CVE-2022-21449: a vulnerability of an implementation of the Elliptic Curve Digital Signature Algorithm (ECDSA) by Oracle

Alexander Zinoni

April 2023

Page Count:
18

Université Grenoble Alpes

Contents

1 Abstract

The focus of this paper is on the Elliptic Curve Digital Signature Algorithm (ECDSA). The first part (section 2) of the document introduces concepts which are key to understanding how ECDSA works. The Second part (section 3) is about explaining what elliptic curves are and what operations we can do on them. Then, sections 4 and 5 introduce two algorithms, ECDSA and ECIES which use the operations presented in the previous section.
The final goal of the paper is to explain the vulnerability CVE 2022 21449. CVE 2022 21449 is a vulnerability in the implementation of the ECDSA algorithm by Oracle (the ECDSA algorithm is not vulnerable if well implemented). We will show how the vulnerability can be exploited and explain the maths behind it.

2 Important concepts

Before introducing Elleptic curves, it is important to introduce the notion of "difficult" computations. Simply put, a computation is considered to be difficult if it would take ages to compute, even with the most powerful machines that exist. Genreally, if a key holds the secret of an important message, it will be considered "difficult" to crack if by the time all possibilities are bruteforced, the message is not relevant anymore (the universe may collapse for example).

Here are some other concepts which are key to understanding how ECDSA works:

2.1 Symmetric vs asymmetric cryptography

2.1.1 Symmetric cryptography

Symmetric cryptography is one of the most ancient ciphering methods, it needs the same key to crypt and decrypt messages. For example, Julius Caesar invented the Caesar shift which is a symmetric encryption algorithm. To code messages, it takes every letter individually and shifts its position in the alphabet. The amount by which every letter is shifted is the key. To decrypt it, we shift the letters backwards according to the key. For example, with a key k = 1, the messages 'abc' becomes 'bcd'. Even though it was beliieved to be unbreakable at the time, the Caesar shift is not the most secure example of symmetric encryption (it has only 26 possibilities for the key). More recent examples include the Vigenère Cipher, Enigma or AES (Advanced Encryption Standard) which is used today.

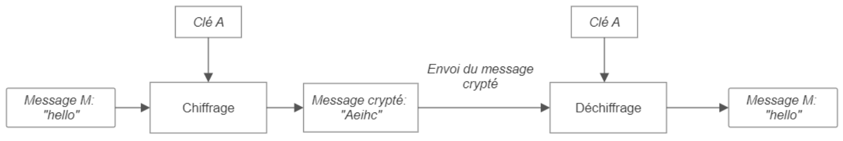

Figure 1: Caption

The advantage of symmetric encryption is that ciphering and deciphering processes require far less computations than for asymmetric methods. Unfortunately, establishing a key when communicating over distance can be complicated, how to make sure it won't be intercepted ? how to make sure only the receiver knows the key ? how to make sure the receiver isn't an impostor ?

2.1.2 Asymmetric cryptography

Asymmetric cryptography, unlike symmetric cryptography, needs a key pair, a public one, and a private one. The keys are mathematically related in such a way that when one key is used to encrypt a messages, only the other can decrypt it. Key pair are calculated thanks to one way functions. Such functions are easy to compute the output of, but incredibly difficult to reverse. For example, let p and q be two prime numbers. While it is very easy to compute their product e, it is much more difficult to retrieve the values of p and q only from e. In this simplified example, the pair (p, q) would be the private key and e the public one. This is exactly how to RSA (Rivest - Shamir - Adleman) algorithm works. Note that when p and q are large enough, retrieving their values when only knowing e takes an unreasonable amount of time.

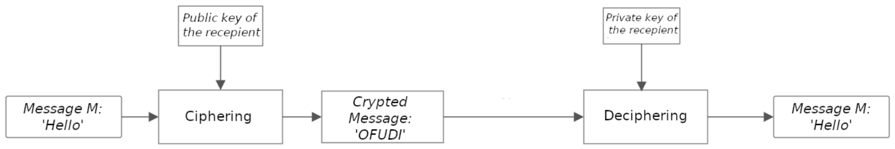

Figure 2: Caption

In practice, if I wanted to send a messages to my friend Ethan without letting anyone else read it, I would publish my public key (in such a way that everyone can see it). Ethan would grab it and use it to encrypt his messages and then send it back to me. Since I am the only one how to knows the private key, I am the only one who will be able to decrypt it. Note that it is crucial that the owner of the key pair keeps the private key to himself.

