

Daniel Gray **CTF Write Up**

Competition: San Diego CTF 2022 (May 6 - 8) 2022

Weight: 24.39

Teamname: hyperfella

I chose to participate in San Diego CTF 2022 w with 3 other people this past weekend (May 6-8). SDCTF is A highly respected CTF. It was great! Now I know how easy it is to play a public CTF and will play more in the future. I worked with 3 team members Wei Wu, Scott Pender, and Soumaya LG Joshi. They were all very active during the weekend trying their best to solve challenges. We started with a teamwork approach, putting 2 members on each challenge. I originally started on an OSINT with Wei called paypal playboy, but when I found out that I had to know a little about bitcoin to solve this challenge, I naturally started looking around for other challenges to do. I was able to solve two hard challenges during competition time one 300 point Hard and another 400 point hard. The first one I will describe the steps for is called bishops duel which was marked as a Hard 300 point challenge broken up into a 100 point and 200 point flag.

On the write up assignment instructions, it says we can write about at least one challenge. So I will discuss the two challenges I solved along with a third I tried to solve during the competition.

Challenge: Bishops Duel

Rating: Hard

points: 300

Here is the exact prompt from sdctf:

chess players will know that if one bishop is on a dark square and another bishop is on a light square, there is no possible way that the two bishops can ever attack or be attacked by one another. Yet this was the situation that the game provided to us. Two bishops both on dark squares. Thus impossible to get captured.

Hmmmmmm.....

My next move was to open up the source code that was included. Looks like it was the code for the game. It was a .rs file! I have never worked with Rust before but heard a few classmates talking about it this semester. I took an hour to watch a few Rust tutorials on youtube and got acquainted with the syntax.

I learned that in Rust, buffer overflows are impossible or at least very hard to cause. Unlike C. So pwning was out.

I noticed a few things. I noticed some bounds checking to make sure a piece didn't go off the edge. I also noticed a slightly unusual position marking system. Instead of a tuple or array to represent the position on the board like (2,7), the position is just marked with a number between 1 and 64 (to represent all 64 spaces on the board) Although this logic could potentially work out fine, the strangeness of this choice made me think that there might be a flaw in the code when it came to handling any situation where the piece reached the end of the board.

```
// make sure we don't exceed the bottom and top of the chessboard
if new_position < 0 || new_position > 64 { return false; }
// make sure we don't exceed the left and right edge of the chessboard
```

Avoiding dead ends and distractions

I also found it strange that there is always an option to request a draw before every move. Because both bishops were on the same color squares, I initially thought that requesting a draw would be the correct path. But upon further inspection of the source code, this draw option is only meant to distract. As you can see from the source code, as soon as a player opts for a draw, he will get the message "A brave bishop shall never resign!" and just be prompted to make another move again.



```
if is_player_turn {
    // if its the white bishop's turn
    execute!(stdout, style::Print("You are the white bishop. Input move > ")).unwrap();
    let command: String = read!("{ }\n");

    let mut command_chars = command.chars();
    if let Some(c1) = command_chars.next() {
        match c1 {
            'r' | 'R' => {
                pause("A brave bishop shall never resign!");
            },
            'd' | 'D' => {
                execute!(stdout, style::Print("The black bishop is thinking.")).unwrap();
                sleep(Duration::from_millis(1000));
                execute!(stdout, style::Print(".")).unwrap();
                sleep(Duration::from_millis(1000));
                execute!(stdout, style::Print(".")).unwrap();
                sleep(Duration::from_millis(1000));
                pause("Accepted! The game is a draw!");
                break;
            },
            _ => {
```

Brute Forcing the possible moves

If a piece can hop off the edge of the board to get to the other side, it could possibly switch colors. I decided to make a few moves in the terminal to see if I could make this happen. No luck. So i made a script to more rapidly try various combinations of moves.

This is my exploit script. Note, I made constant changes to the moves by simply Switching around the letters and numbers.

