ECDSA Cracking Methods

William J. Buchanan[1], Jamie Gilchrist[1], and Keir Finlow-Bates[2]

Blockpass ID Lab, Edinburgh Napier University, Edinburgh.

Abstract. The ECDSA (Elliptic Curve Digital Signature Algorithm) is used in many blockchain networks for digital signatures. This includes the Bitcoin and the Ethereum blockchains. While it has good performance levels and as strong current security, it should be handled with care. This care typically relates to the usage of the nonce value which is used to create the signature. This paper outlines the methods that can be used to break ECDSA signatures, including revealed nonces, weak nonce choice, nonce reuse, two keys and shared nonces, and fault attack.

1 Introduction

ECDSA has been around for over two decades and was first proposed in [1]. The ECDSA method significantly improved the performance of signing messages than the RSA-based DSA method. Its usage of elliptic curve methods speeded up the whole process and supported much smaller key sizes. In 2009, Satoshi Nakamoto selected it for the Bitcoin protocol, and it has since been adopted into Ethereum and many other blockchain methods. This paper provides a review of the most well-known methods of breaking ECDSA.

2 Basics of Elliptic Curve Cryptography

One of the most basic forms of elliptic curves is:

$$y^2 = x^3 + ax + b \pmod{p} \tag{1}$$

and where the elliptic curve is defined with p, a, b , g_x, g_y, and n, and where (g_x,g_y) is a base point on our curve, and n is the order of the curve.

With ECDSA, the curve used in Bitcoin and Ethereum is secp256k1, and which has the form of:

$$y^2 = x^3 + 7 \pmod{p} \tag{2}$$

and where $p = 2^{256} - 2^{32} - 977$. We can also use the NIST P256 (secp256r1) curve or the NIST-defined P521 curve.

ECDSA signatures are non-deterministic and will change each time based on the nonce used. For the public keys, we have an (x, y) point on the curve. This thus has 512 bits (for secp256k1), and where the private key is a 256-bit scalar value.

3 Creating the ECDSA signature

An outline of ECDSA is shown in Figure 1. With our curve, we have a generator point of G and an order n. We start by generating a private key (d) and then generate the public key of:

$$Q = d.G \tag{3}$$

The public key is a point on the curve, and where it is derived from adding the point G, d times.

3.1 Signing the message

With a message (m), we aim to apply the private key and then create a signature (r, s). First, we create a random nonce value (k) and then determine the point:

$$P = k.G \tag{4}$$

Next, we compute the x-axis point of this point:

$$r = P_x \pmod{n} \tag{5}$$

This gives us the r value of the signature. Next, we take the hash value of the message:

$$e = H(m) \tag{6}$$

And then compute the s value as:

$$s = k^{-1}.(e + d.r) \pmod{n} \tag{7}$$

3.2 Verifying the signature

We can verify by taking the message (m, the signature (r, s) and the public key (Q):

$$e = H(m) \tag{8}$$

Next, we compute:

$$w = s^{-1} \pmod{n} \tag{9}$$

$$u_1 = e.w \tag{10}$$

$$u_2 = r.w \tag{11}$$

We then compute the point:

$$X = u_1.G + u_2.Q \tag{12}$$

And then take the x-co-ordinate of X:

$$x = X_x \pmod{n} \tag{13}$$

If x is equal to r, the signature has been verified.

