

COMP3633 Competitive Programming in Cybersecurity II



Common Attack Vectors for Elliptic Curve Cryptography

Jacky (Jackylkk2003)

April 26, 2024



>FIREBIRD CTF
TEAM



Code of Ethics

- ▶ The exercises for the course should be attempted **ONLY INSIDE THE SECLUDED LAB ENVIRONMENT** documented or provided. Please note that most of the attacks described in the slides would be **ILLEGAL** if attempted on machines that you do not have explicit permission to test and attack. The university, course lecturer, lab instructors, teaching assistants and the Firebird CTF team assume no responsibility for any actions performed outside the secluded lab.
- ▶ The challenge server should be regarded as a hostile environment. You should not use your real information when attempting challenges.
- ▶ Do not intentionally disrupt other students who are working on the challenges or disclose private information you found on the challenge server (e.g. IP address of other students). Please let us know if you accidentally broke the challenge. While you may discuss with your friends about the challenges, you must complete all the exercises and homework by yourselves.



Code of Ethics



Elliptic Curves are currently commonly used in various services, even in [Bitcoin protocols](#). And you should **NOT** attack any of these services without prior approval. DON'T -7.5!



- ① Introduction
- ② Mathematical Backgrounds
- ③ Elliptic Curve
- ④ Elliptic Curve Cryptography I - Diffie-Hellman Key Exchange
- ⑤ Elliptic Curve Cryptography II - Digital Signature Algorithm
- ⑥ References



Introduction



Speaker's Bio



No one cares.

If for some reason you care, try ~~OSINTing~~. (No, don't -7.5)





Additional Materials



Some more materials for this presentation can be found [here](#).






Goal of this Presentation



1. Introduce some basic ECC concepts to you
2. Convince you that ECC (at least the basics) is not rocket science
3. Let you understand some attacks on ECC, or at least, be able to use Sage to launch the attacks.
4. ~~Hopefully we can finish digital signature~~



I may go quite fast, but hopefully not too fast.

Feel free to stop me during the presentation if you are like , ,
or even .

I don't expect you to have many crypto background, so please
don't give up before trying. .

I tried to make this presentation more fun so that you will not fall asleep due to all the maths. Hopefully it is not like a boring math lecture.

I won't cover all the slides. Some are intended for your further understanding and exploration.



Software Installation

Please make sure you have installed Sage and some common crypto modules like pycryptodome (installed using pip) if you want to try out the attacks.

Sage can be installed in kali linux by following [this instruction](#) and [this instruction](#) in case you have encountered any difficulties.

An alternative to installing it on your machine is to use [Sagecell](#). Sage is very similar to Python. It can run most, if not all, Python programs. So, we will just cover some special usages related to ECC.



Mathematical Backgrounds



Review of Modular Arithmetic

Some important rules and notations in modular arithmetic:

- ▶ $a \bmod b = c$ means when a is divided by b , the remainder is c
- ▶ $a \equiv b \pmod{c}$ means $a \bmod c = b \bmod c$
- ▶ $(a + b) \bmod c = (a \bmod c + b \bmod c) \bmod c$, and the same holds for $-$ and \times , but not power and division.
- ▶ $a \equiv b \pmod{c}$ also means that $a - b \equiv 0 \pmod{c}$, $a - b$ is divisible by c
- ▶ $a \equiv b \pmod{c}$ also means that $a = kc + b$ for some integer k
- ▶ Division is not defined in modular arithmetic, but we have something similar to that.



Modular inverse and rational numbers

Can we express $1/2 \pmod{11}$ without using division?

Somewhat yes. We can treat $1/2$ as a number that becomes 1 after multiplying by 2. We denote it using 2^{-1} .

In this case, we know (magically) that $2 \times 6 \equiv 1 \pmod{11}$, so we have $2^{-1} \equiv 6 \pmod{11}$.

How to represent $3/2 \pmod{11}$?

$$3/2 \equiv 3 \times 2^{-1} \equiv 3 \times 6 \equiv 18 \equiv 7 \pmod{11}$$



As a side note, $1/2 \bmod 11$ and $3/2 \bmod 11$ are kind of abusing notations. Remember that division is not defined in modular arithmetic.