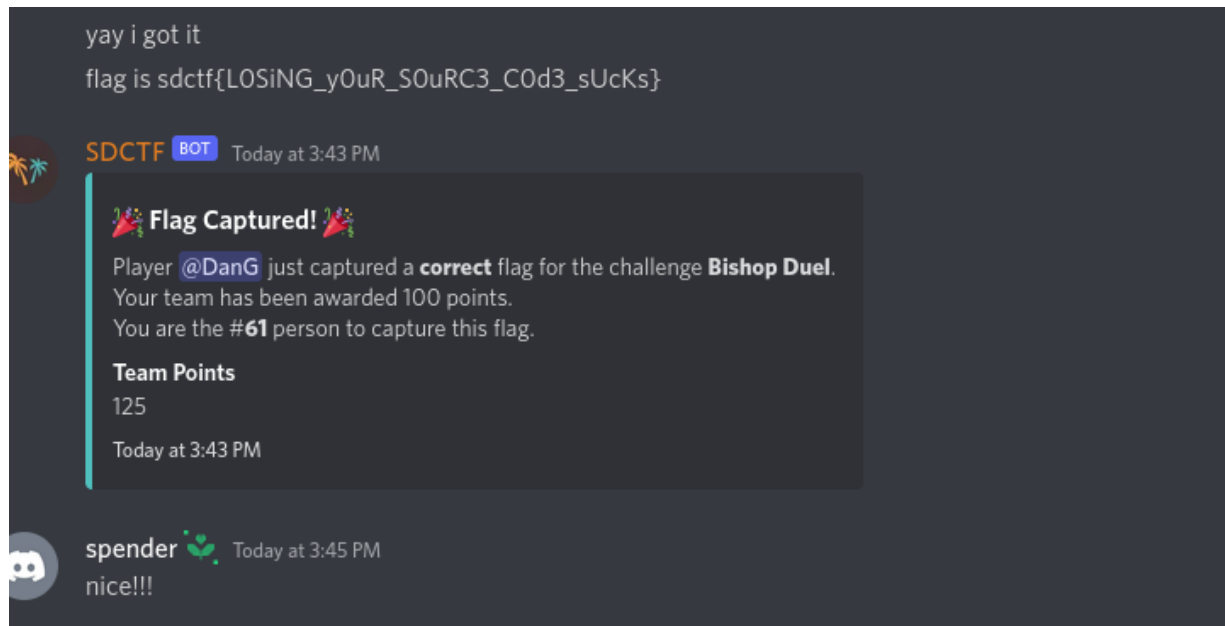
```

1  from pwn import *
2  import random
3  import time
4
5  #try different seeds for random to generate different moves
6  random.seed(5)
7
8  p = remote('bishop.sdc.tf' , 1337)
9  p.sendline('1') # presses enter
10 p.recvuntil(b'You are the white bishop. Input move >')
11
12 # every game begins the same. start off with a valid move to the center of the board
13 p.sendline(b'Z3')
14
15
16 # make some random moves, try to see if anything interesting happens
17 p.sendline('E' + str(random.randint(1,7)))
18 time.sleep(1) # so i have time to see if anything interesting happened
19 p.sendline('C' + str(random.randint(1,7)))
20 #time.sleep(1)
21 p.sendline('Q' + str(random.randint(1,7)))
22 #time.sleep(1)
23 p.sendline('Z' + str(random.randint(1,7)))
24
25 # make some more random moves
26 p.sendline('E1')
27 p.sendline('Z2')
28 p.sendline('C3')
29 p.sendline('Z4')
30 p.sendline('Z5')
31 p.sendline('C6')
32 p.sendline('Z7')
33 p.sendline('Z4')
34 p.sendline('E2')
35
36
37 p.interactive()
38

```

I finally noticed that the piece would jump from one side of the board to the other! Breakthrough. And it changed from a dark square to a light square! It is now possible to purposely get attacked by the piece and retrieve the first flag. I adjusted my script to end on that precise move, and then switch to interactive. I next moved right into the path of the black bishop and got my first flag:

flag 1: sdctf{L0SiNG_y0uR_S0uRC3_C0d3_sUcKs}



Finding the vulnerability and Zeroing in on the kill

The next flag was going to be harder because it would require the black bishop to inadvertently put himself into the path of the white bishop.