More importantly, asymmetric cryptography allows us to communicate over great distance, without having to exchange a secret key like symmetric cryptography.

2.2 Hashing functions

2.2.1 Properties of hashing functions

A hashing function is a map of an arbitrary binary string to a binary string with fixed size. The output of a hashing function is called a hash.

These are the characteristics of hashing functions:

- Determinist. The output will always be the same for a given input.
- Fast. For any input, the output is very easy to compute for machines.
- Non reversible. Unless brute force is used to try all inputs, it is not possible to reverse hashing functions.
- 'Avalanche Effect'. Two inputs which are ever so slightly different have drasticly different outputs.

Also, hashing functions follow these properties:

- Pre-Image Resistance. Given a hash h, it should be very difficult to find a message m such that $hash(m) = h$.
- Second Pre-Image Resistance. Given a message m_1, it should be very difficult to find a message m_2 such that $hash(m_1) = hash(m_2)$.

For example, the function SHA2-256 [6] takes any string as an input and outputs a 256 bit hash. Here is an example implemented in Python 3:

```
m = 'Hello :)'
m2 = 'hello :)'

print(m, 'becomes', hashlib.sha256(message.encode('utf-8')).hexdigest())
print(m2, 'becomes', hashlib.sha256(message.encode('utf-8')).hexdigest())
```

Output:

```
Hello :) becomes 2f0eb1859e295bcd183127558f3c205270e7a8004ad362e5123bd5b2774e0f9c

hello:) becomes f034f9986ae0fa2e3de3b40fcc378bacf6a5a01d269af841121501f610ddc65b
```

2.2.2 How do hashing functions work ?

SHA-1 is the first hashing function of the SHA family. It works in a very similar way as SHA-2. The first thing the algorithm does is to split the text input into blocks of 512 bits (padding is added to reach a message multiple of 512 bits) (step 1 on figure 2). The function has an initial state of length 160 bits, which is divided in five 32 bit strings. After a series of transformations, the initial state is transformed into a 160bit output (2).

Each 512 bit segment is divided again and is passed through a compression function with the initial state. After 80 rounds of compression (3), the output is sum0med with the initial state (4). It the 512 bit block was the last, then the sum becomes the result (the hash), otherwise, it becomes the initial state for the next 512 bit block.

2.3 Digital Signatures

The goal of digital signatures is to guarantee the:

- **Authenticity** of the sender, which means that the identity of the sender is verified.
- **Integrity** of the message, which means that the content of message has not been modified after it was signed.

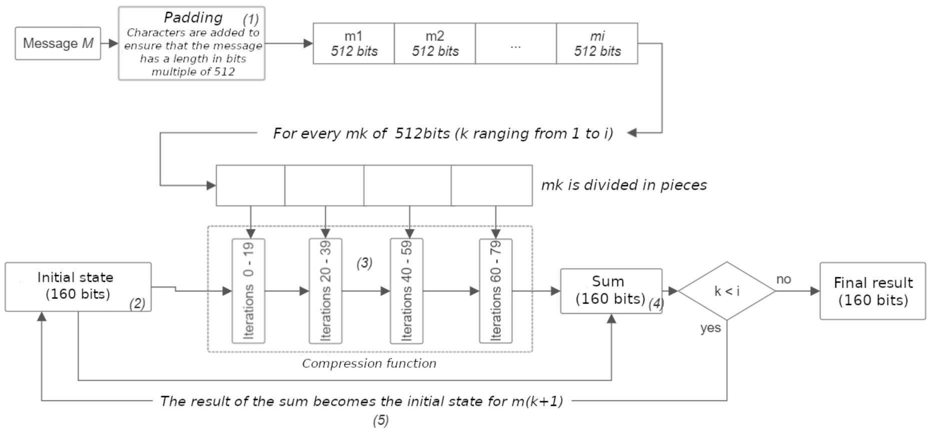

Figure 3: Hashing function diagram

Hashing functions as well as asymmetric cryptography are used to achieve these goals.

