EAV-Secure Diffie–Hellman?

Christopher Wrogg, cwrogg, christheyankee, OPz qt

March 27 2021

Background

I wanted to introduce people to the idea of EAV-Security which we will discuss more in the theoretical side of the write-up. I already had plans for CPA-Secure Diffie–Hellman? at the time and as a result I decided to do EAV-Security with Diffie-Hellman as well. I figured it would be a good refresher for those who might have forgotten the nitty-gritty details of the protocol. I'm sorry that this challenge was a bit inaccessible for less seasoned CTF veterans. This challenge was hands down the topic of the vast majority of the PMs I received and for that I sincerely apologize. In hindsight I could have given some hints to make it more accessible.

The Theory

So what does it even mean for something to be EAV-Secure? It just means that it is secure in the presence of an eves-dropper. That might be self evident but lets formally define that a bit better. To be secure in the presence of an eve's dropper the output of the encryption function must be computationally indistinguishable from random. That means Eve must be unable to answer whether she is give output from uniform random bits or your encryption protocol. Before we get too into the weeds of theory, there are not many Diffie-Helman protocol that are EAV secure, least of all this one.

The Exploit

So from the title and description we can go ahead and guess (correctly) that what we have here is not EAV secure. In fact its not even computationally secure. p is abnormally small. I wanted to discourage brute forcing the discrete log so I chose the p large enough that you would have to be clever in how you filtered for a and small enough that you still could. Using a number sieve you can deduce the residue class of a. From there its just a matter of brute forcing which member of that class actually is a. The python script is included on the next page.

```
from Crypto.Util.number import long_to_bytes

p = 32090785453430065833482757911359568348 9

def exploit():
    a = 675140574589674474202795660911925983 01
    k = 0
    while k < 100000000:
        a += 32090785453430065833482757911359568348 8 * k
        k += 1
        flag = long_to_bytes(a)
        if flag.startswith(b"wsc"):
            print(k, flag.decode())
            break

if __name__ == "__main__":
    exploit()
```