Upon further studying the code, it turns out I was correct about the bounds checking. There is bounds checking that works to prevent the bishop from going off the edge but it does not always work properly.

Looking at the following code, I noticed that there is an integer overflow error that will let the white bishop avoid the boarder check. This only happens if the number of legal squares you can move is a certain number. So i started trying different move combinations at the borders.

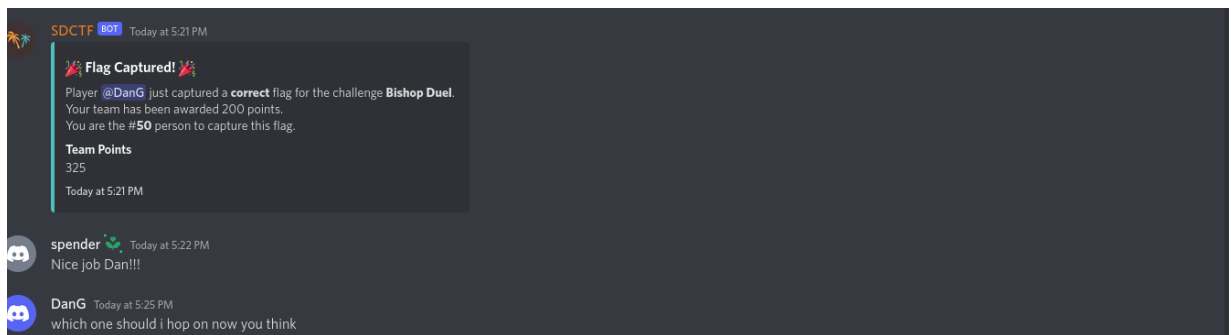
```
impl Bishop {
    fn get_xy(&self) -> (u16, u16) {
        let rows = self.position / 8 + 1;
        return ((self.position % 8) * 3 + rows, rows);
    }

    fn move_position(&mut self, dir: Direction, dist: u16) -> bool {
        let position_multiplier: i32 = match dir {
            Direction::UpRight => -7,
            Direction::DownLeft => 7,
            Direction::UpLeft => -9,
            Direction::DownRight => 9,
        };

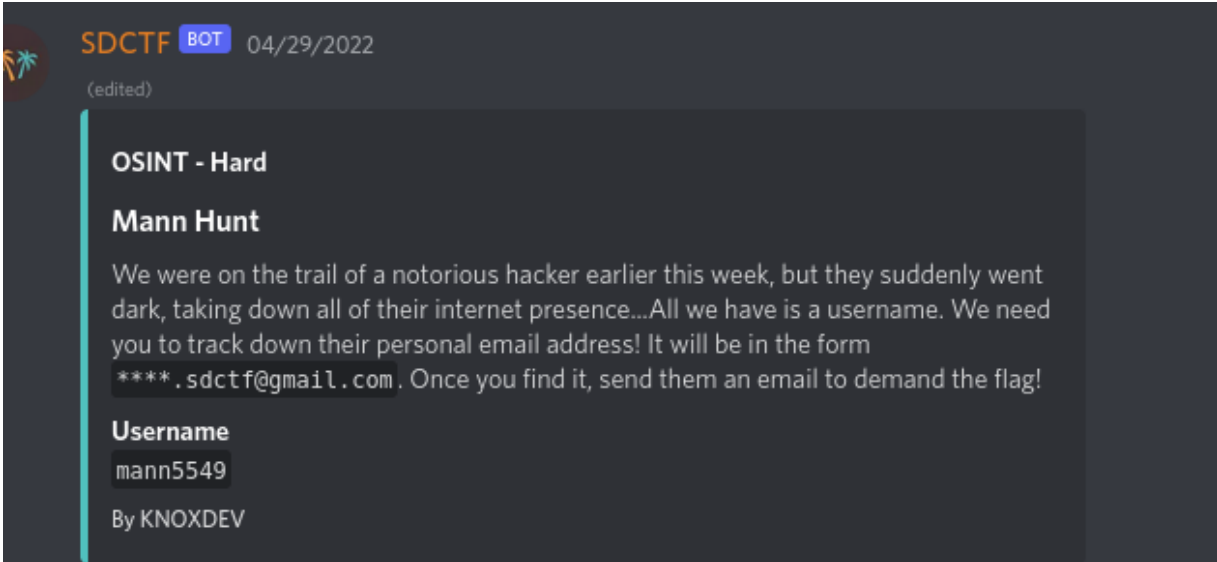
        let new_position = i32::from(self.position) + position_multiplier * i32::from(dist);
```

