature (r,s) for message hash, where r = k.G and s = $k^{-1}(H(M) + r.d)$.

Suppose the sender's announcement of value k has been leaked to intruder, and the steps followed by intruder to recover private key if he has signature (r,s) and nonce k is as follows:

- Intruder uses leaked values r=k.G and

$$s = k^{-1}(H(M) + r.d) \qquad (1)$$

- Using equation (1) we get

$$s.k = H(M) + r.d \qquad (2)$$

- Using equation (2) we get

$$r.d = s.k - H(M) \qquad (3)$$

- Using equation (3) we get

$$d = r^{-1}(s.k - H(M))MODN \qquad (4)$$

Table-V and table-VI shows disclosing the ECDSA private key due to leak of nonce over NIST and SECP curves for multiple signatures respectively.

Case 2.1: Disclosing the private key, due to weak nonces using SECP curves on multiple signatures

One of the catastrophic failure of ECDSA is nonce reuse. In simple terms if multiple signatures are signed using same nonce k. Suppose if we use same nonce k, with r and secret key to create two signatures pairing $(r, s_1) and (r, s_2)$ for two unique messages m_1 and m_2 then we have

$$s_1 = k^{-1}(h_1 + r.sk) \text{ and } s_2 = k^{-1}(h_2 + r.sk) \qquad (5)$$

Equation 5 enables us to leak the secret key with,

$$\frac{= s_2.h_1 - s_1.h_2}{r(s_1 - s_2)}$$

and so k can be leaked

$$= \frac{h_1.h_2 + r.h_1.priv - h_1.h_2 - r.h_2.priv}{r.h_1.r.priv - r.h_2 - r.priv}$$

$$= \frac{r.h_1.priv - r.h_2.priv}{r.h_1 - r.h_2} = priv$$

Nonce is also recoverable with

$$= \frac{h_1 - h_2}{s_1 - s_2} = \frac{h_1 - h_2}{k^1(h_1 - h_2 + priv(r - r))} = k$$

Fig.3 demonstrates disclosing the private key, due to weak nonces using SECP curves on multiple signatures.

Case 2.2: Disclosing the private key, due to weak nonces using LLL on NIST curves with multiple signatures

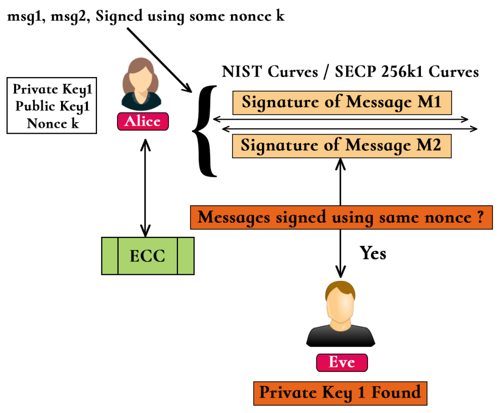

Fig 3. Weak Nonce

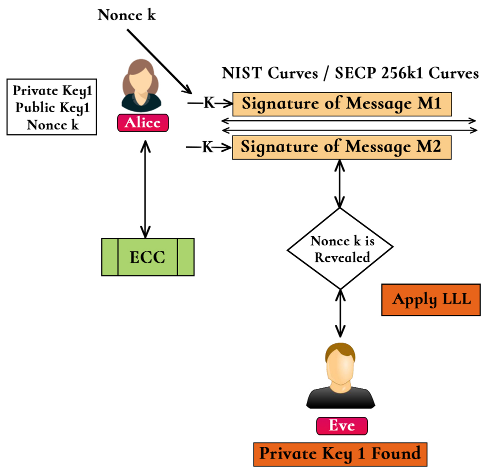

Fig 4. LLL on multiple signatures

Quantum computing is a threat to to data security and quantum computers can crack traditional encryption methods. Most of the traditional encryption algorithms strength rely's on modular arithmetic. Shors algorithm works on modular arithmetic to find the prime factors of any integer there by paving a way to crack traditional encryption schemes. Quantum computers with Shors algorithm would thus be more effective in cracking traditional encryption schemes and therefore a method to resist such attacks is to use encryption schemes based on lattices. A lattice based encryption is one that relies on hard math problems involving lattices [5]. A lattice is simple a mathematical structure used to represent an infinite grid of points. Every lattice is formed from basis vector, when multiplied together can form any point in lattice. A lattice can comprise of infinite number of dimensions and infinite number of basis vectors. To use lattices for encryption involve a hard problem called as shortest vector problem. As we add more and more number of dimensions the problem becomes more and more complicated until you find a solution to break them. Here using LLL algorithm as a black box and we will to attack signatures generated from bad nonce or random number generator. These nonces will have fixed prefix or suffix, where many of the least significant bits or most significant bits are repeated. Even though if least or most significant bits are not equal, this attack works normally. Therefore, the LLL algorithm starts with a matrix input. A collection of ECDSA signatures make up this input matrix. The expected output from this algorithm is a matrix with all nonces. This nonces enables to generate ECDSA private keys described in case 1 and 2.1. Fig. 4 demonstrates application of LLL algorithm on multiple signatures.

A lattice is denoted by λ and it is an additive subgroup of real numbers. Its represented by a basis vector $g_1, g_2, ...g_n$ in N-dimensional space. Consider X is a lattice point represented as integral- linear combinations of basis vector such as X=$g_1b_1 + g_2b_2 + ... + g_nb_n$ where the bi are integers. Fig. 8 illustrates a two-dimensional lattice having two generator vectors i.e g_1 and g_2. By mapping the generator vectors and columns in a matrix forms a lattice point X is equal to the generator matrix G times B where B is a vector of integers where b $\in z^n$ is a vector or integers. In fig. 6 we choose B as equal to integer vector [0,0], then X is equals to G times B and therefore we get the lattice point as 0. In fig. 7 we consider 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 new basis B' comparatively smaller without changing the lattice L.Figure 5 represents a lattice having two different basis in two dimensional structure. The determinant of the basis in Figure 5 is shaded and the right basis is reduced and orthogonal. The following procedure changes the basis by retaining the same lattice : 1. Swap the two vectors in the basis
2. b_i used for a vector, $b_i \in B$
3. Add a linear combination of other basis vectors to $b_i \in B$ vector.