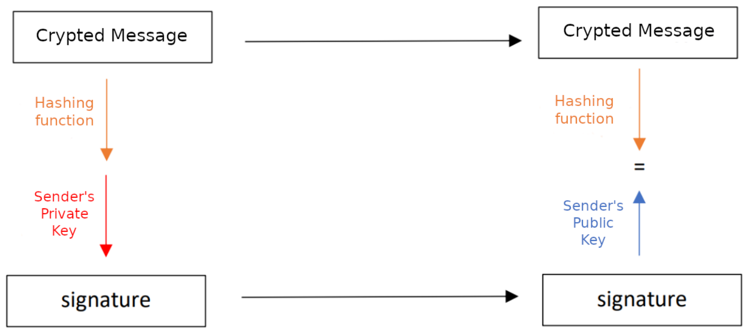

Figure 4: Digital Signatures [4]

In order to work, the encrypted message is sent alongside a copy of itself passed through a hashing function, then, the hash output is encrypted by the sender's private key. When the message is received, the encrypted message should be hashed and should be compared with the signature decrypted with the public key of the sender; if they match, the signature is verified.
Hashing the encrypted message when is sent and received makes sure of the integrity of the message. If the message was changed, the signature would not match after it was decrypted.
Encrypting the hash with the private key ensures that the sender is authenticated, because he is the only owner of the his private key.

3 Elliptic curves over $\mathbb{R}$

3.1 Generality, Definitions

Point at infinity: Let us define $\mathcal{O}$ the point "at infinity" which is the point at the end of every **vertical line** in a geometric space.

An elliptic curve is the set containing $\mathcal{O}$ and all points of coordinates x, y with two real parameters a, b such that $y^2 = x^3 + ax + b$ and $\delta = 4a^3 + 27b^3 \neq 0$, such curve is usually denoted $\mathcal{C}(a, b)$. Graphically, the condition $\delta = 4a^3 + 27b^3$ ensures that the curve is "nonsingular", meaning that at all points, the curve is smooth and has no cusps or angles.
$\forall a, b \in \mathbb{R}, 4a^3 + 27b^3 \neq 0,$
$\mathcal{C}(a, b) = \{(x, y) \in \mathbb{R}^2; y^2 = x^3 + ax + b\} \cup \mathcal{O}$

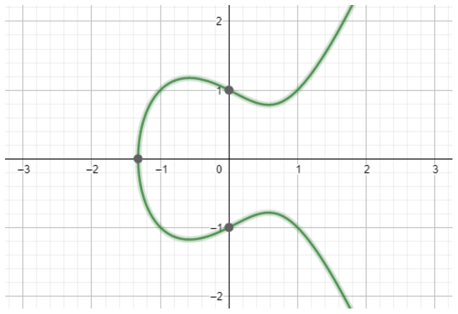

Figure 5: An elliptic curve with parameters a = -1 and b = 1

3.2 Lines passing through an elliptic curve

Each line passing through an elliptic curve $\mathcal{C}(a, b)$ intersects the curve at 3 points, at most. We will show that given two points P, Q having different x-coordinates implies that if the line PQ is not tangent to the curve $\mathcal{C}(a, b)$ then there is 3 intersection points between $(PQ) and \mathcal{C}(a, b)$

Proof. let $a, b \in \mathbb{R}$ such that $4a^3 + 27b^3 \ \neq 0$
Consider the elliptic curve $\mathcal{C}(a, b)$ on $\mathbb{R}$
Consider the points $P(x_p, y_p), Q(x_Q, y_Q)$ such that $x_p \neq x_Q$
Knowing the line PQ is linear, we can find an equation of it in the form
$y = \lambda x + \beta$
where λ is the **slope** of the line: $\lambda = \frac{\Delta_y}{\Delta_x}$
This means that, at any point of coordinates x, y, $\lambda = \frac{y - y_P}{x - x_P}$ **(i)**, and the slope is the same everywhere in a straight line
In particular at the point Q, the slope is equal to:
$\lambda = \frac{y_Q - y_P}{x_Q - x_P}$ **(ii)**
from **(i)**, we can obtain an equation of the line:
$\lambda = \frac{y - y_P}{x - x_P} \rightarrow (x - x_P)\lambda = y - y_P \rightarrow y = \lambda x - \lambda x_P + y_P$

we then obtain the linear equation for the line (PQ): $y = \lambda x + \beta$ where
$\lambda = \frac{y_Q - y_P}{x_Q - x_P}$ and $\beta = y_P - \lambda x_P$
Then for all point of intersection (x, y) between the curve $\mathcal{C} and (PQ)$,