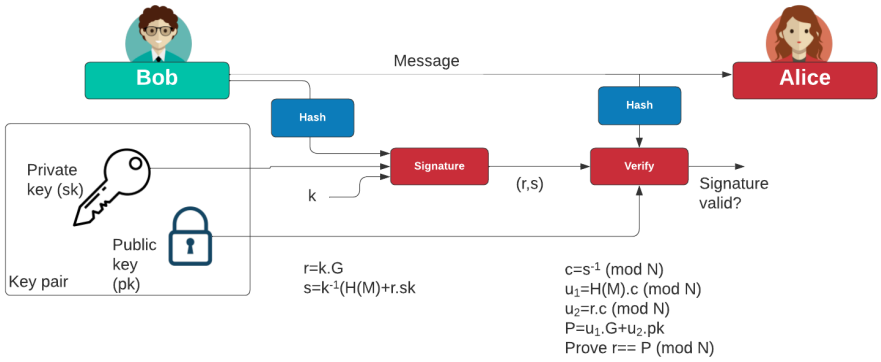

Fig. 1. ECDSA signature

4 ECDSA attacks

4.1 Revealed nonce

If the signer reveals just one nonce value by mistake, an intruder can discover the private key [2]:

$$priv = r^{-1} \times ((k \cdot s) - H(M)) \tag{14}$$

This works because:

$$s \cdot k = H(M) + r \cdot priv \tag{15}$$

and so:

$$r \cdot priv = s \cdot k - H(M) \tag{16}$$

and for $priv$:

$$priv = r^{-1}(s \cdot k - H(M)) \tag{17}$$

As r, s, $H(M)$, and k are known, $priv$ can therefore be calculated.

4.2 Weak nonce choice

A weak nonce can be broken with the Lenstra–Lenstra–Lovász (LLL) method [3], and where we crack the signature and discover the private key used to sign the message [4].

4.3 Nonce re-use

It is well-known that simply keeping the selected nonce secret is not enough to secure the private key [5]. If a nonce is used to sign a first message to produce a first signature (s_1) and is then reused to sign a second message to produce a second signature (s_2), then s_1 and s_2 will have the same r value, and it is possible to derive the private key ($priv$) from the two signatures.

In ECDSA, Bob creates a random private key ($priv$) and then a public key from [6]:

$$pub = priv \times G \tag{18}$$

Next, to create a signature for a message M_1, he creates a random number (k) and generates the signature from the SHA-256 hash of the message, $H(M_1)$, using the private key $priv$. We denote $H(M_1)$ as h_1.

$$r_1 = k \cdot G \tag{19}$$

$$s_1 = k^{-1}(H(M_1) + r \cdot priv) \tag{20}$$

The signature is then (r_1, s_1), where r_1 is the x-co-ordinate of the point kG.

Bob now signs a second message M_2 using the SHA-256 hash of the second message, $H(M_2)$, the same random number k to produce a second signature. We denote $H(M_2)$ as h_2.

$$r_2 = k \cdot G \tag{21}$$

$$s_2 = k^{-1}(H(M_1) + r \cdot priv) \tag{22}$$

Note that r_1 is equal to r_2 (see equations 19 and 21). In general, if two signatures generated using the same private key have the same r value, then the same nonce value has been used for each signature. This provides a quick way of checking whether the nonce reused attack applies.

We can then recover the private key with:

$$\frac{s_2 \cdot h_1 - s_1 \cdot h_2}{r(s_1 - s_2)} = \frac{h_1 \cdot h_2 + r \cdot h_1 \cdot priv - h_1 \cdot h_2 - r \cdot h_2 \cdot priv}{r \cdot h_1 \cdot r \cdot priv - r \cdot h_2 - r \cdot priv} \tag{23}$$

$$= \frac{r \cdot h_1 \cdot priv - r \cdot h_2 \cdot priv}{r \cdot h_1 - r \cdot h_2} \tag{24}$$

$$= priv \tag{25}$$

We can also recover the nonce with:

$$\frac{h_1 - h_2}{s_1 - s_2} = \frac{h_1 - h_2}{k^{-1}(h_1 - h_2 + priv(r - r))} = k \tag{26}$$

4.4 Two keys and shared nonces

With an ECDSA signature, we sign a message with a private key ($priv$) and prove the signature with the public key (pub). A random value (a nonce) is then used to randomize the signature. Each time we sign, we create a random nonce value, which will produce a different (but verifiable) signature. The private key, though, can be discovered if Alice signs four messages with two keys and two nonces [5]. In this case, she will sign message 1 with the first private key (x_1), sign message 2 with a second private key (x_2), sign message 3 with first private key (x_1) and sign message 4 with the second private key (x_2) The same nonce (k_1) is used in the signing for messages 1 and 2, and another nonce (k_2) is used in the signing of messages 3 and 4 [7].