Any vector v in lattice L, is represented as

$$v = \sum_{i=0}^{m} z_i b_i \tag{6}$$

Once inducted, we obtain new basis vector b_j, where

$$b_j = b_j + \sum_{i \neq j} y_i b_i, y_i Z. \tag{7}$$

A lattice L with new basis is represented as,

$$v = \sum_{i \neq 0} z_i b_i + z_j (b_j + \sum_{i \neq j} y_i b_i) \tag{8}$$

Thus, despite changing the basis lattice remains same.

$$\begin{pmatrix} [N] & 0 & \dots & 0 & 0 \\ 0 & [N] & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 \\ [r_1s_1^{-1}-r_ns_n^{-1}] & [r_2s_2^{-1}-r_ns_n^{-1}] & \dots & [B/N] & 0 \\ [m_1s_1^{-1}-m_ns_n^-1] & [m_2s_2^{-1}-m_ns_n^-1] & \dots & 0 & [B] \end{pmatrix}$$

Input Matrix To LLL Algorithm: for the Unknown Nonce Bias.

The LLL algorithm is the most effective estimating method for the shortest vector issue. It operates in polynomial time and finds an approximation of the desired result that is within an exponential factor. It is a practical approach with sufficient accuracy for breaking cryptosystems, factoring polynomials over integers, and solving integer linear programming.Let $b_1, b_2,$ ůůů$, b_n$ be a basis for a N-dimensional lattice L, and $b_1^*, b_2^*,$ ůůů$, b_n^*$ be the orthogonal basis and we have

$$u_{i,k} = \frac{b_k^* b_i}{b_i^* b_i} \tag{9}$$

The reduced basis of LLL is $b_1, b_2,$ ůůů$, b_n$ if following two conditions are met:
(1) $\forall i \neq k, u_{i,k} \leq \frac{1}{2}$,
(2) for each i,$||b_{i+1}^* + u_{i,i+1} b_i^*||^2 \geq \frac{3}{4}||b_i^*||^2$

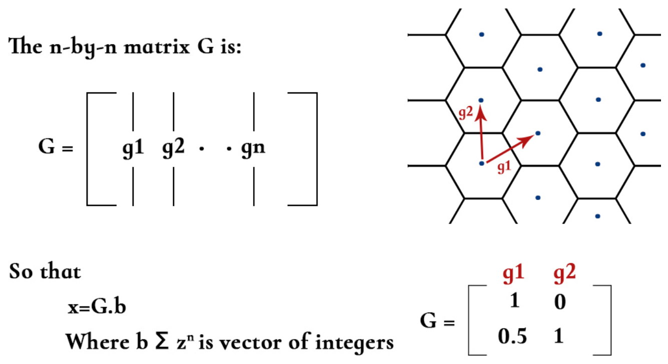

Fig 5. Lattice: linear code over real numbers with generator matrix NxN

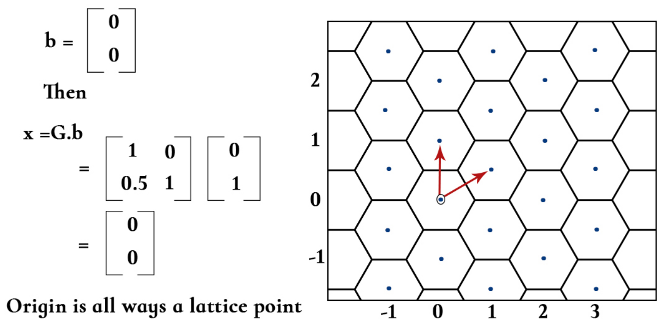

Fig 6. Example 1:Integers to Lattice

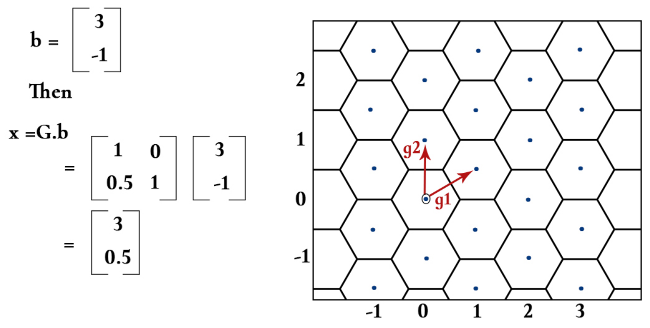

Fig 7. Example 2:Integers to Lattice

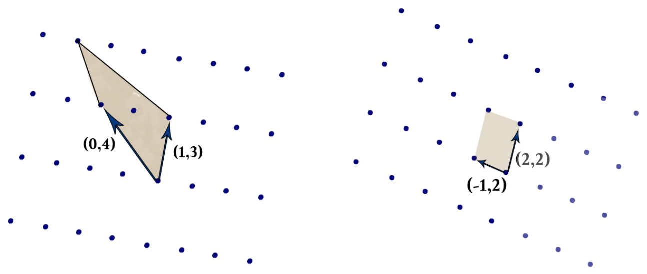

Fig 8. A two dimension lattice with two different basis.

The constant values from $\frac{1}{4}$ and 1 determines the termination of algorithm in polynomial time. The constant chosen in this work is 0.75. Another important condition is the ordering of the basis. Consider a basis $b_1, b_2,$ ůůů$, b_n$ in n-dimension space. This LLL algorithm reduces basis as shown below:

Algorithm 1: LLL Algorithm

```
Input:$b_1, b_2, \cdot \cdot \cdot \cdot \cdot \cdot, b_n$
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 $||b_{i+1} + u_{i,i+1} b_i||^2 \geq \frac{3}{4}||b_i||^2$ then
            swap b$_{i+1} and b_i$
            go to step 1
        end if
    end for
```

To perform this attack on NIST P-256 elliptic curve we use ECDSA and LLL library functions in python, as it allows us to input our own nonces of size 64, 110, 128 , 512 and 1024 in bits generated from bad random number generators.

$$\begin{bmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ r_1 s_1^{-1} & r_2 s_2^{-1} & B/N & 0 \\ m_1 s_1^{-1} & m_2 s_2^{-1} & 0 & B \end{bmatrix}$$

As infant steps we begin work with two ECDSA signatures and then we increases the lattice dimensions to work with multiple signatures.

The upper bound limit established for our nonce's is B in the aforementioned matrix, where N is the order of NIST P-256 (Nonces used in our research study are of 64,110,128,512 and 1024,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 given input message. Once the matrix is obtained it is given as input to black-box i.e LLL algorithm, which will output matrix that contains nonce used multiple times to sign the signatures.