- $R \in \mathcal{C} \to y^2 = x^3 + ax + b$ **(1)**
- $R \in (PQ) \to y = \lambda x + \beta$ **(2)**

From **(2)** we obtain: $y^2 = (\lambda x + \beta)^2$, which, from **(1)**, is also equal to
$x^3 + ax + b$
The following equation then holds: $(\lambda x + \beta)^2 = x^3 + ax + b$ **(E)**
(E) $\equiv x^3 - \lambda^2 x^2 - 2\lambda x \beta - \beta^2 + ax + b = 0$, this is a cubic equation since it is in the form $Ax^3 + Bx^2 + Cx + D = 0$ where $A \neq 0$, which has **Three real roots** (which can be distinct or not). And since the points P and Q are known intersection points between $\mathcal{C}, and (PQ)$, x_P, x_Q are then solutions to **(E)**
It is then possible to find the third solution to the equation using the sum of the roots of the equation, which states that <u>given a cubic equation $Ax^3 + Bx^2 + Cx + D = 0$ and three solutions $\alpha_1, \alpha_2, \alpha_3$, $(\alpha_1 + \alpha_2 + \alpha_3) = -B$ (the sum of the solutions is equal to the negation of the coefficient of x^2)</u>
$x_P + x_Q + x = \lambda^2 \to x = \lambda^2 - x_P - x_Q$
And by plotting this result into the slope equation, we get
$y = ((\lambda^2 - x_P - x_Q) - x_P)\lambda + y_P$
Thus, given an elliptic curve $\mathcal{C}(a, b)$, two points P, Q of the curve with different x-coordinates, There exists a third point belonging to both the curve and the line PQ with
$x_R = \lambda^2 - x_P - x_Q$, $y_R = (x_R - x_P)\lambda + y_P$ where:

- $\lambda = \frac{y_Q - y_P}{x_Q - x_P}$

□

3.3 Sum of two points

Given an elliptic curve C Let us define the sum of two points [5], G and $G1$, on the elliptic curve C.
First, take the line d defined by:

- if $G = G1$ then d is the **tangent** line to the curve at the point G
- if $G \neq G1$ then d is the line formed by $(GG1)$

Then define the point $G2'$ as follows:

- if $G = G1$ then $G2'$ is the second intersection between the tangent and the curve
- if $G \neq G1$ then $G2'$ is the third intersection between d and the curve (This point always exists as shown in (3.2) with known coordinates)

The sum $G2$ is then the opposite point of $G2'$ on the curve C. We have computed $G2 = G \oplus G1$ *(figure 6)*.

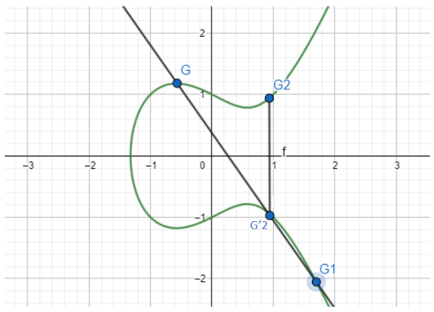

Figure 6: $G2 = G \oplus G1$

If G and G1 are the same point, the only difference with the previous computation is that the line d is the tangent of C at point G.

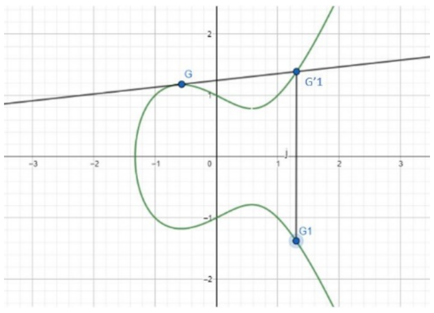

Figure 7: $G1 = G \oplus G$

3.4 Elliptic curves on a finite fields

3.4.1 Graphical representation

For cryptography use cases, elliptic curves are defined on finite fields $\mathbb{Z}/p\mathbb{Z}$ with p being a large prime number.
An elliptic curve on a finite field forms a mathematical group. The elements of the group are points on the curve and the operations defined are point summation and multiplication of points by scalars.