Rings

Ring is a set (of numbers) that has the following properties regarding addition:

- ▶ There is some element $\mathbf{0}$ in the ring that $\mathbf{0} + A = A + \mathbf{0} = A$
- ▶ For any A , you can find B such that $A + B = \mathbf{0}$. We say $B = -A$ (additive inverse of A)
- ▶ $(A + B) + C = A + (B + C)$
- ▶ $A + B = B + A$



Rings



Ring also has the following properties regarding multiplication:

- ▶ There is some element **1** in the ring that $\mathbf{1} \times A = A \times \mathbf{1} = A$
- ▶ $(A \times B) \times C = A \times (B \times C)$
- ▶ $A \times B = B \times A$



Modulo Fields

Like most Cryptosystems, we are dealing with numbers modulo a prime p . We denote it as a field F_p , which is just a set $\{0, 1, 2, 3, \dots, p-1\}$.

For example, F_5 is $\{0, 1, 2, 3, 4\}$.

And F_p is a ring, we will visualize it later.

Sage code:

$$F = GF(p)$$



Modulo Fields

If you have taken COMP2711 before...

Modular Arithmetic on \mathbb{Z}_m

- **Definition**

$$\mathbb{Z}_m = \{0, 1, \dots, m-1\}$$

- **Definition**

For $a, b \in \mathbb{Z}_m$

- $a +_m b = (a + b) \bmod m$
- $a \cdot_m b = (a \cdot b) \bmod m$
- **Examples**
 - $7 +_{11} 9 = (7 + 9) \bmod 11 = 16 \bmod 11 = 5$
 - $7 \cdot_{11} 9 = (7 \cdot 9) \bmod 11 = 63 \bmod 11 = 8$



Elliptic Curve



Elliptic Curve

No. It is not related to ellipses.

The Elliptic Curve is a graph that has a Weierstrass Equation.

$$y^2 = x^3 + ax + b$$

A visualization [here](#).

Sage code:

```
E = EllipticCurve(F, [a, b])
```

```
P = E(x, y)
```

where F is the modulo field (explained later in the slides), and P is a point on E .



Elliptic Curve

If you print out a point, you may see there are 3 numbers separated by :, like

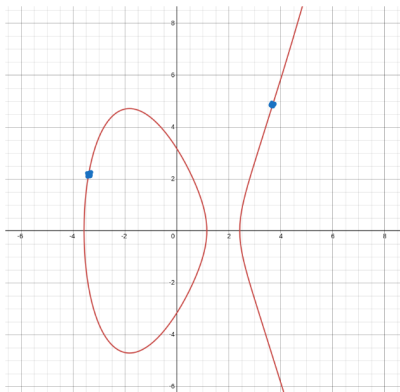
$(31 : 57 : 1)$

You can simply ignore the last number for now. The coordinates of this point is $(31, 57)$.



Point Addition

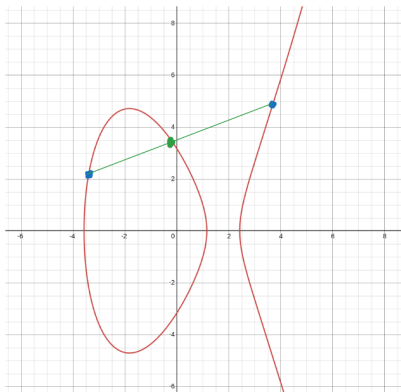
We have a strange way to add up 2 points.
Suppose we want to add the 2 blue points.





Point Addition

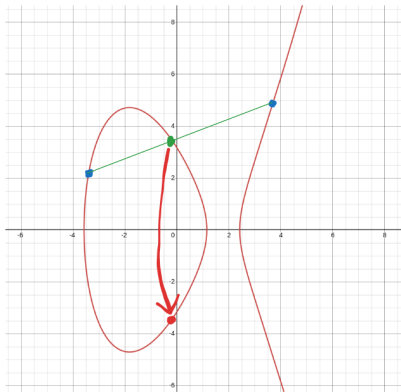
We first connect the 2 points using a line (infinite length), and find its intersection with the original curve.





Point Addition

Finally, the point is reflected across x-axis and that is the result of the addition (the red point).





Point Addition



Sage code:

```
A = E(A_x, A_y)
```

```
B = E(B_x, B_y)
```

```
C = A + B
```

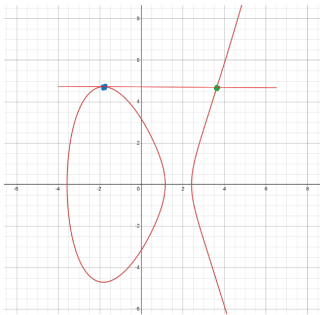


Does this always work?