Assume there are two signatures (r, s_1) and (r, s_2) derived on messages msg_1, msg_2 respectively. These messages are from same nonce k then r value will remain same for both messages as the k value is same so one can easily detect the ECDSA private key using below formulations 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) $k = (sig_1 – sig_2)^{-1}(Hash(msg_1) – Hash(msg_2))$

The public-key for the produced signature is accessible to the attacker alone. Therefore, one might decide with ease whether to compute the associated ECDSA public-key or to find the matching ECDSA private-key, both of which are easily accessible. This approach has a significant failure rate for this form of attack, which is a big flaw. By conducting the same attack with increasing number of signatures, the failure rate can be reduced. Table-VIII depicts the ECDSA private key using weak nonces on NIST curves. Table-VIII discloses the ECDSA private key using weak nonces on NIST curves. Table-VIII shows disclosing the ECDSA private key using weak nonces on NIST curves and Table- IX shows disclosing the ECDSA private key using weak nonces on SECP curves with LENSTRA–LENSTRA–LOVASZ (LLL) method respectively.

TABLE I
ECDSA: Disclosing the private key, due to weak nonce (NIST521p Recommended Parameters)

CurveFp=6864797660130609714981900799081393217269435300143305409394463459185543183397656052122559640661454554977296311391480858037121987999716643812574028291115057151
a=-3
b=1093849038073734274511112390766805569936207598951683748994586394495953116150735016013708737573759623248592132296706313309438452531591012912142327488478985984
h=1
Order: 6864797660130609714981900799081393217269435300143305409394463459185543183397655394245057746333217197532963996371363321113864768612440380340372808892707005449
Gx=2661740802050217063228768716723360960729859168756973147706671368418802944996427808491545080627771902352094241225065558662157113545570916814161637315895999846
Gy=3757180025770020463545507224491183603594455134769762486694567779615544477440556316691234405012945539562144444537289428522585666729196580810124344277578376784
Message 1: The International Association of Engineers
Sig 1(R,S):107444816945277544813011427209657010974218458132716230633581708442672398386102723221068918047415751050159644896292707687256978606128136185568346447514068964964681793480895517360266081837903158283000125842243622974737750474201050860686613102995026306787155941038781620904847813101796647616679936497024015424928259 01
Random value (k): 41747331298864425333522831818903323854
Private Key:92521831048352635813262230461374105016393689692081640879310408802395548540064574354558426732280282576699565765718097716671016279396582768819741535 1115272467
The private key is found:9252183104835263581326223046137410501639368969208164087931040880239554854006457435455842673228028257669956576571809771667101627939658276881974153511 15272467

Unpredictable damage to a signature might result from leaking fractional parts of nonce. The work carried out by N.A. Howgrave-Graham, N.P.Smart showed the application of lattice attacks to break DSA from partial nonce leakage [6]. Nguyen and Shparlinski extended their work after this to extract a secret key from a 160-bit DSA, and from every 100 signatures in an ECDSA, a secret key was retrieved by only understanding the first three bits of each nonce [7]. Mulder et al. continued their study by performing additional attacks on partial nonce leakage using a Fourier transform-based attack and re- covered secret keys from 384-bit ECDSA with only a 5-bit knowledge of each nonce from 4,000 signatures [8]. Most of us are aware of lattice attacks and Minerva attacks, which use a number of temporal side channels to recover partial nonce leaking [9]. Even if the size of the nonce was exposed, they were still able to retrieve the private key by using enough signatures. In the most recent attack, dubbed the "Ladder Leak Attack," which is considerably worse than the "Fourier Analysis Attack," one might retrieve secret keys only by having a one-bit nonce that has been published [10] [11]. Further, even if one is successful in keeping his nonce a secret, never divulge any information about it and never use the same nonce twice. The contribution of Heninger and Breitner shows the use of lattice attacks has the potential to attack signature schemes that use flawed random numbers [12] [13]. If a bias of 4 bits is applied using 256-bit ECDSA on nonce, even though you are unaware of the biased values, your signature scheme is entirely broken [14].

Case 3: Disclosing private key, from multiple signed messages and shared nonces

In some cases secret keys leak even if same nonces are not reused with secret keys. For instance two private keys x_1, x_2 are used with two nonces k_1, k_2. Here nonces are not repeated twice with same key, but nonces are reused across the keys k_1, k_2. The system has four solvable independent linear equations with four unknowns. The problem can be simplified as an attacker extracts a set of signatures from bitcoin block chain to leak as many possible keys or nonces there by solving system of liner equations. In this setup sender has two private keys x_1, x_2 and two nonces k_1, k_2 and four messages m_1 m_2 m_3 m_4 to sent. Message m_1 is signed using x_1 and nonce k_1 , message m_2 is signed using x_2 and nonce k_1, similarly m_3 is signed using x_1 and nonce k_2, m_4 is signed x_2 and nonce k_2 as shown in fig.9 The hashes for all the messages are as follows:

$h_1 = Hash(m_1)$
$h_2 = Hash(m_2)$
$h_3 = Hash(m_3)$
$h_4 = Hash(m_4)$

The generated signatures for all the messages are (s_1, r_1),(s_2, r_1),(s_3, r_2) and (s_4, r_2) where

$s_1 = {k_1}^{-1}(h_1 + r_1.x_1)MODP$
$s_2 = {k_1}^{-1}(h_2 + r_1.x_2)MODP$
$s_3 = {k_2}^{-1}(h_3 + r_2.x_1)MODP$
$s_4 = {k_2}^{-1}(h_4 + r_2.x_2)MODP$

private keys can be easily recovered using Gaussian eliminations,

$$x_1 = \frac{h_1r_2s_2s_3 - h_2r_2s_1s_3 - h_3r_1s_1s_4 + h_4r_1s_1s_3}{r_1r_2(s_1s_4 - s_2s_3)} \quad (10)$$

$$x_2 = \frac{h_1r_2s_2s_4 - h_2r_2s_1s_4 - h_3r_1s_2s_4 + h_4r_1s_2s_3}{r_1r_2(s_2s_3 - s_1s_4)} \quad (11)$$

Table-VII shows disclosing the ECDSA private key due to weak nonces from multiple signed messages and shared nonces using SECP curves on multiple signatures

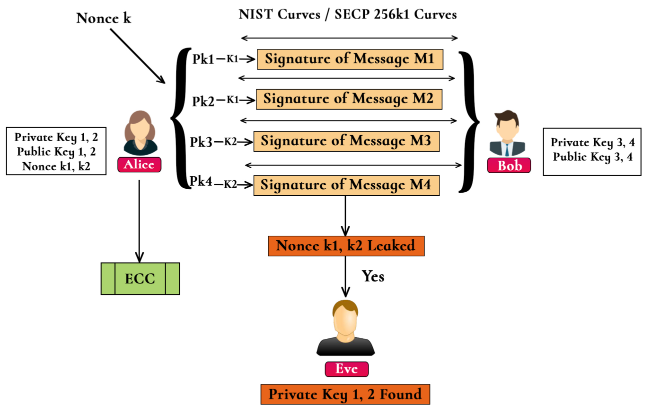

Fig 9. Disclosing private key from multiple signed messages and shared nonces

D. *ECDSA-Disclosing the private key using Lenstra–Lenstra–Lov ́asz(LLL) method: for nonce known with real-world ECDSA bugs:*