The above code actually has an off by one error. This will allow the white bishop to move past either the rightward and leftward edges and wrap around to the other side. This wrap around allows the bishop to break out of the normal pattern and be in a position to attack the black bishop. Trying different combinations at the borders eventually led me to get my flag. If I saw that there was 3 spaces to the end of the boarder, I would first try to move 3 spaces, then 4 , then 5. One of these seemed to always work to jumping me over to the other side. Attacking the black bishop with this strategy, I finally got him to enter my path by mistake so I could capture him.

You have claimed victory! Your flag is
sdctf{l_d1dnt_hAND_0u7_th3_s0urC3_c0D3_thIs_TIME}



Challenge: Mann Hunt
Weight: 400
Difficulty: Hard
Topic: Osint



This challenge was special to me it was the first Osint challenge I ever did. I got to learn about a completely new genre of CTF problem. My first move was to search the username in google. No luck. Next, I watched some YouTube Videos on Osint. I learned about the Osint Framework and started peaking around.

OSINT Framewo



I got a hint from the SDCTF admin who designed the challenge to search for old deprecated twitter accounts. So I used the wayback machine
And found an old version of the user Mann5549's page. It contained link to a blog mann.codes. I searched Google and found it! I went to the link.





mann.codes



mann.codes

I know who you are, and you'll never find me.

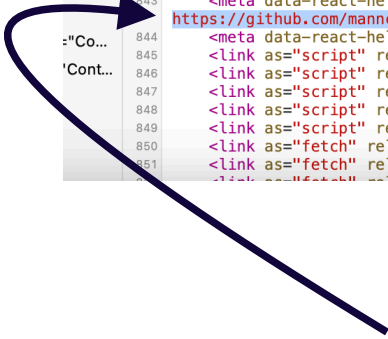
Nothing useful here. But since I had to go to so much trouble to get here, and the fact that this site exists indicated to me that I was clearly on the right track, I felt like

the only option right now must be to inspect the source code to this web page. So I did a right click and clicked "show page source". I looked around for awhile

and eventually found a meta tag description field that contained the link to an old GitHub repo.