Are there always 3 different points on that line?

What if we want to add the blue point and green point here?

Or maybe, to add blue point to blue point, how to even define the line?





Does this always work?



Just like defining the tangent, we use 2 very close points to calculate the slope.

We can apply the same concept here by treating the blue point as 2 very close but different points.

So blue + blue would be green flipped across the x-axis.

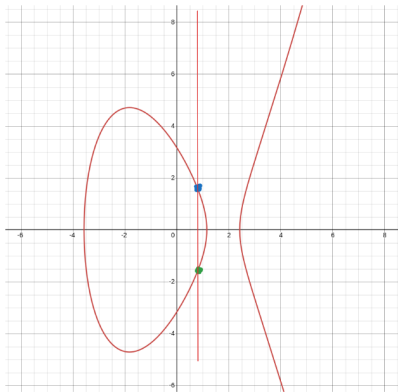
And blue + green would be blue flipped across the x-axis.

Any more cases that would “hack” the point addition?




Does this always work?

What if we add these 2 points?





Does this always work?

A horizontal bar with a teal segment on the left and an orange segment on the right.

We define a point O at infinity. The coordinates of it are (any, ∞) . All x values share the same O .

Also define $O + O = O$ and $O + P = P + O = P$.

In other words, O behaves like the **0** element in the ring properties.

A fun property is that 3 collinear points in the curve will sum to O .

If you print out the point O in sage, you will see $(0 : 1 : 0)$, and you can specify the point O using $E(0, 1, 0)$.

So the last number indicates whether it is the point at infinity or not.

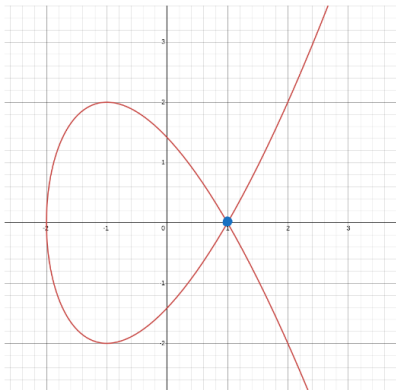
And again, anything more?



Does this always work?

What if we have a curve like $y^2 = x^3 - 3x + 2$?

How to add the blue point to itself?





Does this always work?

Simple answer, we cannot, (partly) since the slope is ambiguous. For these curves, we call them singular curves, since they contain singularity points.

A check for the singular curve is $4a^2 - 27b^3 = 0$. [No proof here, sorry 🤖]

We will not be focusing on singular curves today. So we assume $4a^2 - 27b^3 \neq 0$ for the rest of the slide.

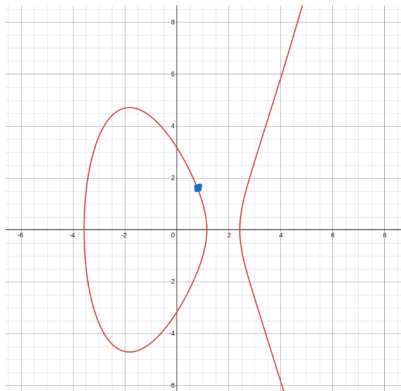
Example challenge: VolgaCTF 2024 Qualifier - Single R
Solution [here](#).



Point Negation

Can we negate the points?

Suppose $A(x, y)$ is the blue point, what is $-A$?



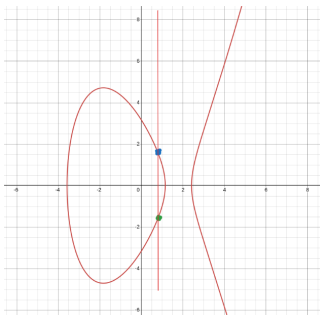


Point Negation

$-A$ should be a point that satisfy a property: $A + (-A) = \mathbf{0} = O$.

So $-A$ is the green point.

Also, notice that $-A$ is uniquely defined as $(x, -y)$.



Sage code: Left as an exercise.



Point Subtraction



It should be trivial after we define $-A$.

$$A - B = A + (-B)$$

Sage code: Left as an exercise again.