If generated signatures are made from nonces upto 82 fixed bits, the secret key can be easily extracted from generating five to six signatures. We constructed the required matrix with N=6 to decrease the error rate. The nonce of 82 fixed-bit generation is sparse in the actual world.This kind of attack is far more powerful when employed with 256-bit elliptic curves and remains successful even when the nonce's first four bits are fixed. The attacker only needs to increase the size of the lattice or increase the value of N and repeat the attack; the implementation does not become challenging. The algorithm will take longer to execute using this technique without becoming more difficult. We computed the value of N empirically by trying to carry out an attack using a different number of signatures on a different number of fixed bits. N in our tests stands for the total number of signatures required to obtain the secret key. N=2 when the first 128 bits of the nonce were fixed to 0, and N=3 when the first 128 bits were fixed but their fixed values were unknown. When the nonce's 80 fixed bits were randomly selected, N=5. Using the formulae below, one can discover the secret key:

1) $Sig_1 = k_1^{-1}(Msg_1 + xr_1) and Sig_n = k_n^{-1}(Msg_n + xr_n)$
2) $Sig_1k_1 = Msg_1 + xr_1 and Sig_nk_n = Msg_n + xr_n$
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) $Sig_1Sig_n(k_1 – k_n) = Sig_n(Msg_1 + xr_1) – Sig_1(Msg_n +$

$xr_n)$

6) $Sig_1 Sig_n(k_1 – k_n) = xSig_n r_1 – xSig_1 rn + Sig_n Msg_1 – Sig_1 Msg_n$

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 = $(r_n Sig_1 – r_1 Sig_n)^{-1}(Sig_n Msg_1 – Sig_1 Msg_n – Sig_1 Sig_n(k_1 – k_n))$

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

Message 1: The International Association of Engineers **Message 2:** IEEE Journal
Sig 1(R,S):95373752039725530428968840472943405024508846645925964488109526096527853332560860110338437959764400154877644964007159370696115252631574204399113341411688816
Sig 2(R,S):79937594961494168954501398582580902762471405121278159831627793097261654947217940034597301294531782880927967969195724170990365969399656053732247537721461 04
Random value (k1):544079690520661127105791673855 **R**andom value (k2):51894268455819342715039874460607282054
Private Key:21463921422229879901409461769575568583853786173485411408350838442455737947355
The private key is found: 214639214222298799014094617695755685838537861734854114083508384424557379473 55

*E. **ECDSA: X-of-Y threshold signing algorithm using GG20 with Kryptology:***

Several independent nonce vulnerabilities were computed to find ECDSA private keys in section III-A and III-B. In this section we will discuss ECDSA threshold multiparty case which enables us to have the nonce k distributed across Y parties than having with one single party using coin-base kryptology on SECP256k1 curve and generate the private key. In this method, a minimum of X shares are needed to compute the signature for Y parties. Then a create the private key that helps in signing the message using them. A valid signature requires the participation of X persons or more once the generated private key is divided into multiple shares (Y). In this manner we construct an ECDSA signature using multiple Shamir shares there by not revealing the private but but rather is split over multiple parties.We do not store the private key on parties and is deleted once the shares are distributed. Later stage the public key can be generated and split the key into multiple shares. The signatures created by parties are verified by public key.

Elliptic curves are basically defined as equation of form $y^2 = x^3 + ax + b$ over prime field P and all points that are generated on elliptic curve are within the prime field P. Each of the selected NIST curves have predefined curve parameters such as generator point G, prime P>3 and variables a, b where a,b Galois field (p). Elliptic curves can be defined over either NIST or SECP curves. An elliptic curve is a tuple (G, **G**,q), where **G** is the generator point for the group, G is order q number of points on the elliptic curve. An ECDSA signature for message M over a secret key SK is composed of a set of integers (SIG, r_x) in Z_q such that

$$SIG = \frac{H(M) + SK * r_x}{k}$$

Here r_x is the x coordinate of the elliptic curve point R=k * **G**. We use a setup protocol to construct a multiparty computational framework in which participating parties will obtain one time initialization of additive shares of signature by having secret key SK as input. The setup protocol used here is Doerner with small changes to ascertain the security against fraudulent participants. On termination of setup protocol, all the n participating parties will receive a point on a polynomial of degree (t-1). As per Shamir's scheme the y intercept form of this polynomial is the secret key SK [15] [16]. Therefore a group of X parties will obtain an additive sharing of SK using Lagrange coefficients. The input to our signing protocol generated additive sharing. On the other hand the signing protocol deviates from Doerner. This is explained in below three phases[17].

1. Multiplication and Inversion: Initially a group of P parties will agree to sign a message such that $|P = t|$, a inverse sampling protocol is used by them to sample k. From this every participating party will get an additive share of k and value $R{=}K{*}G$. Then they use GMW multiplication protocol to compute SK from their additive sharing of 1/k and sk.

2. Consistency Check: In order to ensure that correct and consistent inputs were used in previous phase the participating parties will use PK=SK * G and R=$k{*}G$. The consistency check employed here is similar to Doerner but it works in distributed fashion. All the parties in broadcast use a set of values whose cumulative values is equal to predictable target if all the participating parties have used GMW style input multiplier that are consistent with the outputs of the inverse-sampling protocol.

3. Signing: With consistency verification of previous steps each party i in group of participants P is assured that for some value k he holds v_i, w_i, and R.

$$\sum_{i \in P} v_i = \frac{1}{2} \qquad \sum_{i \in P} w_i = \frac{SK}{k} \qquad and \qquad R{=}k.G$$

The participating parties locally compute their signature share and broadcast them $SIG_i := v_i * Hash(msg) + w_i{*}r_x$. Finally signature is reconstructed and verified using the standard verification algorithm $SIG := \sum_{i \in P} SIG_i$.