3.4.2 Point at infinity

The point at infinity $\mathcal{O}$ is a notation used when the sum of two points is said to be impossible. When two points with the same *x-value* are summed, there is no third intersection between their line and the curve (figure 9).

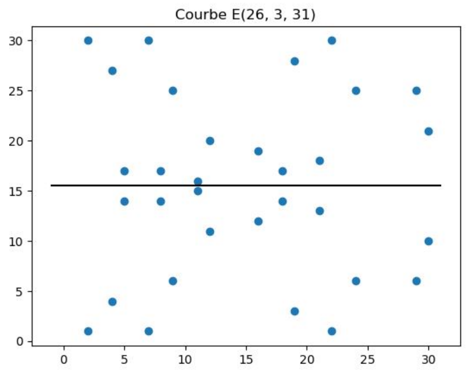

Figure 8: $E(26, 3, 31)$ $(a = 26, b = 3, p = 31)$

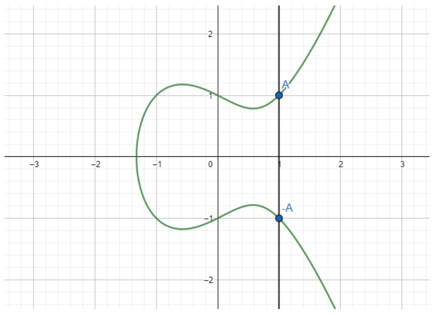

Figure 9: $A \oplus -A = \mathcal{O}$

The point at infinity has the following properties:

- For any point A we have:
$$0 \cdot A = \mathcal{O}$$
- The point at infinity is the identity element for summation:
$$A \oplus \mathcal{O} = A$$
$$\mathcal{O} \oplus \mathcal{O} = \mathcal{O}$$

3.4.3 Subgroups and their order

By looking at the multiples of point (3, 6) on the curve E(2, 3, 97), we begin to notice that the values loop around (*figure 10*) [1].
In fact, we are creating a **cyclic subgroup** of points on curve E, which are generated by $P(3, 6)$. A group G is said to be **cyclic** if:

- G has a finite number of elements.

0G = O	5G = O	10G = O	(5k)G = O
1G = (3,6)	6G = (3,6)	11G = (3,6)	(5k + 1)G = (3,6)
2G = (80,10)	7G = (80,10)	12G = (80,10)	(5k + 1)G = (80,10)
3G = (80,87)	8G = (80,87)	13G = (80,87)	(5k + 1)G = (80,87)
4G = (3,91)	9G = (3,91)	...	(5k + 1)G = (3,91)

Figure 10: Multiples of $P(3,6)$

- $\forall x, y \in G,\ \exists n \in \mathbb{Z},\ y = x \cdot n$

The order of the subgroup is the number of elements it contains. The subgroup in the previous example has an order of 5:
The group includes the elements: $(3,6)$, $(80,10)$, $(80,87)$, $(3,91)$ and $\mathcal{O}$.

3.4.4 Notation

By convention, an elliptic curve C for cryptography is defined by the following parameters:

- a and b: the parameters of the equation of the curve
- p: a prime number which defines the space
- G: A point on the curve used as a generator of a cyclic sub-group
- n: the order of the sub-group created by G

An elliptic curve is therefore defined by the following quintuplet:

$$C = (a, b, p, G, n)$$

4 ECDSA – Elliptic Curve Digital Signature Algorithm

The Elleptic Curve Digital Signature Algorithm (ECDSA) takes a string as a input and outputs a signature under the form of a pair of integers, (r, s). The safety of the algorithm is based on difficulty of reversing point summation on elliptic curves.
Adding point P to itself n times is relatively easy to compute. In fact, thanks to algorithms like *double and add* [1], the complexity of $P \cdot n$ is reduced to $\mathcal{O}(log(n))$.
On the other hand, recovering n from P and $P \cdot n$ is much harder, since it requires an attacker to check if every multiple P is equal to $P \cdot n$.
The integer n is noted as the private key and $P \cdot n$ is the public key (the point P is standard and public).

4.1 Generating a signature

To calculate a signature, the sender and receiver must agree on a curve with parameters (a, b, p, G, n). K_{priv} and K_{pub} are the private and public keys (respectively) of the sender of the message.

