**CyberCity Challenge Solution:  
Faith See (1002851)**

|  |
| --- |
| spaghetti.jpg |

<https://drive.google.com/file/d/1D8jnBB0OvCdwmKrB3lnu5l-nX9WRkveQ/view>

1. Obtain [recipe.txt](https://drive.google.com/open?id=1gPGqhXgFTCfRHGPlX0CL5hOwQ5b2P95E) hidden inside with the password, spaghetti:

* Use online tools: <https://futureboy.us/stegano/decinput.html>
* Use other tools: Steghide

1. Decipher [recipe.txt](https://drive.google.com/open?id=1gPGqhXgFTCfRHGPlX0CL5hOwQ5b2P95E) to obtain PurrflTneyvpXabg

* The smart way is to use an online [Chef Esolang converter](http://p-helpers.appspot.com/chef/chef.html)
* The Faith way is to do it manually:
  + Take the integers tied to each ingredient in the given order
    - Convert it from decimal (103, 98, 97, 88, 112, 118, 121, 101, 110, 84, 108, 102, 114, 114, 117, 80) to ASCII.
  + After obtaining the 16-character string (gbaXpvyenTlfrruP) reverse the order

1. Decrypt PurrflTneyvpXabg using a [ROT13 decoder](https://cryptii.com/pipes/rot13-decoder) to obtain CheesyGarlicKnot
2. Complete the inventory: inventory find CheesyGarlicKnot

This was the first time I have ever heard of Chef Esolang. This was an interesting exposure to esoteric programming languages. Each step gave me an opportunity to try different tools to tackle the challenges in the same way, with tools that required me to utilise terminal commands and others using online tools.

I have to admit though, the interface took some getting used to where I didn’t have autocomplete with tab or the ability to recall previous commands by pressing the up arrow key. Being able to paste without holding shift also felt a little strange, but it’s really cool that their team did this.

|  |
| --- |
| meatballs.jpg |

<https://drive.google.com/file/d/1jWgACaxHUhWVkXqLrxRsRDbIHXr0M6pm/view>

1. Obtain ciphertext by opening the image in a text editor and scroll to the bottom:  
   pASARFeDXPsj4V5u7qTOYoHzljCS+lc8YmSG/IiYJq74qKT3fcQAvuT24WJlMoMIinB9eKYGbR4FXRsAZ+kdbrytAmSDxFv33YARollK0IvXBVgVLsE9dmGghi71J9FP
2. Decrypt 128-bit ciphertext above using AES with ECB *(Egg-Cheese-Bacon in that order on the inventory list)* and the key CheesyGarlicKnot to obtain a base64 encoded cipher:

NmI2MjYxNzIxMjQ0NTkxNDkyNTk3MjQ1ZGUwN2RiZWYxNjI2YjQ3MjI0NGYzNDA2MTRmOTA4M2M2ZTY4M2E3YzA0NmMwZjBhNDYzNjBmNWI=

1. Decrypt the base64 encoded cipher with a decoder to obtain the plaintext which is Remi’s pin:   
   6b6261721244591492597245de07dbef1626b472244f340614f9083c6e683a7c046c0f0a46360f5b
2. Obtain the link to the door vault by entering   
   remi 6b6261721244591492597245de07dbef1626b472244f340614f9083c6  
   e683a7c046c0f0a46360f5b

|  |
| --- |
| vault\_door.zip |

<https://drive.google.com/open?id=1uhMuhCWxopIrPGDsHqG6pGxLseQg8T9N>

1. According to the creators, apply a reverse-engineering attack, to obtain 4354467b346c6c5f6a3030725f663030647a5f3472335f62336c306e675f325f6d335f6e3077217d stored in the hex-CTF.key file.
2. Convert the HEX to ASCII to obtain the flag:  
   CTF{4ll\_j00r\_f00dz\_4r3\_b3l0ng\_2\_**m3**\_n0w!}

*But we solved it with the backdoor way (below) instead. Sobbles.*

Even after I pleaded with the creators to explain the frontdoor way to me, I still don’t quite fully understand it and I guess this showed me that backdoors are honestly sometimes a better choice, even if it’s manual labour with obfuscated javascript...