In ECDSA, Bob creates a random private key ($priv$), and then a public key from:

$$pub = priv \cdot G \tag{27}$$

Next, in order to create a signature for a message of M, he creates a random number (k) and generates the signature of:

$$r = k \cdot G \tag{28}$$

$$s = k^{-1}(H(M) + r \cdot priv) \tag{29}$$

The signature is then (r, s) and where r is the x-co-ordinate of the point kG. $H(M)$ is the SHA-256 hash of the message (M), and converted into an integer value. In this case, Alice will have two key pairs and two private keys (x_1 and x_2). She will sign message 1 (m_1) with the first private key (x_1), sign message 2 (m_2) with a second private key (x_2), sign message 3 (m_3) with the first private key (x_1) and sign message 4 (m_4) with the second private key (x_2). The same nonce (k_1) is used in the signing of messages 1 and 2, and another nonce (k_2) is used in the signing of messages 3 and 4. Now let's say we have four messages (m_1 .. m_4) and have hashes of:

$$h_1 = H(m_1) \tag{30}$$

$$h_2 = H(m_2) \tag{31}$$

$$h_3 = H(m_3) \tag{32}$$

$$h_4 = H(m_4) \tag{33}$$

The signatures for the messages will then be (s_1, r_1), (s_2, r_1), (s_3, r_2), and (s_4, r_2):

$$s_1 = {k_1}^{-1}(h_1 + r_1 \cdot x_1) \tag{34}$$

$$s_2 = {k_1}^{-1}(h_2 + r_1 \cdot x_2) \tag{35}$$

$$s_3 = {k_2}^{-1}(h_3 + r_2 \cdot x_1) \tag{36}$$

$$s_4 = {k_2}^{-1}(h_4 + r_2 \cdot x_2) \tag{37}$$

Using Gaussian elimination, we can also recover the private keys with:

$$x_1 = \frac{h_1 r_2 s_2 s_3 - h_2 r_2 s_1 s_3 - h_3 r_1 s_1 s_4 + h_4 r_1 s_1 s_3}{r_1 r_2 (s_1 s_4 - s_2 s_3)} \tag{38}$$

and:

$$x_2 = \frac{h_1 r_2 s_2 s_4 - h_2 r_2 s_1 s_4 - h_3 r_1 s_2 s_4 + h_4 r_1 s_2 s_3}{r_1 r_2 (s_2 s_3 - s_1 s_4)} \tag{39}$$

4.5 Fault Attack

In the case of a fault attack in ECDSA, we only require two signatures. One is produced without a fault (r, s), and the other has a fault (r_f, s_f). From these, we can generate the private key [8,9].

In ECDSA, Bob creates a random private key ($priv$), and then a public key from [10]:

$$pub = priv \cdot G \tag{40}$$

Next, in order to create a signature for a message of M, he creates a random number (k) and generates the signature of:

$$r = k \cdot G \tag{41}$$

$$s = k^{-1}(h + r \cdot d) \tag{42}$$

and where d is the private key and $h = H(M)$ The signature is then (r, s) and where r is the x-co-ordinate of the point kG. h is the SHA-256 hash of the message (M), and converted into an integer value.

