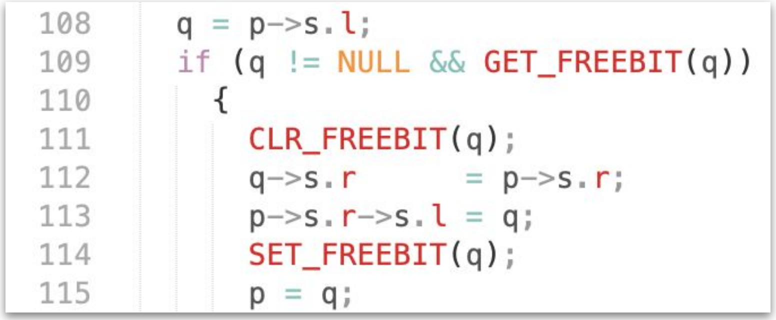
Section 4: Lab 1 Hints, Modular Arithmetic and 2DES

Administrivia

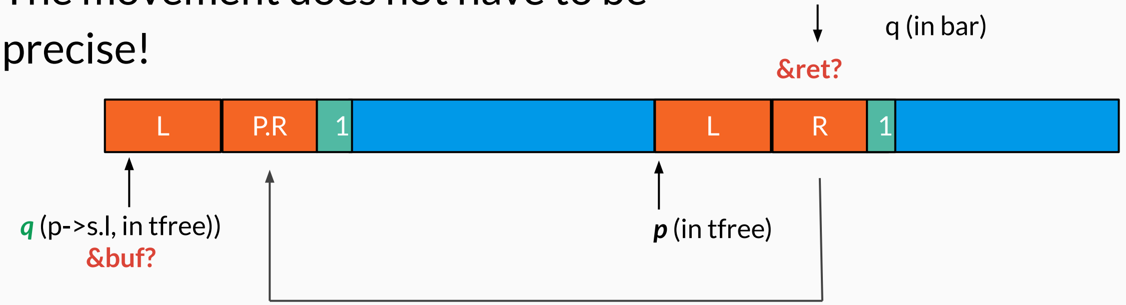
- Final deadline for lab1 is Wednesday, Oct 27 @ 11:59pm
  - Run the md5sum command on your last 4 exploits
  - Put the outputs in <netid>_<netid>_<netid>.txt
  - Submit on Canvas
- Homework 2 is out!
  - Hands-on work with cryptography
  - Individual assignment

Lab 1 Notes/Hints

- Sploit 5: See tfree from last section.
    - Make sure the free bit of the left chunk is set
    - The 2nd four bytes of ***q*** will be overwritten by line 112
    - How can you move past this?
        1. Point to an assembly instruction?
        2. Hardcode an instruction code?
        3. The movement does not have to be precise!

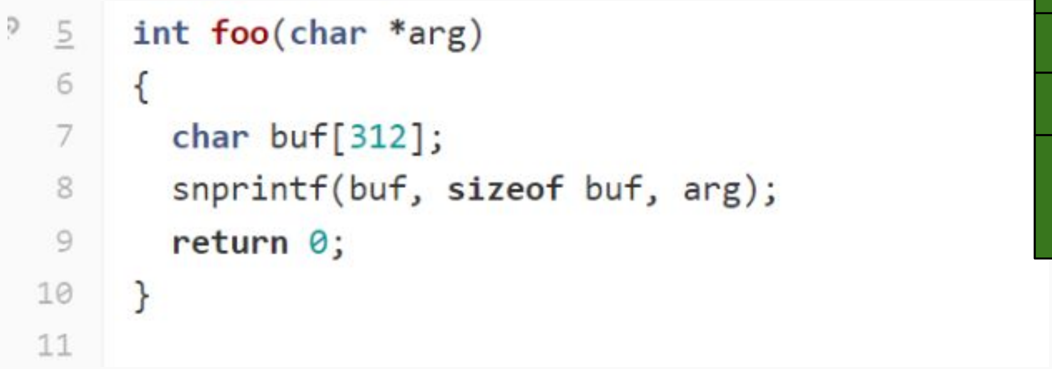


```
108      q = p->s.l;
109      if (q != NULL && GET_FREEBIT(q))
110        {
111          CLR_FREEBIT(q);
112          q->s.r       = p->s.r;
113          p->s.r->s.l = q;
114          SET_FREEBIT(q);
115          p = q;
```