1. Compute the hash e = hash(m)
2. Choose a random number k, such that $k \in [1, n-1]$
3. Compute the point $P(x_1, y_1) = k \cdot G$. $r \equiv x_1$ mod n.
4. Compute $s = k^{-1}(e + r \cdot K_{priv})$ mod n. Si s = 0, go back to step 2.
5. The signature is the pair (r, s).

4.2 Verifying a signature

Once the message m is received, the signature (r, s) is to be verified.

1. Check that r and s are integers such that $r, s \in [1, n-1]$. Otherwise the signature is invalid.
2. Compute the hash e = hash(m).
3. Compute the values $u_1 = e \cdot s^{-1}$ mod n and $u_2 = r \cdot s^{-1}$ mod n.
4. Retrieve the point $P(x_1, y_1) = u_1 \cdot G \oplus u_2 \cdot K_{pub}$. If P is the point at infinity, then the signature is invalid.
5. If $r \equiv x_1$ mod n, then the signature is valid.

[1] Wikipedia: Elliptic curve point multiplication

4.3 Example

4.3.1 Encrypting a message

In this example, Bob wishes to wishes to send the message m to Alice, using his key pair: $k_{priv} = 3, K_{pub} = (22, 30)$ as well as the following simplified elliptic curve: $E = (26, 3, 31, (21, 18), 11)$.
1. Compute the hash e = hash(m)
e = SHA3_256(m) = 10125
2. Choose a random number k, such that $k \in [1, n-1]$
We choose $k = 10$ ($k \in [1, 10]$)
3. Compute the point $P(x_1, y_1) \equiv k \cdot G$ $r \equiv x_1$ *mod n.*
$P(x_1, y_1) = k \cdot G$
$10 \cdot (21, 18) = (21, 13)$
$r \equiv 21 \text{ mod } 11 \Rightarrow r = 10$
4. Compute $s = k^{-1}(e + r \cdot K_{priv})$ *mod n. If s = 0, go back to step 2.*
First, let's compute the inverse of k mod 11, which is 10.
$s = 10 \cdot (10125 + 10 * 3) mod 11 = 9$
5. The signature is the pair (r, s)*. Bob can now send the document with the signature to Alice.*
The signature is (r, s) = (10, 9).

4.3.2 Decrypting a message

To finish the example, Alice has received Bob's message m as well as the signature pair (10, 9). She also knows the curve used which is $E = (26, 3, 31, (21, 18), 11)$ as well as Bob's public key: $K_{pub} = (22, 30)$.
1. Check that r and s are integers such that $r, s \in [1, n-1]$*. Otherwise the signature is invalid.*
r = 10 and s = 9, they are included in the interval [1, n-1]
2. Compute the hash e = hash(m). Just like Bob did.s
e = SHA3_256(m) = 10125
3. Compute the values $u_1 = e \cdot s^{-1}$ *mod n and* $u_2 = r \cdot s^{-1}$ *mod n.*

$$u_1 \equiv e \cdot s^{-1} \text{ mod } n \equiv 10125 \cdot 5 \text{ mod } 11 \equiv 3$$
$$u_2 \equiv r \cdot s^{-1} \text{ mod } n \equiv 10 \cdot 5 \text{ mod } 11 \equiv 6$$

4. Retrieve the point $P(x_1, y_1) = u_1 \cdot G \oplus u_2 \cdot K_{pub}$*. If P is the point at infinity, then the signature is invalid.*
$P(x_1, y_1) = u_1 \cdot G \oplus u_2 \cdot K_{pub}$
$P(x_1, y_1) = 3 \cdot (21, 18) + 6 \cdot (22, 30)$
$P(x_1, y_1) = (21, 13)$
5. If $r \equiv x_1$ *mod n, then the signature is valid.*

$$r \equiv x_1 \text{ mod } n$$
$$10 \equiv 21 \text{ mod } n$$

Therefore the signature is **valid**.