Point Multiplication with (Positive) Integers

Again, it should be trivial.

$$n \times A = A + A + \dots + A$$

Sage code:

```
A = E(A_x, A_y)
```

```
B = A * n
```

```
C = n * A
```

```
assert B == C
```

But how to compute it efficiently?

Obviously looping it n times is not good enough.



Double-and-Add Algorithm

We first write n as sum of powers of 2 (binary numbers 🐱)

Then $n \times A = 2^{n_1}A + 2^{n_2}A + 2^{n_3}A + \dots$

Computing $2^k A$ is easy, we can find it by doubling.

- ▶ Adding $1 \times A$ to itself gives $2 \times A$
- ▶ Adding $2 \times A$ to itself gives $4 \times A$
- ▶ Adding $4 \times A$ to itself gives $8 \times A$
- ▶ And it gets boring if I continue. 🐱



Seems familiar?

Modular Exponentiation

- How to compute

$$a^n \bmod m$$

efficiently for large n ?

- Repeated squaring method

- Compute

$$a^2 \bmod m$$

$$a^{2^2} \bmod m = a^4 \bmod m = (a^2 \bmod m)^2 \bmod m$$

$$a^{2^3} \bmod m = a^8 \bmod m = (a^4 \bmod m)^2 \bmod m$$

...

- Write n in binary $n = (b_k \dots b_1 b_0)_2$

$$a^n \equiv a^{b_0 \cdot 1} \cdot a^{b_1 \cdot 2} \cdot a^{b_2 \cdot 2^2} \dots \pmod{m}$$

- Example: $n = 50 = (110010)_2$

$$a^{50} \equiv a^{2^1} a^{2^4} a^{2^5} \pmod{m}$$



Modulo field

To incorporate modulo in the Elliptic Curve, we can modify the equation.

$$y^2 \equiv x^3 + ax + b \pmod{p}$$

Now this becomes a “curve” in the field F_p .

Now we know the meaning of F in the sage code above.



Order of curve



It means the number of different points that are in the field and is on the curve.

It is upper bounded by $2p + 1$ and in reality, tends to be much smaller than that.

Sage code:

```
E.order()
```



input: A curve $E : y^2 = x^3 + ax + b$

output: Order of E

function GETORDER(E)

$order \leftarrow 1$

▷ Point O at ∞

for $x \leftarrow 0$ to $p - 1$ **do**

for $x \leftarrow 0$ to $p - 1$ **do**

if $y^2 \bmod p = (x^3 + ax + b) \bmod p$ **then**

$order \leftarrow order + 1$

end if

end for

end for

return $order$

end function



Order of a Point

The order s of a point P is the number of distinct elements in the set $\{0P, 1P, 2P, 3P, \dots\}$.

This sequence forms a cycle when the curve is in a modulo field, so the order is a finite integer.

Some important properties of the order include:

- ▶ s is a factor of the order of the curve.
- ▶ s is the minimum positive integer such that $sP = O$

Combining both, we can know that if $n = \text{order of } E$, then $nP = O$ for any point P on E .

Sage code:

```
P = E(x, y)
```

```
P.order()
```



Trace of Frobenius

Some strange value defined as $t_p = p + 1 - \text{order of } E$.

Definition. The quantity

$$t_p = p + 1 - \#E(\mathbb{F}_p)$$

appearing in Theorem 6.11 is called the *trace of Frobenius* for E/\mathbb{F}_p . We will not explain the somewhat technical reasons for this name, other than to say that t_p appears as the trace of a certain 2-by-2 matrix that acts as a linear transformation on a certain two-dimensional vector space associated to E/\mathbb{F}_p .

Sage code:

```
E.trace_of_frobenius()
```



How about division?


Somehow we name the division problem as **Elliptic Curve Discrete Log Problem (ECDLP)** although you can say it is division rather than logarithm.

That is, find a number k such that $k \times G = P$ given 2 points G, P on E under modulo p .

This is known to be difficult. The current fastest known algorithm that solves ECDLP on Elliptic Curve runs in $O(\sqrt{p})$. Considering p could be hundreds of bits long, it is very slow.



Quick Summary

A horizontal bar with a teal segment on the left and an orange segment on the right.

E on field F_p is defined as $y^2 \equiv x^3 + ax + b \pmod{p}$
Solving the Elliptic Curve Discrete Log Problem (ECDLP) is difficult.
ECDLP: Given E, G, P , find k such that $k \times G = P$.