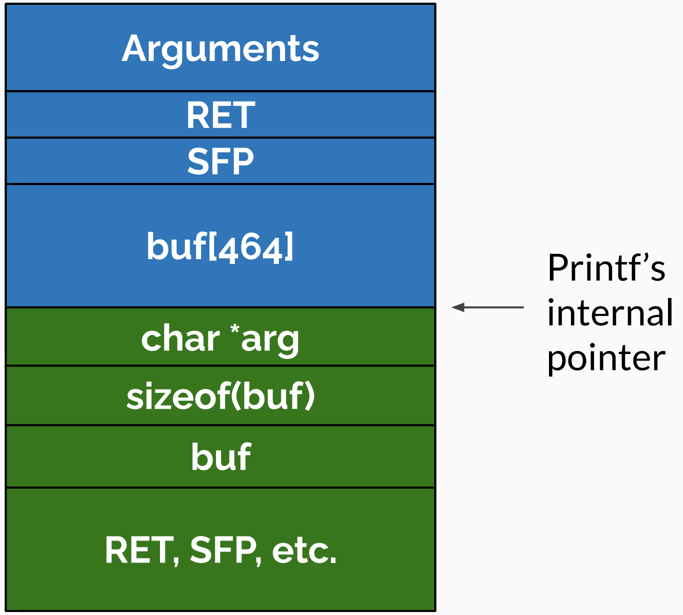
Lab 1 Notes/Hints

- Sploit 6: snprintf to a location.
    - Overwrite ret with %n (will need > 1)
    - Pad %u, %d, %x to get the value to write
    - %u, %d, %x, %n all expect an argument
    - Internal pointer begins after (char *) arg



```
int foo(char *arg)
{
  char buf[312];
  snprintf(buf, sizeof buf, arg);
  return 0;
}
```

Blue: foo's stack frame
Green: snprintf's stack frame



Additional arguments to `snprintf` would (normally) be after `arg`.

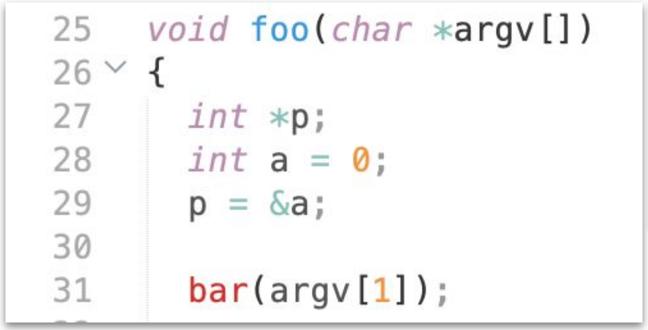
```
int snprintf ( char * s, size_t n, const char * format, ... );
```

Printf helpers

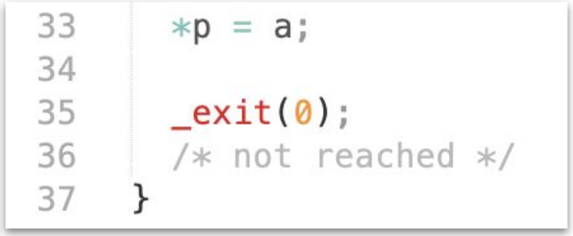
- %.[number]x (x with “number” decimal points)
  Ex: %.484x
- %hhn %hn: writes 1 and 2 bytes respectively

Lab 1 Notes/Hints

- Sploit 7: Similar to sploit 2.
  - However, you can’t overwrite RET since `foo` calls `_exit` before returning.
  - Where can you take over execution?
    - Hint: Think about `*p = a`
  - Try disassembling `_exit`