TABLE III
ECDSA: 5 of 7 threshold signing algorithm

Message: International Association of Engineers
Sharing scheme: Any 5 from 7
Random secret:=(bd1d82a745196ce6ef7c89956b 24cd029d624a3f7a53145f3caa7d34e34fbe1c)
Public key: (108093126856915364035593825 37383200849113127988492658392211848346931320080 82 14719479315552741770347880327176071271819192031 1 787075480448989894736359903110 0)
Share 1:=000000054dfdbebf091a3217d6dd6d7d 2b30bec0383df962126b5ebc4e6907b30ec7b3b6
Share 2:=00000001c2deaf8f221d96d1113eaa9 6b90647524a29a65287667908005accc89417a01f
Share 3:=0000000215aa5a11e2be7cb33b50dc 4f419481b76149fc371e387d5be84cfcd77ff0a957
Share 4:=0000000382785a618d05299a3993052 deb8afd6617e97606ef38faa9a466aed8c0cf9b35
Share 5:=00000004eca789000444b1a8d5a329e 8ccf5a23cccc5386eab73ef2bd5cfad5ca199c106
Overall signature:(114060822406062602393922600 3241949680428663975637945673753110735032146110 51 483 10962678976898372603388693210098730272562890 1 03537477589345663761984465244 94)
Signature Verified: True !!!

IV. Performance Analysis

In this section, compute bitcoin and ethereum private keys that were generated due to biased nonces, by applying crypt analytics attack in opposition to digital signatures discovered in public block chains. We apply lattice based algorithms to solve to unhide ECDSA private keys that were generated using biased nonces. Both bitcoin and ethereum use elliptic curve SECP256k1. The elementary attack done on ECDSA crypto currencies are based on following vulnerabilities on already generated cryptocurrencies in wild,

1) Leak of nonce used to generate the signature
2) Use of same nonce to sign different messages
3) Implicit prefixes and implicit suffixes in nonce

Bitcoin data collection:
Bitcoin: The modified official client was used to collect bitcoin signatures and hash values. We used the instance of block chain from april 07 2022. At this instance block chain had 11,795,065,208 signatures from 31,644,506,974 distinct keys. Out of these keys 83,794,257 has been used to generate more than one signature. 27% of signatures in our instance are generated by one of these keys.
Crypt analysis test beds for biased nonces: Public key clusters were formed by eliminating duplicated signatures, i.e., keys that had same signature (r, s) and hash h. Following randomized tests were ran on the signatures for the keys that had $m > 1$ distinct signatures. The selected parameters are chosen so that the computational times for tests are reasonable for most common key.
- Verify if this key has generated a set of distinct signatures that has redundant r values. Then compute the private key and required signature nonces using case 1 and case 2.1.
- Randomly select any two signatures and verify length of nonces less than 128 bits and test is repeated for 2m times for every key.
- Randomly select any four signatures and verify the length of nonces less than 172 bits and test is repeated for 2m times for every key.
- Randomly select any four signatures and verify nonces sharing 172 least significant bits and test is repeated for 2m times for every key.
- Randomly select any two signatures generated from the pair (r,s) and verify nonces sharing 128MSB and test is repeated for 2m times for every key.

Implementing the cryptanalysis: All the test cases stated were implemented on sage using built in BKZ library for lattice based reduction. The computations were for heterogeneous cluster on Raspberry pi. The computations ended by running twice, firstly without signature normalization on a block chain from April 2022 and secondly with signature normalization on a block chain from may 2022. The computational bottleneck we found was intensive elliptic curve multiplications needed to ascertain whether the private key found is correct. The cumulative running time for both jobs was 15 days where a a single key could generate 1,021,572 signatures in April 2022. After applying elementary attacks we could compute 43 private keys were compromised via nonces with shared suffixes / prefix and small nonces. We encountered 132 signatures with these vulnerable nonces block chain. The figure 10 and 11 shows signature from biased and repeated nonces over period of time, large circles denote more signatures over the years. Most of the compromised nonces fell into different classes depending on length of variable portion of nonce. We obtained shorter nonces of length 32 bits, 64 bits, 110 bits, 128 bits and 160 bits and some nonces that shared suffix and prefix of 64 bit variable length.

Ethereum data collection: Querying was done on local Ethereum node via and RPC interface to collect the Ethereum signatures. Analysis is done on ethereum block chain instance on 14 May 2022, it had 143,457,786 distinct public keys that generated 831,811,259 signatures. Totally 31,855,806 keys had generated multiple signature, resulting in 592,229,478 signatures from such keys.
Crypt analysis test beds for biased nonces: Public key signature clusters were formed and verified that the keys had generated more than one signature. Same test applied for bitcoin were applied, computations were done twice, firstly with normalized signatures from April 2022 block chain instance and secondly without the May 2018 block chain instance. The 296 private keys were responsible for generating repeated signatures nonces and they used 0x7fffffffffffffffffffffffffffffff5d576e7357a4501ddfe92f46681b20

a0, which is $((n-1)/2)$ where n represents the order of SECP56k1 curve.

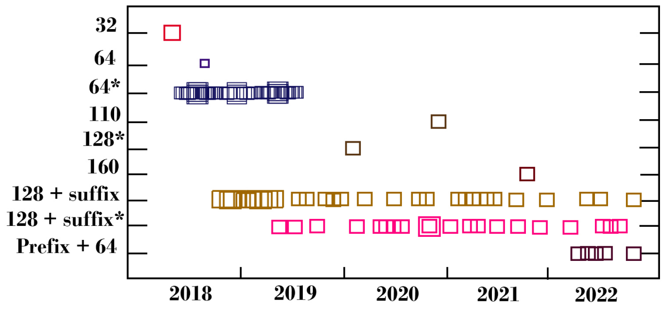

Fig 10. Bit coin signatures with biased and small nonces over the years.

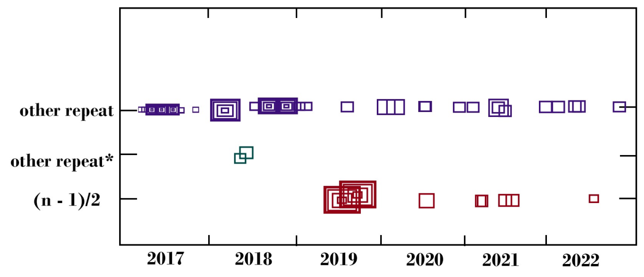

Fig 11.Bitcoin signatures with repeated nonces over time.