Elliptic Curve Cryptography I - Diffie-Hellman Key Exchange

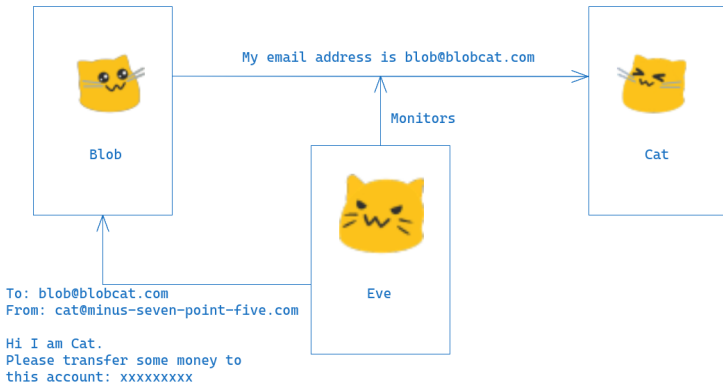


Diffie-Hellman Key Exchange

Now (finally) into a cryptography task: We want to share some secrets using a communication channel where someone (Eve 🐱) is eavesdropping.

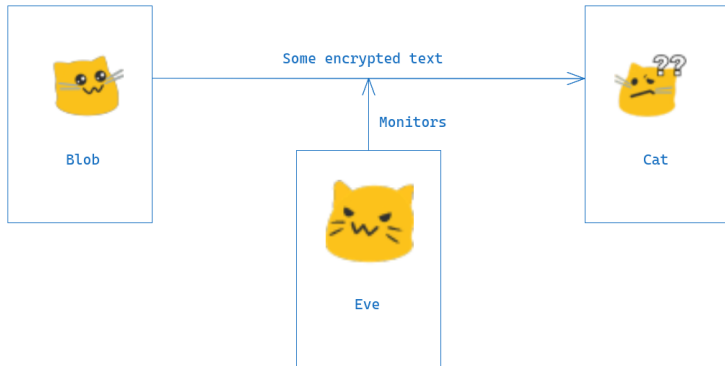


Diffie-Hellman Key Exchange





Diffie-Hellman Key Exchange

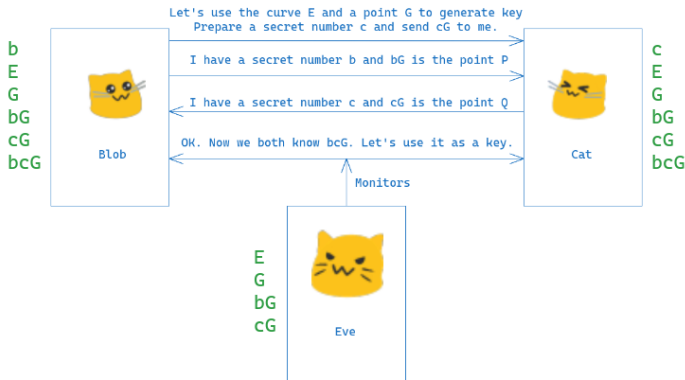


We need a way so that Cat understands what Blob says, but Eve should not know about it.



Diffie-Hellman on Elliptic Curve

Recall that ECDLP is difficult to solve. We can make use of it.





Diffie-Hellman on Elliptic Curve

What do Blob and Cat know?

Blob knows the secret number b , and also the cG that Cat sends to Blob.

Blob can compute $bcG = b \times (cG)$.

Cat knows the secret number c , and also the bG that Blob sends to Cat.

Cat can compute $bcG = c \times (bG)$.

They now share the same point (secret), usually using the x-coordinate of that point.



Diffie-Hellman on Elliptic Curve

How about Eve 🐱?

Eve knows about the curve E and G , and can see the bG and cG sent by Blob and Cat.

Eve does not have any easy way to reverse b and c . Remember ECDLP is difficult to solve.

Question: Can Eve directly compute bcG without knowing b or c ?



More Minor Details

Suppose for whatever strange reason, we don't want to send so much data just to exchange a key. Can we reduce the amount of data sent?

We can actually just send out the x-coordinates of the points.

But there are 2 possible values, suppose the point $P = (x, y)$ is on E , $-P = (x, -y)$ is also on E , but will have the same x-coordinates.