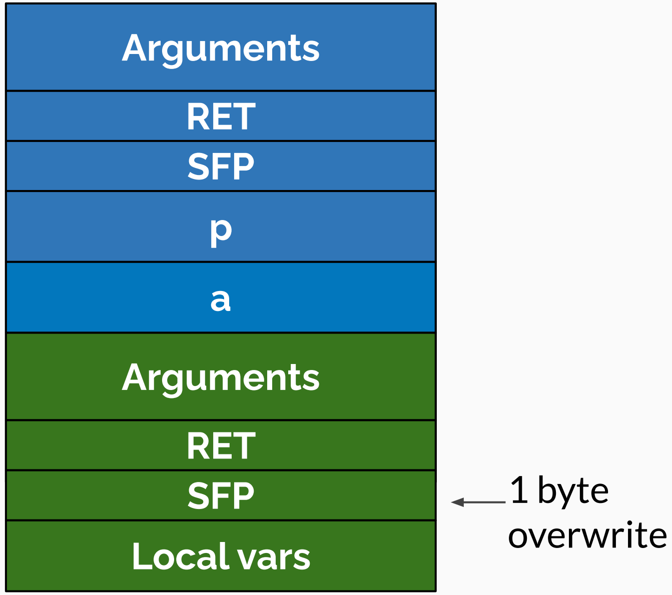


```c
void foo(char *argv[])
{
    int *p;
    int a = 0;
    p = &a;

    bar(argv[1]);
```



```c
    *p = a;

    _exit(0);
    /* not reached */
}
```

Blue: Foo’s stack frame
Green: bar’s stack frame



Program expects the stack to look like the layout of `foo` when returning from `bar`.

Homework 2 Pointers

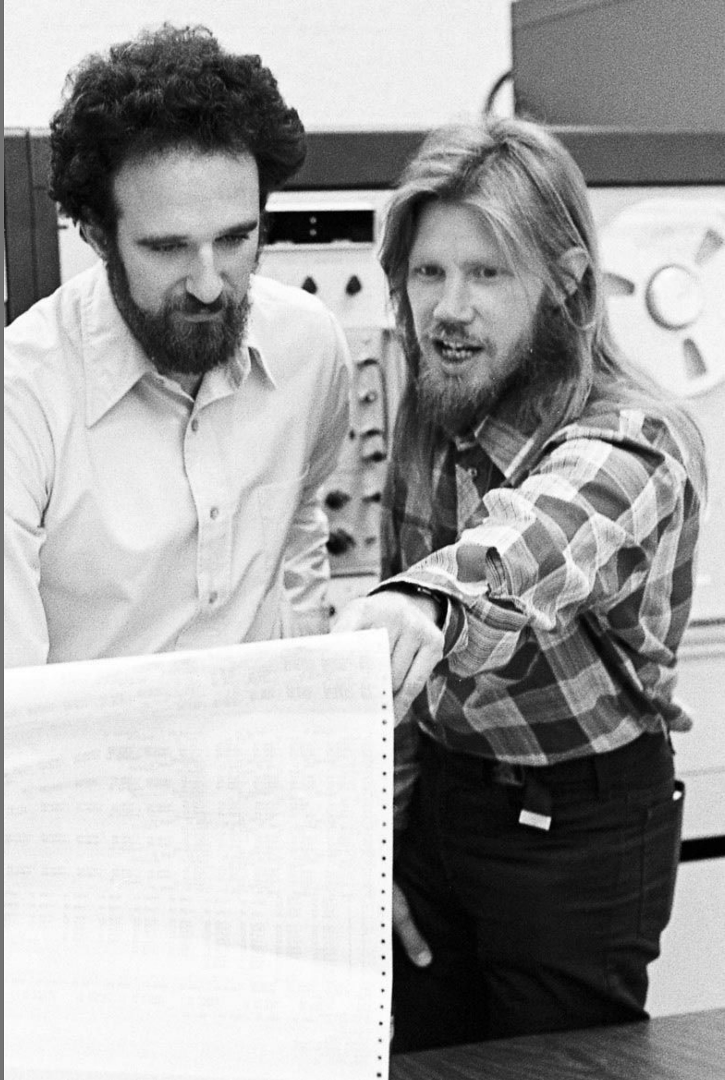
- RSA functionality (more next section)
- Block modes: CTR, ECB
- Diffie-Hellman (lecture, soon)
- Certificate Authorities (lecture, soon)
- Meet-in-the-middle vs 2DES (lecture 10)
    - Python quickstart guide: https://learnxinyminutes.com/docs/python/
    - Python DES package: https://pypi.org/project/des/