mann.codes

I know who you are, and you'll never find me.



```
826    })
827  }, !0);
828  </script>
829  <link rel="alternate" type="application/rss+xml" title="Gatsby Starter Blog RSS Feed" href="/rss.xml"/>
830  <link rel="icon" href="/favicon-32x32.png?v=4a9773549091c227cd2eb82ccd9c5e3a" type="image/png"/>
831  <link rel="manifest" href="/manifest.webmanifest" crossorigin="anonymous"/>
832  <link rel="apple-touch-icon" sizes="48x48" href="/icons/icon-48x48.png?v=4a9773549091c227cd2eb82ccd9c5e3a"/>
833  <link rel="apple-touch-icon" sizes="72x72" href="/icons/icon-72x72.png?v=4a9773549091c227cd2eb82ccd9c5e3a"/>
834  <link rel="apple-touch-icon" sizes="96x96" href="/icons/icon-96x96.png?v=4a9773549091c227cd2eb82ccd9c5e3a"/>
835  <link rel="apple-touch-icon" sizes="144x144" href="/icons/icon-144x144.png?v=4a9773549091c227cd2eb82ccd9c5e3a"/>
836  <link rel="apple-touch-icon" sizes="192x192" href="/icons/icon-192x192.png?v=4a9773549091c227cd2eb82ccd9c5e3a"/>
837  <link rel="apple-touch-icon" sizes="256x256" href="/icons/icon-256x256.png?v=4a9773549091c227cd2eb82ccd9c5e3a"/>
838  <link rel="apple-touch-icon" sizes="384x384" href="/icons/icon-384x384.png?v=4a9773549091c227cd2eb82ccd9c5e3a"/>
839  <link rel="apple-touch-icon" sizes="512x512" href="/icons/icon-512x512.png?v=4a9773549091c227cd2eb82ccd9c5e3a"/>
840  <title data-react-helmet="true">All posts | mann.codes</title>
841  <meta data-react-helmet="true" name="description" content="Contribute to the blog at
842  https://github.com/mannnyber/manncodes.github.io"/>
843  <meta data-react-helmet="true" property="og:title" content="All posts"/>
844  <meta data-react-helmet="true" property="og:description" content="Contribute to the blog at
845  https://github.com/mannnyber/manncodes.github.io"/>
846  <meta data-react-helmet="true" property="og:type" content="website"/>
847  <link as="script" rel="preload" href="/webpack-runtime-7d0b522f562915713d3b.js"/>
848  <link as="script" rel="preload" href="/framework-e413e527015be9a1bdfd.js"/>
849  <link as="script" rel="preload" href="/app-ca822c6c494526c5bd8e.js"/>
850  <link as="script" rel="preload" href="/commons-ca5a9f45e933ef651e4c.js"/>
851  <link as="script" rel="preload" href="/component---src-pages-index-js-1dd2ec5b0e837db96a78.js"/>
852  <link as="fetch" rel="preload" href="/page-data/index/page-data.json" crossorigin="anonymous"/>
853  <link as="fetch" rel="preload" href="/page-data/sq/d/2841359383.json" crossorigin="anonymous"/>
```

Naturally, I went to this GitHub repo. Here is what the linked GitHub repo looked like:

mannycyber/manncodes.github.io

Search or jump to... / Pulls Issues Marketplace

mannycyber / manncodes.github.io Public Watch 1

Code Issues Pull requests Actions Projects

main Go to file Add file Code

mannycyber removed everything because I'm bei... on Jan 7 2

src	removed everything because I'm b...	4 months ago
static	init	4 months ago
.gitignore	init	4 months ago
.node-version	init	4 months ago
.prettierignore	init	4 months ago
.prettierrc	init	4 months ago
LICENSE	init	4 months ago

Peaked around. Couldn't find anything for awhile but then started peaking around the Git history because we know this guy
Is trying to delete his tracks. So I must be looking for some older code. I eventually found his real name, Emanuel Hunt.

```
src/components/seo.js
@@ -54,26 +54,6 @@ const Seo = ({ description, lang, meta, title }) => {
 54 54     property: 'og:type',
 55 55     content: 'website',
 56 56   },
 57 -   {
 58 -     name: 'twitter:card',
 59 -     content: 'summary',
 60 -   },
 61 -   {
 62 -     name: 'twitter:creator',
 63 -     content: site.siteMetadata?.social?.twitter || '',
 64 -   },
 65 -   {
 66 -     name: 'twitter:title',
 67 -     content: title,
 68 -   },
 69 -   {
 70 -     name: 'twitter:description',
 71 -     content: metaDescription,
 72 -   },
 73 -   {
 74 -     name: 'author',
 75 -     content: 'Emanuel Hunt'
 76 -   },
 77 57   ].concat(meta)}
 78 58   />
 79 59 }
```

My next move was to google him. I found a link to his twitter account!
Definitely on the right track here!