More Minor Details

Turns out it does not matter for this ambiguity.

$$n \times (-P) = -(n \times P)$$

Since they have the same x-coordinates.

$$-(x, y) = (x, -y)$$



After Diffie-Hellman

But we want to send messages, not to share keys!

Diffie-Hellman provides only a way to share keys or secrets, and does not provide a “guideline” on what to do next.

One common thing to do afterward is to use the shared key to do other encryption like one-time pad (XOR cipher), or AES.



Now onto Real Attacks

Time to do more maths.

As a warning, many more math ahead.

But, where are the vulnerabilities of Diffie-Hellman Key Exchange?



Now onto Real Attacks

There are 2 main places that are vulnerable:


1. The ECDLP problem, discrete log in Elliptic Curves.
2. Vulnerabilities after using Diffie-Hellman Key Exchange, for example, repeated key usage in one-time pad.

We will focus on the first one only.

That is, given $P = kG$, find k .



Naive ECDLP "Attack"

A horizontal bar with a teal segment on the left and an orange segment on the right.

```
def attack(P, G):  
    # We want to find k such that  $P = kG$   
    k = 0  
    while k * G != P:  
        k += 1  
    return k # Time complexity  $O(p)$ 
```



BSGS ECDLP "Attack"

```
def attack(P, G, p):  
    # We want to find k such that  $P = kG$   
    # p is the mod base of the curve E  
    order = P.order()  
    s = isqrt(order) + 1  
    baby_step = {}  
    for bs in range(s):  
        baby_step[bs * G] = bs  
    for gs in range(0, order, s):  
        if P - gs * G in baby_step:  
            return gs + baby_step[P - gs * G]  
    return None # Time complexity  $O(\sqrt{p})$ 
```



Pohlig-Hellman Attack

Condition: Order of E is easily factorizable (having small prime factors only)

We let $n = \text{order of } E = p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_r^{e_r}$

We want to solve for k in the equation $P = kG$.



Pohlig-Hellman Attack

To solve $P = kG$, we will decompose it into many smaller problems.

- ▶ What is $k \bmod p_1$?
- ▶ What is $k \bmod p_1^2$?
- ▶ ...
- ▶ What is $k \bmod p_1^{e_1}$?
- ▶ What is $k \bmod p_2$?
- ▶ What is $k \bmod p_2^2$?
- ▶ ...
- ▶ What is $k \bmod p_2^{e_2}$?
- ▶ ...



Pohlig-Hellman Attack

Let's first solve the easiest version $k \bmod p_1$.

We have $P = kG$ and $nP = nG = O$.

Let $P_0 = \frac{n}{p_1}P$, and we know $p_1P_0 = O$

Also let $G_0 = \frac{n}{p_1}G$, and we know $p_1G_0 = O$

Now we can solve $P_0 = kG_0$. A solution to $P = kG$ will also satisfy this new equation.

We can decompose $k = mp_1 + r$, where $r = k \bmod p_1$.



Pohlig-Hellman Attack



$$P_0 = kG_0$$

$$P_0 = (mp_1 + r)G_0$$

$$P_0 = mp_1G_0 + rG_0$$

$$P_0 = mO + rG_0$$


$$P_0 = rG_0$$

$$P_0 = (k \bmod p_1)G_0$$

So if we solve $P_0 = rG_0$, we get $k \bmod p_1$.



From ECDLP to ECDLP

A horizontal bar with a teal segment on the left and an orange segment on the right.

But the ECDLP problem is still there, we just changed the points!
The order is much smaller than what we had originally.
We can simply apply our simple “attacks” to solve it. We can use
BSGS, but the linear one also works (depending on the size of the
primes).



Pohlig-Hellman Attack

Our second problem is $k \bmod p_1^t$, for demonstration, we will use $t = 2$ here.

We have $P = kG$ and $nP = nG = O$.

Let $P_1 = \frac{n}{p_1^2}P$, and we know $p_1^2 P_1 = O$

And let $G_1 = \frac{n}{p_1^2}G$, and we know $p_1^2 G_1 = O$

Now we can solve $P_1 = kG_1$. A solution to $P = kG$ will also satisfy this new equation.

We can decompose $k = m_2 p_1^2 + m_1 p_1 + r$, where $r = k \bmod p_1$,
 $m_1 p_1 + r = k \bmod p_1^2$

Note that r has been found in the previous step.