**Backdoor-solution:**

Examine the app.js file after running it through [de4js](https://lelinhtinh.github.io/de4js/):

You will find a string, e)~/5ss\*p11k\*y11rm\*5k6\*w6s1f&\*9\*d6\*f1v$@ and a series of mappings:

|  |
| --- |
| \_0x5acb26['a'] = 'q';  \_0x5acb26['b'] = 'w';  \_0x5acb26['C'] = 'e';  \_0x5acb26['d'] = 'r';  \_0x5acb26['e'] = 't';  \_0x5acb26['f'] = 'y';  \_0x5acb26['G'] = 'u';  \_0x5acb26['h'] = 'i';  \_0x5acb26['i'] = 'o';  \_0x5acb26['j'] = 'p';  \_0x5acb26['K'] = 'a';  \_0x5acb26['l'] = 's';  \_0x5acb26['m'] = 'd';  \_0x5acb26['n'] = 'f';  \_0x5acb26['o'] = 'g';  \_0x5acb26['p'] = 'h';  \_0x5acb26['q'] = 'j';  \_0x5acb26['r'] = 'k';  \_0x5acb26['s'] = 'l';  \_0x5acb26['t'] = 'z';  \_0x5acb26['u'] = 'x';  \_0x5acb26['v'] = 'c';  \_0x5acb26['w'] = 'v';  \_0x5acb26['x'] = 'b';  \_0x5acb26['c'] = '^';  \_0x5acb26['g'] = '&';  \_0x5acb26['k'] = '(';  \_0x5acb26['y'] = 'n';  \_0x5acb26['z'] = 'm';  \_0x5acb26['{'] = '/';  \_0x5acb26['}'] = '@';  \_0x5acb26['\_'] = '\*';  \_0x5acb26['4'] = '5';  \_0x5acb26['0'] = '1';  \_0x5acb26['3'] = '6';  \_0x5acb26['2'] = '9';  \_0x5acb26['!'] = '$';  \_0x5acb26['T'] = ')';  \_0x5acb26['F'] = '~'; |

If you map it, you’ll obtain:

|  |
| --- |
| \_0x5acb26['C'] = 'e';  \_0x5acb26['T'] = ')';  \_0x5acb26['F'] = '~';  \_0x5acb26['{'] = '/';  \_0x5acb26['4'] = '5';  \_0x5acb26['l'] = 's';  \_0x5acb26['l'] = 's';  \_0x5acb26['\_'] = '\*';  \_0x5acb26['j'] = 'p';  \_0x5acb26['0'] = '1';  \_0x5acb26['0'] = '1';  \_0x5acb26['r'] = 'k';  \_0x5acb26['\_'] = '\*';  \_0x5acb26['f'] = 'y';  \_0x5acb26['0'] = '1';  \_0x5acb26['0'] = '1';  \_0x5acb26['d'] = 'r';  \_0x5acb26['z'] = 'm';  \_0x5acb26['\_'] = '\*';  \_0x5acb26['4'] = '5';  \_0x5acb26['r'] = 'k';  \_0x5acb26['3'] = '6';  \_0x5acb26['\_'] = '\*';  \_0x5acb26['b'] = 'w';  \_0x5acb26['3'] = '6';  \_0x5acb26['l'] = 's';  \_0x5acb26['0'] = '1';  \_0x5acb26['n'] = 'f';  \_0x5acb26['g'] = '&';  \_0x5acb26['\_'] = '\*';  \_0x5acb26['2'] = '9'; |

**Final flag:** CTF{4ll\_j00r\_f00dz\_4r3\_b3l0ng\_2\_**m3**\_n0w!}

This challenge was interesting but also incredibly frustrating. As my very first CTF challenge, I really had to learn a lot about CTFs as I learned to tackle this.