We extend our analysis to compute execution time of algorithm to crack ECDSA by using NIST and SECP curves. The reliability of elliptic curve cryptography depends on hardness in computing elliptic curve discrete logarithm problem (ECDLP). Edward Yin claims there are no algorithms to solve ECDLP [17] and if there are large number of ECC keys then a brute force technique is not suitable to the ECC algorithm, our work contradicts his statements. We have examined ECDSA security using NIST and SECP curves. The security of ECDSA was examined on three dimensions the rigidity of curve, safety level curve, and security against multiplicative transfer. If curve succeeds to resist all of the above methods then we consider the curve to be safe else we term it as unsafe curves. With our work we claim bitcoin curve SECP256k1 fails to resist leak-weak nonce attack and LLL attack using private keys,nonces and signatures of varying bit sizes of length 64, 128, 256, 512, and 1024-bits. The results clearly demonstrate time to crack ECDSA grows exponentially as bit sizes grow and cracking ECDSA seems to be more feasible using leak-weak, shared nonces than unlike LLL. Table V, VI, VII, shows average time to crack ECDSA with leak-weak nonce and Table VIII, IX shows average time to crack ECDSA using LLL algorithm using multiple signatures. Fig. 12 shows security of ECDSA with leak-weak nonces and LLL with NIST and SECP curves.

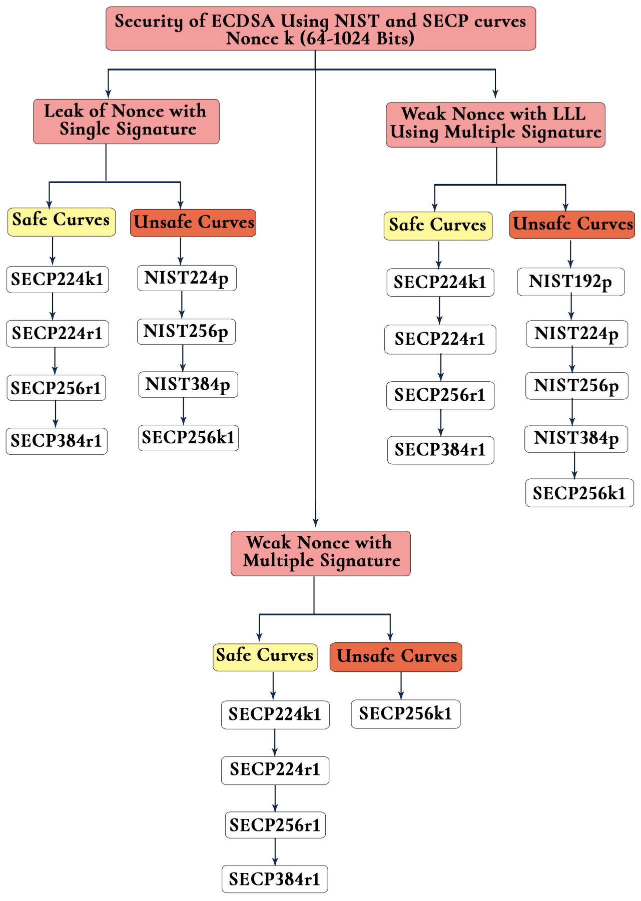

Fig 12. Security of ECDSA with leak-weak nonces and LLL with NIST and SECP curves

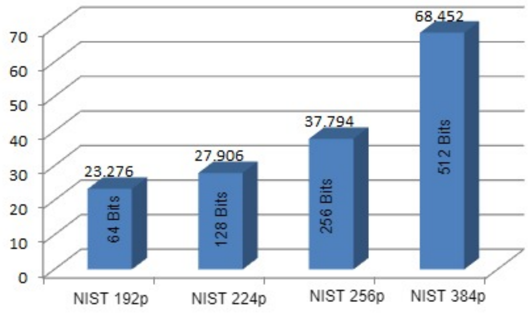

Fig 13. CRACK ECDSA BY WEAK NONCE FROM MULTIPLE SIGNATURES ON NIST CURVES USING LLL

TABLE IV
Compromised signatures and keys are classified based on type of nonce vulnerability that led to compromised private key. Most of the compromised keys were found to be of multi signature address.

Type of Nonce (Bits)	Multi Signature Keys	Single Signature Keys	Unique Keys
≤ 32	0	3	1
64	11	17	23
110	6	18	0
128	1	19	1
160	17	13	2
128 + Suffix	1	5	12
Prefix + 64	0	2	23

TABLE V
NODES FEATURE USED IN IMPLEMENTATION

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 VI
DISCLOSE ECDSA KEY FROM LEAK OF NONCE FOR MULTIPLE SIGNATURE ON NIST CURVES

Private Key	**64 Bits**	**128 Bits**	**256 Bits**	**512 Bits**	**1024 Bits**	
Nonce k	**64 Bits**	**128 Bits**	**256 Bits**	**512 Bits**	**1024 Bits**	
No of Signatures	**100**	**100**	**100**	**100**	**100**	**AVG**
NIST 192p	9.37	10.01	10.63	11.09	11.32	10.48
NIST 224p	9.49	10.28	10.91	11.66	11.63	10.79
NIST 256p	11.45	10.52	10.96	12.41	13.21	11.71
NIST 384p	11.63	11.51	11.85	13.87	14.49	12.67
NIST 521p	11.81	11.99	13.17	15.68	19.29	14.39

TABLE VII
DISCLOSE ECDSA KEY FROM LEAK OF NONCE FOR MULTIPLE SIGNATURE ON SECP CURVES

Private Key	**64 Bits**	**128 Bits**	**256 Bits**	**512 Bits**	**1024 Bits**	
Nonce k	**64 Bits**	**128 Bits**	**256 Bits**	**512 Bits**	**1024 Bits**	
No of Signatures	**200**	**200**	**200**	**200**	**200**	**AVG**
SECP224k1	SAFE	SAFE	SAFE	SAFE	SAFE	NA
SECP224r1	SAFE	SAFE	SAFE	SAFE	SAFE	NA
SECP256k1	16.62	16.85	17.29	18.31	18.84	17.58
SECP256r1	SAFE	SAFE	SAFE	SAFE	SAFE	NA
SECP384r1	SAFE	SAFE	SAFE	SAFE	SAFE	NA

V. Conclusions

In this paper, we compute Bitcoin and Ethereum private keys by performing cryptanalytic attacks on digital signatures available in public block chains and we analyze curves recommended by various standards. Each curve applied on ECDSA algorithm can be cracked in two ways, firstly with leaked nonce and secondly by performing lattice attacks using Lenstra-Lenstra-Lovasz (LLL) algorithm on bad nonces that were generated using faulty random number generator. From comparative table its clear that the computation times taken by each curves for both the cases. From this result analysis, it is deducted that the computation times to crack ECDSA using curves increases as the field size increases. Due to this fragility, we advise using EdDSA, in which nonces are created securely without the need of a random number generator. To combat side channel attacks, NIST has also standardised the usage of EdDSA with Curve- 25519. Applying ECDSA should be done with care, so that nonces used for ECDSA signatures are not redundant and confidential. Also generated safely. Finally we come to a conclusion that Elliptic Curve Cryptography using the the NIST 192p,NIST 224p,NIST 256p,NIST 384p,NIST 521p and SECP256k1 curves over leak-weak nonce are not safe for the transactions that are confidential and are to be kept secured down the line.