5 Common Vulnerabilities and Exposures (CVE)-2022-21449

The vulnerability CVE-2022-21449 "**allows unauthenticated attacker** with network access via multiple protocols **to compromise Oracle Java SE**, Oracle GraalVM Enterprise Edition. **Successful attacks** of this vulnerability can result in **unauthorized creation, deletion or modification access to critical data** or all Oracle Java SE, Oracle GraalVM Enterprise Edition accessible data." [3]
The vulnerability is not due to a weakness of the ECDSA algorithm itself, but because of a faulty implementation by Oracle. In fact, the vulnerability first appeared in *Java 15* when the code of the EC code was rewritten from native C++ to Java [2]. The original C++ implementation was not vulnerable to this attack, but the rewrite was.
The vulnerability was fixed after Java 18, in April 2022, and had appeared in September 2020.
The Common Vulnerability Scoring System (CVSS) score awarded to the CVE-2022-21449 was 7.5/10 (high).

5.1 How does the attack work ?

By entering the signature pair (r, s) = (0, 0), Oracle's services would consider the signature as valid.

5.2 The math behind the vulnerability

To fully understand why Oracle's implementation fails to verify signatures properly, let us run through the steps of ECDSA verification.

5.2.1 Oracle doesn't verify if s = r = 0

1. Check that r and s are integers such that $r, s \in [1, n-1]$. Otherwise the signature is invalid.
A crucial verification of the algorithm is omitted: Oracle forgot to verify that the provided signature was not null. In the earlier version (before Java 15), the verification was not forgotten. The verification was forgotten when translating the code about elliptic curves from C++ to Java.

5.2.2 Petit Théorème de Fermat

3. Compute the values $u_1 = e \cdot s^{-1} \bmod n$ and $u_2 = r \cdot s^{-1} \bmod n$.
In order to compute the inverse of s modulo n, Oracle uses the Petit Théorème de Fermat [2]. Problems begin since we are computing the inverse of 0. In theory, the inverse of 0 is not defined, unfortunately, when using the Petit Théorème de Fermat:

$$x^n = x \bmod n$$
$$x^{(n-1)} = 1 \bmod n$$
$$x^{(n-2)} = x - 1 \bmod n$$

Then with $x = 0$, we get:

$$0^n = 0 \text{ mod n}$$

Which means that:

$$u_1 = e \cdot s^{-1} = 0$$
$$u_2 = r \cdot s^{-1} = 0$$

Note that 0 is a forbidden value for x when using the Théorème de Fermat.

5.2.3 The point P

4. Retrieve the point $P(x_1, y_1) = u_1 \cdot G \oplus u_2 \cdot K_{pub}$. If P is the point at infinity, then the signature is invalid.
Since u_1 and u_2 are equal to 0, the point P...

$$P(x_1, y_1) = u_1 \cdot G \oplus u_2 \cdot K_{pub}$$
$$P(x_1, y_1) = 0 \cdot G \oplus 0 \cdot K_{pub}$$
$$P(x_1, y_1) = \mathcal{O} \oplus \mathcal{O}$$
$$P(x_1, y_1) = \mathcal{O}$$

... is the point at infinity.
If implemented properly, ECDSA would reject the signature since P shouldn't be the point at infinity if the signature was valid. Unfortunately, Orcal forgot to implement this verification.

[2]: $a^p \equiv a \mathrm{mod} p$

5.2.4 The point at infinity

5. If $r \equiv x_1$ mod n, then the signature is valid.
Oracle decided to implement the point at infinity as the point of coordinates (0, 0). If the rest of the algorithm is correct, this particular implementation is not a problem since the curves used for signatures exclude the point $(0, 0)$. Since P is the point at infinity, then $P = (0, 0)$, and:

$$r \equiv x_1 \text{ mod n}$$
$$0 \equiv 0 \text{ mod n}$$

References

[1] A. Corbellini. Elliptic Curve Cryptography: finite fields and discrete logarithms. URL: https://andrea.corbellini.name/2015/05/23/elliptic-curve-cryptography-finite-fields-and-discrete-logarithms/.

[2] Neil Madden. CVE-2022-21449: Psychic Signatures in Java. URL: https://neilmadden.blog/2022/04/19/psychic-signatures-in-java/.

[3] NIST. CVE-2022-21449 Detail. URL: https://nvd.nist.gov/vuln/detail/cve-2022-21449.

[4] Dr. Mike Pound. What are Digital Signatures? URL: https://www.youtube.com/watch?v=s22eJ1eVLTU.

[5] N. Sullivan. A (Relatively Easy To Understand) Primer on Elliptic Curve Cryptography. URL: https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/.

[6] wikipedia. SHA2. URL: https://en.wikipedia.org/wiki/SHA-2.