Pohlig-Hellman Attack



$$P_1 = kG_1$$

$$P_1 = (m_2 p_1^2 + m_1 p_1 + r)G_1$$

$$P_1 = m_2 p_1^2 G_1 + m_1 p_1 G_1 + rG_1$$

$$P_1 = m_2 O + m_1 p_1 G_1 + rG_1$$

$$P_1 - rG_1 = m_1(p_1 G_1)$$

After solving, we will know $k \bmod p_1^2 = m_1 p_1 + r$.



Pohlig-Hellman Attack



Keep solving the equations, we will know:

- ▶ $k \bmod p_1^{e_1}$
- ▶ $k \bmod p_2^{e_2}$
- ▶ ...

We can obtain k from all these equations using CRT, the Chinese Remainder Theorem.



I don't understand anything you just said

Sadly, it would be almost impossible to understand this without working on the math yourself or spending some time to read that. But since time is limited, let's write it in Sage.

```
k = discrete_log(P, G, operation='+')
```

Done. But remember this only works (fast) if the order of the curve contains only small prime factors.

~~In terms of solving CTF challenges, it is usually more important for you to know HOW to do discrete log, but not WHY it works.~~



The Only Demonstration



HTB Cyber Apocalypse 2024 Hacker Royale - Arranged
Writeup [here](#).



Other Attack Methods





- ▶ Smart's Attack
 - ▶ Requires Trace of Frobenius $= 1$
 - ▶ Example attack code [here](#).
- ▶ Frey-Ruck attack
- ▶ MOV attack
- ▶ Pollard's Rho Attack
- ▶ And much more



Another Problem



Looks like our lovely blobcat Eve  is unhappy because the Diffie-Hellman is too secure that the above attacks do not work and Eve cannot eavesdrop on the conversation going on with Blob and Cat and cannot -7.5.

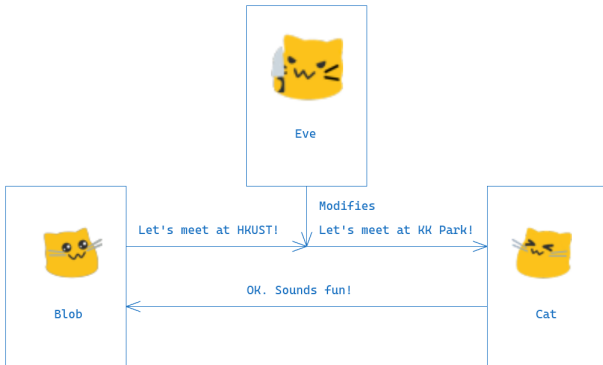
So now Eve is more aggressive (and got a knife ) and tries to tamper with the messages sent between Blob and Cat instead of just passively eavesdropping.

There is basically no way to prevent this, but can we at least detect this?



Another Problem

For simplicity, let's suppose we allow messages to be seen by Eve, but Eve may modify the message.





Digital Signature

A digital signature is a proof that someone is sending a particular message.

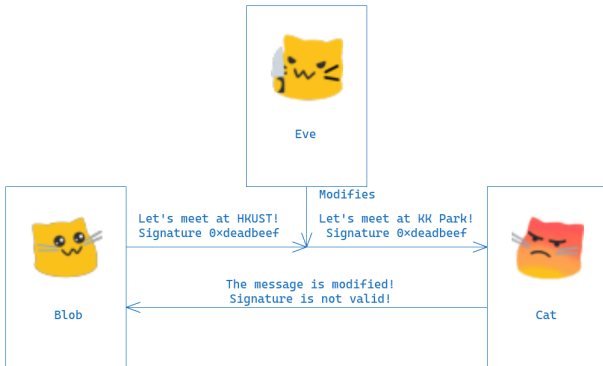
It is practically impossible for anyone to forge a digital signature for others.

That is, only one person can sign the message, but everyone else can verify that the message is indeed sent by that person.



Digital Signature

Suppose we have some magical digital signature system.





A Tutorial on the Magic

The Digital Signature Algorithm on Elliptic Curves is called Elliptic Curve Digital Signature Algorithm (ECDSA).

First, we generate a curve E on field F_p and a point G on E . Let $q = \text{order of } E$. We have an additional requirement that q is a prime. (No more Pohlig-Hellman attacks 🐱)

Now Blob needs to choose a secret private signing key b , $1 < b < q - 1$. And Blob publishes a public verification key $V = bG$.