Appendix

Table IX - shows ECDSA: Disclosing the private key if nonce is known using NIST-256P recommended parameters Table X- shows ECDSA: disclosing the private key if nonce is known using NIST-521P recommended parameters.

TABLE VIII
DISCLOSE ECDSA KEY FROM WEAK NONCE (SAME) FROM MULTIPLE SIGNATURES ON SECP CURVES

Private Key	**64 Bits**	**128 Bits**	**256 Bits**	**512 Bits**	**1024 Bits**	
Nonce k	**64 Bits**	**128 Bits**	**256 Bits**	**512 Bits**	**1024 Bits**	
No of Signatures	**200**	**200**	**200**	**200**	**200**	**AVG**
SECP224k1	SAFE	SAFE	SAFE	SAFE	SAFE	NA
SECP224r1	SAFE	SAFE	SAFE	SAFE	SAFE	NA
SECP256k1	12.55	12.58	12.61	12.92	13.66	12.86
SECP256r1	SAFE	SAFE	SAFE	SAFE	SAFE	NA
SECP384r1	SAFE	SAFE	SAFE	SAFE	SAFE	NA

TABLE IX
DISCLOSE ECDSA KEY FROM WEAK NONCE WITH MULTIPLE NONCES ON NIST CURVES USING LLL

Private Key	**64 Bits**	**128 Bits**	**256 Bits**	**512 Bits**	**1024 Bits**	
Nonce k1	**64 Bits**	**128 Bits**	**256 Bits**	**512 Bits**	**1024 Bits**	
Nonce k2	**64 Bits**	**128 Bits**	**256 Bits**	**512 Bits**	**1024 Bits**	
No of Signatures	**50**	**50**	**50**	**50**	**50**	**AVG**
NIST 192p	22.93	24.59	22.72	22.98	23.16	23.28
NIST 224p	27.56	29.05	26.24	29.39	27.29	27.91
NIST 256p	37.77	36.91	38.77	35.38	40.14	37.79
NIST 384p	66.14	66.86	68.11	70.91	70.24	68.45

TABLE X
DISCLOSE ECDSA KEY FROM WEAK NONCE WITH SINGLE NONCE ON SECP CURVES USING LLL

Private Key	**64 Bits**	**128 Bits**	**256 Bits**	**512 Bits**	**1024 Bits**	
Nonce k	**64 Bits**	**128 Bits**	**256 Bits**	**512 Bits**	**1024 Bits**	
No of Signatures	**50**	**50**	**50**	**50**	**50**	**AVG**
SECP224k1	SAFE	SAFE	SAFE	SAFE	SAFE	NA
SECP224r1	SAFE	SAFE	SAFE	SAFE	SAFE	NA
SECP256k1	31.42	32.59	33.80	34.84	35.28	33.58
SECP256r1	SAFE	SAFE	SAFE	SAFE	SAFE	NA
SECP384r1	SAFE	SAFE	SAFE	SAFE	SAFE	NA

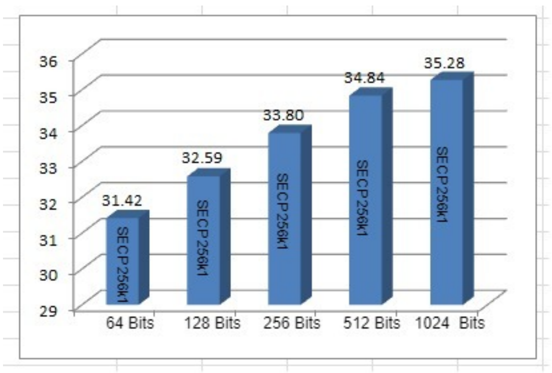

Fig 14. CRACK ECDSA BY WEAK NONCE FROM MULTIPLE SIGNATURES ON SECP 256k1 CURVE USING LLL

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 $\vec{v_1}^*$

$\vec{v_1} = (201, 37), \vec{v_2} = (1648, 297) and \vec{v_1}^* = (201, 37)$
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 $\vec{v_2^*} = (1.133, -6.155)$

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

$\vec{v_1} = (201, 37), \vec{v_2} = (40, 1)$
$\vec{v_1^*} = (201, 37)$ 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.

TABLE XI
ECDSA: Disclosing the private key, if nonce known (SECP256k1 Recommended Parameters)

CurveFp= p=115792089237316195423570985008687907853269984665640564039457584007908834671663
a=0
b=7
h=1
Order: 115792089237316195423570985008687907852837564279074904382605163141518161494337
Gx=55066263022277343669578718895168534326250603453777594175500187360389116729240
Gy=32670510020758816978083085130507043184471273380659243275938904335757337482424
Message 1: The International Association of Engineers (IAENG)
Sig 1(R,S):1090178978484625472626051078590318298857972588647162614981421996289387993138918265459205352446360663392410240856847177744969239952673627391122796700593599
Random value (k): 48559132611523869537634198757975587885
Private Key:53144400248017703868307005160578004559346968752187515812330620661442985198127
The private key is found:53144400248017703868307005160578004559346968752187515812330620661442985198127

TABLE XII
ECDSA: Disclosing the private key, if nonce known (NIST-521p Recommended Parameters)

Message 1: The International Association of Engineers (IAENG)
Sig1(R,S):2128226067352806767965385148949322423874778993494995375421389696415739940087523358653739666503930184035865240267411738520465568274624818894669233688372748609 663438386533491789268917133775138412649914951989874801838592660920313316815480643465176261181068257098030508137971670535782630373768587344613216642037803918
Random value (k):82964491372644003424833079795788819303
Private Key:5244497444874146855767107190613635184378638935505090001942679639088308352332600692201269707095795694552911474820787539810143490829397335285433794739421013412
Private Key:5244497444874146855767107190613635184378638935505090001942679639088308352332600692201269707095795694552911474820787539810143490829397335285433794739421013412