Now, let's say we have two signatures. One has a fault and the other one is valid. We then have (r, s) for the valid one, and (r_f, s_f) for the fault. These will be:

$$s_f = k^{-1} \cdot (h + d \cdot r_f) \tag{43}$$

$$s = k^{-1} \cdot (h + d \cdot r) \tag{44}$$

and where h is the hash of the message. Now, if we subtract the two s values, we get:

$$s - s_f = k^{-1} \cdot (h + d \cdot r) - k^{-1} \cdot (h + d \cdot r_f) \tag{45}$$

Then:

$$s - s_f = k^{-1} \cdot (d \cdot r - d \cdot r_f) \tag{46}$$

$$k \cdot (s - s_f) = (d \cdot r - d \cdot r_f) \tag{47}$$

$$k = (d \cdot r - d \cdot r_f) \cdot (s - s_f)^{-1} \tag{48}$$

This can then be substituted in:

$$s = k^{-1}(h + r \cdot d) \tag{49}$$

This gives:

$$s = (s - s_f) \cdot (d \cdot r - d \cdot r_f)^{-1} \cdot (h + d \cdot r) \tag{50}$$

$$s \cdot (d \cdot r - d \cdot r_f) = (s - s_f) \cdot (h + d \cdot r) \tag{51}$$

$$s \cdot d \cdot r - s \cdot d \cdot r_f = s \cdot h + s \cdot d \cdot r - h \cdot s_f - d \cdot r \cdot s_f \tag{52}$$

$$-s \cdot d \cdot r_f = s \cdot h - h \cdot s_f - d \cdot r \cdot s_f \tag{53}$$

$$d \cdot r \cdot s_f - s \cdot d \cdot r_f = s \cdot h - h \cdot s_f \tag{54}$$

$$d \cdot (r \cdot s_f - s \cdot r_f) = h \cdot (s - s_f) \tag{55}$$

We can then rearrange this to derive the private key (d) from:

$$d = h \cdot (s - s_f) \cdot (s_f \cdot r - s \cdot r_f)^{-1} \tag{56}$$

5 Conclusions

We can see that ECDSA needs to be handled carefully, especially when using the nonce value.

References

1. D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, pp. 36–63, 2001.
2. W. J. Buchanan, "Ecdsa: Revealing the private key, if nonce known (secp256k1)," https://asecuritysite.com/ecdsa/ecd3, Asecuritysite.com, 2025, accessed: April 09, 2025. [Online]. Available: https://asecuritysite.com/ecdsa/ecd3
3. A. K. Lenstra, H. W. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients," 1982.
4. W. J. Buchanan, "Ecdsa crack using the lenstra–lenstra–lovász (lll) method," https://asecuritysite.com/ecdsa/ecd, Asecuritysite.com, 2025, accessed: April 04, 2025. [Online]. Available: https://asecuritysite.com/ecdsa/ecd
5. M. Brengel and C. Rossow, "Identifying key leakage of bitcoin users," in *Research in Attacks, Intrusions, and Defenses: 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings 21*. Springer, 2018, pp. 623–643.
6. W. J. Buchanan, "Ecdsa: Revealing the private key, if nonce revealled (secp256k1)," https://asecuritysite.com/ecdsa/ecd5, Asecuritysite.com, 2025, accessed: April 09, 2025. [Online]. Available: https://asecuritysite.com/ecdsa/ecd5
7. ——, "Ecdsa: Revealing the private key, from two keys and shared nonces (secp256k1)," https://asecuritysite.com/ecdsa/ecd6, Asecuritysite.com, 2025, accessed: April 04, 2025. [Online]. Available: https://asecuritysite.com/ecdsa/ecd6

8. G. A. Sullivan, J. Sippe, N. Heninger, and E. Wustrow, "Open to a fault: On the passive compromise of {TLS} keys via transient errors," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 233–250.
9. D. Poddebniak, J. Somorovsky, S. Schinzel, M. Lochter, and P. Rösler, "Attacking deterministic signature schemes using fault attacks," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 338–352.
10. W. J. Buchanan, "Cracking rsa: Fault analysis," https://asecuritysite.com/rsa/rsa_fault, Asecuritysite.com, 2025, accessed: April 09, 2025. [Online]. Available: https://asecuritysite.com/rsa/rsa_fault