Modular Arithmetic

- The modulo:

  a mod *b*
  =
  the remainder of *a*÷*b*

- Many parts of cryptography depend on properties of modular arithmetic
- We'll talk more about it in lecture soon™ - public key cryptography, Diffie-Hellman Protocol (1976)



Modular Exponentiation

How would we compute something like this?

Let $p = 11$. Let $g = 7$.
Compute $g^{400} \bmod p$

$7^{400} \approx 1.09 \times 10^{338}$...

$$(a*b) \bmod \mathbf{p}$$

$$=$$

$$(a \bmod \mathbf{p} * b \bmod \mathbf{p}) \bmod \mathbf{p}$$

Q1

Let $p = 11$. Let $g = 10$.
Compute $g^1 \bmod p$, $g^2 \bmod p$, $g^3 \bmod p$, …, $g^{100} \bmod p$.

$$(a*b) \bmod \mathbf{p}$$
$$=$$
$$(a \bmod \mathbf{p} * b \bmod \mathbf{p}) \bmod \mathbf{p}$$

Q1 Solution

Let p = 11. Let g = 10.
Compute g^1 mod p, g^2 mod p, g^3 mod p, ..., g^{100} mod p.

10^1 mod 11 = 10   10^2 mod 11 = 1
10^3 mod 11 = (10^1 mod 11 * 10^2 mod 11) mod 11 = (10 * 1) mod 11 = 10
10^4 mod 11 = (10^2 mod 11 * 10^2 mod 11) mod 11= (1 * 1) mod 11 = 1
10^5 mod 11 = (10^1 mod 11 * 10^4 mod 11) mod 11 = (10 * 1) mod 11 = 10

…. Etc.

Creates cyclic group {10, 1}.

(**a***b*) mod **p**
=
(**a** mod **p** * **b** mod **p**) mod **p**

Q2

Let $p = 11$. Let $g = 7$.
Compute $g^1 \bmod p$, $g^2 \bmod p$, $g^3 \bmod p$, …, $g^{100} \bmod p$.

$$(a*b) \bmod p$$
$$=$$
$$(a \bmod p * b \bmod p) \bmod p$$

Q2 Solution

Let p = 11. Let g = 7.
Compute g^1 mod p, g^2 mod p, g^3 mod p, …, g^{100} mod p.

7^1 mod 11 = 7	7^2 mod 11 = 5	7^3 mod 11 = 2	7^4 mod 11 = 3
7^5 mod 11 = 10	7^6 mod 11 = 4	7^7 mod 11 = 6	7^8 mod 11 = 9
7^9 mod 11 = 8	7^10 mod 11 = 1		
7 ^11 mod 11 = 7	7^12 mod 11 = 5	…. Etc.	

Creates cyclic group {7,5,2,3,10,4,6,9,8,1}.
This is generating all positive integers < p.

(a*b) mod **p**
=
(a mod **p** * b mod **p**) mod **p**

Q3

Let $p = 11$. Let $g = 7$.
Compute $g^{400} \bmod p$, without using a calculator.

$$(a*b) \bmod p$$
$$=$$
$$(a \bmod p * b \bmod p) \bmod p$$

Q3 Solution

Note that 400 = 256 + 128 + 16.