Emanuel Hunt

Blogger and Coder for ACM at UC San Diego

UC San Diego

San Diego, California, United States · [Contact info](#)

2 connections

[Connect](#)

[Message](#)



About

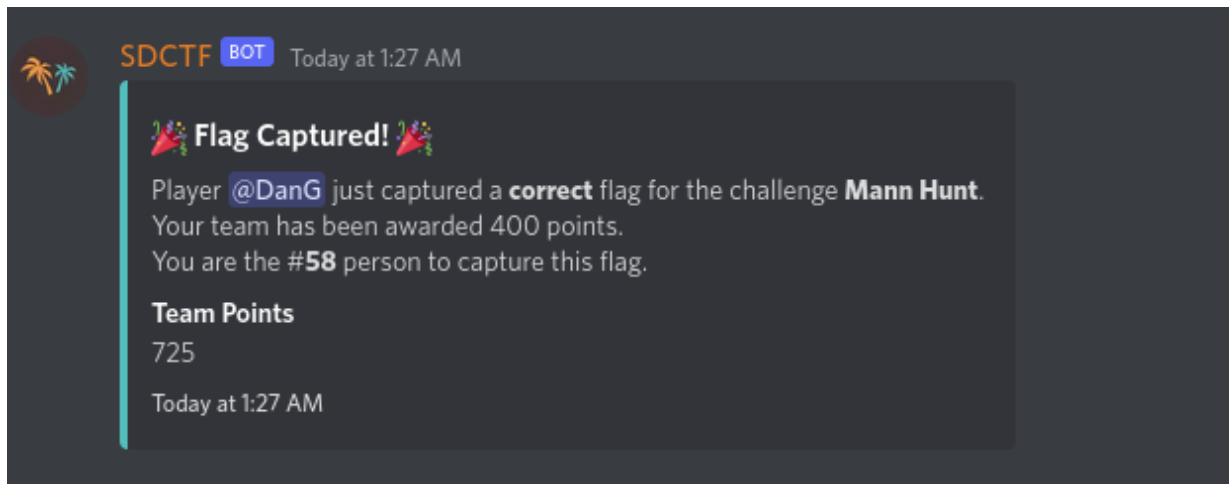
I'm probably the person you've been hunting for, wink wink.

You can get my resume at:

https://drive.google.com/file/d/10No4G_5iv2t5jxbvg2-weXkFouZrnQtg/view?usp=sharing

If you go to his resume , I finally found the email that is listed for him. I emailed demanding a flag, and finally got it!

Here is the flag! sdctf{MaNN_tH@t_w@s_Ann0YinG}



Challenge: Kleptomaniac
Weight: 400
Difficulty: Hard
Topic: Crypto

CRYPTO - Hard

Kleptomaniac

An internationally wanted league of bitcoin thieves have finally been tracked down to their internet lair! They are hiding behind a secure portal that requires guessing seemingly random numbers. After apprehending one of the thieves offline, the authorities were able to get a copy of the source code as well as an apparently secret value. Can you help us crack the case? We have no idea how to use this "backdoor number"...