$||\vec{v_1^*}^2|| = 41770$, $||\vec{v_2^*}^2|| = 39.16$

$$\mu_{2,1} = \left(\frac{(40,1)\cdot(201,37)}{(201,37)\cdot(201,37)} = 0.193\right)$$

$$\left(\frac{3}{4} - \mu_{2,1}^2 \approx 0.713\right)$$

So, $||\vec{v_2^*}||^2 \ngeq \left(\frac{3}{4} - \mu_{2,1}^2\right)||\vec{v_1^*}||^2$ and we should swap, making $\vec{v_1}$= (40,1) and $\vec{v_2}$=(201,37)

Step 2:
We have: $\vec{v_1} = (40,1), \vec{v_2} = (201,37)$ and $\vec{v_1^*} = (40,1)$,

TABLE XIII
ECDSA: 2 of 3 threshold signing algorithm

Message: Hello
Sharing scheme: Any 2 from 3
Random secret:=(409dad2ef3aa49b1d5c352f0bd89e8a48f91f98e5abbab6e711367efdb42b6e9)
Public key: (10675422765967490806456867177375358912119726959856519695817415874045977056690 83967447299151140641130112076400326354735315602294989779527406731720970330161)
Share 1:=00000001b0f8d63a0ee0470a9cf8a456edfe4bb2e190be7f19e6a10f847e531accdb7ecc
Share 2: =000000022153ff452a164463642df5bd1e72aec278e0a68929c8f674d816dfb8ee3e056e
Overall signature:(21537892111068306770159239146372506631861916523671677860452903162171928870792 12259516882648318079446047637950938085779642468911292651091164123787383339954)
Signature Verified: True !!!

Now apply the Gram-Schmidt reduction, using $\vec{v_1^*} = \vec{v_1}$

$$\vec{v_2} = (201,37) - \frac{(201,37)\cdot(40,1)}{(40,1)\cdot(401,1)}(40,1) \approx (-0.799, 31.956)$$

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 $\vec{v_2}$

$$\vec{v_2} = (201,37) - \lfloor\frac{(201,37)\cdot(40,1)}{(40,1)\cdot(401,1)}\rceil(40,1) = (1,32)$$

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, $||\vec{v_2^*}||^2 \ngeq \left(\frac{3}{4} - \mu_{2,1}^2\right)||\vec{v_1^*}||^2$ 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 $\vec{v_1^*} = (1,32)$,
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:
$\vec{v_1} = (1,32), \vec{v_2} = (40,1)$ $\vec{v_1^*} = (1,32)$ and $\vec{v_2^*} = (39.93, -1.25)$
Using $\vec{v_1}$ to reduce $\vec{v_2}$

$$\vec{v_2} = (40,1) - \lfloor\frac{(40,1)\cdot(1,32)}{(1,32)\cdot(1,32)}\rceil(1,32)$$

$\vec{v_2}$=(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, $||\vec{v_2^*}||^2 \geq \left(\frac{3}{4} - \mu_{2,1}^2\right)||\vec{v_1^*}||^2$
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.

References

[1] Chintan Patel, Nishant Doshi, “Secure Light Weight Key Exchange Using ECC For User Gateway Paradigm”, IEEE Transactions on Computer, Volume: 70, Issue: 11, Pages 1789 - 1803,2021

[2] Cherkaoui “ Diffie-Hellman Multi-Challenge using a New Lossy Trapdoor Function Construction”, IAENG International Journal of Applied Mathematics Open Access, Volume 51, Issue 3,Pages: 736-742, 2021.

[3] Kuanyun Zhu, Jingru Wang, and Yongwei Yang “Lattices of (Generalized) Fuzzy Ideals in Residuated Lattices”, IAENG International Journal of Applied Mathematics, Open Access, Volume 50, Issue 3, Pages: 505-511, 2020.

[4] Zhang And Xuesong Wang ”Digital Image Encryption Algorithm Based on Elliptic Curve Public Cryptosystem”, IEEE Access, Volume: 6,Issue 2, Pages: 70025 - 70034, 2018”.

[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 Open Access, Volume : 8, Pages: 52018 – 52027, 2020.

[6] Nizar Ouni and Ridha Bouallegue “Performance And Complexity Analysis of Reduced Iterations LLL Algorithm”, International Journal of Computer Networks Communications (IJCNC),Volume:8, Issue 3, Pages:09-27 , 2016.

[7] 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, Volume:20, Issue 2,Pages: 107–121, 2016.

[8] Michael Brengel and Christian Rossow “ Identifying Key Leakage of Bitcoin Users”, International Symposium on Research in Attacks, Intrusions, and Defenses Open Access, Volume: 11050, Pages: 623–643 2018.

[9] Mohammed Mujeer Ulla and Deepak S. Sakkari, “Application of Elliptic Curve Crypto System to Secure Multi-Signature Bitcoin Block Chain”, Journal of Computer Science, 2022 Open Access,Volume 19,Issue: 1, Pages:112-125, 2023.

[10] Joachim Breitner and Nadia Heninger, “Biased Nonce Sense: Lattice Attacks against Weak ECDSA Signatures in Cryptocurrencies”, Lecture Notes in Computer Science Springer International Publishing - Financial Cryptography and Data Security,Volume: 11598, Pages:3-20, 2019.

[11] Jack Doerner, Yashvanth Kondi, Eysa Lee and abhi shelat, “Threshold ECDSA from ECDSA Assumptions:The Multiparty Case”, IEEE Symposium on Security and Privacy, Pages:1051-1066, 2019.

[12] 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”, IEEE International Conference on Microwaves, Antennas, Communications and Electronic Systems,Pages: 1-4 2018.

[13] Shen Guicheng, Yu Zhen “Application of Elliptic Curve Cryptography in Node Authentication of Internet of Things”,Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing,IEEE, Pages: 452-455,2013.

[14] Ashwitha Naikoti and Ravi Kishore Kodali ”ECDH based Security Model for IoT using ESP 8266” International Conference on Control, Instrumentation, Communication and Computational Technologies,IEEE,Pages:629-633, 2016.

[15] Deepak S. Sakkari Mohammed Mujeer ulla “Review on Insight into Elliptic Curve Cryptography”, Modern Approaches in Machine Learning Cognitive Science: A Walkthrough (LNCS),Springer, Volume 1027,Pages:81–93, 2022.

[16] Deepak S. Sakkari Mohammed Mujeer ulla “Design and Implementation of Identifying Points on Elliptic Curve Efficiently Using Java”, Modern Approaches in Machine Learning Cognitive Science: A Walkthrough, Springer (LNCS),Volume 1027,Pages:95–105, 2022

[17] 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” , Innovation in Electrical Power Engineering, Communication, and Computing Technology(LNEE), Springer, Volume 814,Pages:323–337,2021.