Magic Tutorial (con't)

Suppose m is the (hashed) plaintext message converted to integers.
If Blob wants to send the message, randomly choose a nonce e
(some random numbers to make things less predictable).
Compute eG and let $s_1 = (\text{x-coordinate of } eG) \bmod q$.
Also compute $s_2 = (m + b \cdot s_1) \cdot e^{-1} \bmod q$.
The digital signature is (s_1, s_2) . It is NOT a point on E .



Verifying the Magic

Now Cat should verify whether a pair (s_1, s_2) is a valid digital signature of a (hashed) message m .

Compute $v_1 = m \cdot s_2^{-1} \bmod q$ and $v_2 = s_1 \cdot s_2^{-1} \bmod q$

The signature is valid if and only if (x-coordinate of $(v_1 \times G + v_2 \times V)) \bmod q = s_1$.



A code for ECDSA is provided in the materials of this presentation.



A (Incomplete) Proof that No One Cares

We have $V = bG$


$$s_1 = (\text{x-coordinate of } eG) \bmod q, s_2 = (m + b \cdot s_1) \cdot e^{-1} \bmod q$$

$$v_1 = m \cdot s_2^{-1} \bmod q, v_2 = s_1 \cdot s_2^{-1} \bmod q$$

We want to show that when the signature is valid, $(\text{x-coordinate of } (v_1 \times G + v_2 \times V)) \bmod q = s_1$.




A (Incomplete) Proof Sketch that No One Cares


$$\begin{aligned} & v_1 \times G + v_2 \times V \\ &= m \cdot s_2^{-1} \times G + s_1 \cdot s_2^{-1} \times bG \\ &= (m + b \cdot s_1) \times s_2^{-1} \times G \\ &= (m + b \cdot s_1) \times (m + b \cdot s_1)^{-1} \cdot eG \\ &= eG \end{aligned}$$

Kind of abusing notations here. This just shows a brief idea of how to prove its correctness. I don't intend to be presenting a formal and correct proof here.




Will Anyone Ever Use this Complicated Thing?

A horizontal bar with a teal segment on the left and an orange segment on the right.

Turns out it is very commonly used nowadays in real cryptosystems. Examples include [Bitcoin protocol](#) and Ethereum protocol. They both use the curve $y^2 = x^3 + 7 \pmod{a \text{ large number}}$. This curve is named as secp256k1.



Some Rules on Digital Signatures


A horizontal bar with a teal segment on the left and an orange segment on the right.

Always compute the signatures on a hashed message but not the message itself.

Never reuse the nonces if any. In our case, e is the nonce.



Attacks

A horizontal bar with a teal segment on the left and an orange segment on the right.

Suppose we can solve ECDLP, the discrete log problem, then ECDSA would be vulnerable. The attack is easy (at least compared to ECDLP) if you spend some time reversing the private key b from s_1 and s_2 .

But it is not sufficient for us to simply rely on the security of ECDLP.



Nonce Reuse

The signature contains 2 parts,
 $s_1 = (\text{x-coordinate of } eG) \bmod q$
 $s_2 = (m + b \cdot s_1) \cdot e^{-1} \bmod q$

What happens if we know that there are 2 different signatures that are generated using the same e ?



Nonce Reuse

$$s_1 = (\text{x-coordinate of } eG) \bmod q$$

$$s_2 = (m + b \cdot s_1) \cdot e^{-1} \bmod q$$

Suppose we have 2 tuples, (m, s_1, s_2) , and (m', s'_1, s'_2) from 2 different (hashed) messages m and m' .

One very important fact is that $s_1 = s'_1$. So we can calculate

$$s_2 - s'_2 = (m - m') \cdot e^{-1} \bmod q$$

It is possible for us to solve for e and hence solve for b ! The private key is leaked!!

Now, the digital signature can be forged.

Example attack code [here](#).



References

A horizontal bar with a teal segment on the left and an orange segment on the right.

Darren's presentation on Elliptic Curve Cryptography last year (No I didn't copy directly from him, we discussed quite different topics)

[An Introduction to Mathematical Cryptography Second Edition](#) by

Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman, ISBN:

978-1493917105

HKUST COMP2711 Lecture Notes

[crypto-attacks](#) repository by jvdsn

Bitcoin wiki on [ECDSA](#) and [secp256k1](#).

[Sage documentation](#)



Thank you!

Thank you!



And sorry for not having clear blobcats.