"Secret Value"

```
5b8c0adce49783789b6995ac0ec3ae87d6005897f0f2ddf47e2acd7b1abd
```

Connect via

```
nc lair.sdc.tf 1337
```

Source code

[main.py](#)

By KNOXDEV

The math part

I spent the rest of the competition time attempting a super interesting crypto problem called Kleptomaniac. It was based on a backdoor that exists in elliptic key cryptography. I studied elliptic key cryptography a tiny bit in Internet Security and Privacy but totally forgot the concepts. So I pulled up a few videos by Computerphile which I used to watch a lot of during ISP. The challenge prompt gave us a secret key that they didn't know how to use and mentioned that it was some kind of backdoor. So when I found that my favorite computer file guy had a video on the elliptic curve back door, I knew this must be the exact thing I needed to research about. Basically, elliptic curves are used instead of modular arithmetic to establish a shared secret between two parties. Let's call them Alice and Bob over an insecure channel. Although elliptic curves does the same job as modular arithmetic, many have turned to the use of Elliptic curves because you can potentially use a smaller key than you would with modular arithmetic. So even though it seems much more complicated, there are efficiency reasons someone might want to use elliptic curve cryptography over traditional modular arithmetic.



Elliptic Curve Back Door - Computerphile

Computerphile ✓

445K views • 4 years ago

The math is quite complicated but not that bad. I first reminded myself about how The regular Diffie Hellman Key exchange works with modular arithmetic, and then taught myself how elliptic curve is accomplishing the same thing. Basically, Alice and Bob

both start with their own private key, lets call each one A and B. So that's in their private space. Only they can't see it. In the public space (where a man in the middle attacker

would be able to see) is a generator g , and big prime number N . If we do g , raise it to the power of A(the private key) and take mod N , Alice has something that she can Send across the network. Although a man in the middle can see this, he can't do anything with it. Imagine a big circle with N notches. We know $(g^A) \bmod N$ must lie on the

Circle. So if a man in the middle sees $(g^A) \bmod n$, he will see where it is on the circle, but he has no idea what power of g (which is A) was used to get it to that point.

When Bob receives this message, he can use his private key B, to decrypt the message. And the opposite can happen when Bob wants to send a message. Basically, the

shared secret is $(g^A) \bmod n$ when Alice is sending it and can be decrypted by Bob using his secret key B. Or the shared secret is $(g^B) \bmod n$ when Bob is sending a message

and can only be decrypted by Alice by using her secret key A. Anyway, I realize I am not explaining this too great and that is only because this is my third challenge. Just giving

A highlight here. Anyway, just like the attacker would have to guess what value of A was used as an exponent of the generator G to reach some point on a circle with N notches, we can do the same thing with an elliptic curve. We will move points around the elliptic curve and it will be impossible for the attacker to guess how many times it was moved around

The curve to get to that point. And it is precisely that value that will be the secret key.

While the attacker can steal from the network where a given point is on an elliptic curve, It is impossible to know how many times it was moved around to get there and the how many times it was moved around to get there is the precise secret key we are looking for.



Elliptic Curves - Computerphile

Computerphile ✓

419K views • 4 years ago

The super interesting part

In the early 2000s, the NSA published a list of four new random number generators. Most of these were standard but one of them was based on elliptic curves and started to peak

Everyone's interest. Like I mentioned in the last few sentences of the math part, The idea is that if we have a point on the Curve P , and a point on the curve aP which just means ,

some other point on the curve moved a times around starting From the original P . a is the secret key. It is impossible to know what a is. This works using random variables to get us

the initial point on the curve. But many crypto professionals have actually argued that this initial point generation is not so random after all. They suspected that a certain seed

Value may exist that generates these random numbers. How they came to this conclusion is also based on some awesome math. But there are heavy suspicions that the random

Number generator that determines the initial point on the elliptic curve is not so random. And that there could be this magic seed number that if known, can break all of Elliptic curve cryptography. And this random seed value is precisely the value that was given to us by the challenge! So the social implications of this are vast. Many crptographers

Are implying that the government would actually intentionally place a backdoor in a cryptographic system that they then attempted to spread across the world as one of the standards.

This is shady for obvious reasons. Anyway, when my masters degree is over, I plan to solve this challenge because the maths are super interesting to me.