7^2 mod 11 = 5
7^4 mod 11 = (7^2 mod 11 * 7^2 mod 11) mod 11 = 5 * 5 mod 11 = 3
7^8 mod 11 = (7^4 mod 11 * 7^4 mod 11) mod 11 = 3 * 3 mod 11 = 9
7^16 mod 11 = (7^8 mod 11 * 7^8 mod 11) mod 11 = 9 * 9 mod 11 = 4

… … …

7^128 mod 11 = (7^64 mod 11 * 7^64 mod 11) mod 11 = 3 * 3 mod 11 = 9
7^256 mod 11 = (7^128 mod 11 * 7^128 mod 11) mod 11 = 9 * 9 mod 11 = 4

Thus, 7^400 mod 11 = (7^256 mod 11 * 7^128 mod 11 * 7^16 mod 11) mod 11
= (4 * 9 * 4) mod 11
= 1 mod 11
= 1

Modular Exponentiation

$$a = g^{x} \bmod p$$

Given a, g, and p, what is x?

Calculate using a ***discrete logarithm*** - computationally very hard

- Why is this hard? There's not much we can learn from cyclical groups - very little is understood about the sequence of values
- You can base cryptographic schemes around the hardness of calculating the discrete logarithm, especially if you pick large values

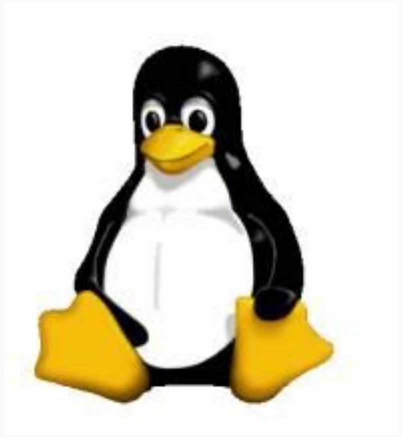
Thinking about encryption

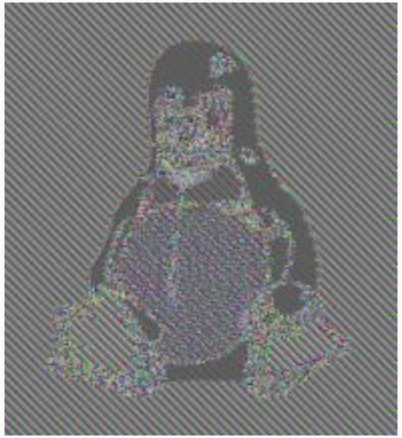
Which symmetric encryption mode would you use for the following situations? Why?

- You are going to send a small one-time command to fire to your nukes.
- You are living in the 1970s and want to send a long letter to your lover on ARPANET.
- Everything else (given the tools we've learned)

Thinking about encryption

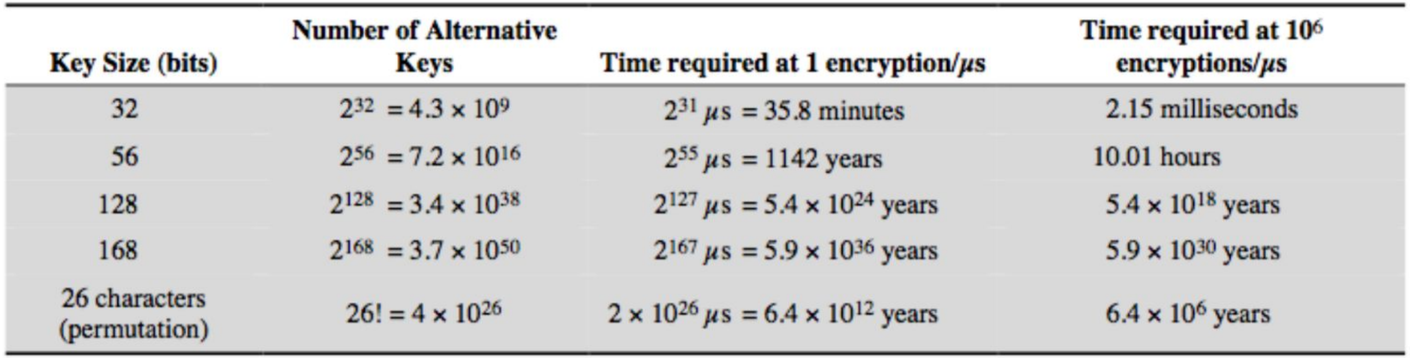
What is a flaw with ECB encryption?





DES and 56 bit keys

- 56 bit keys are quite short

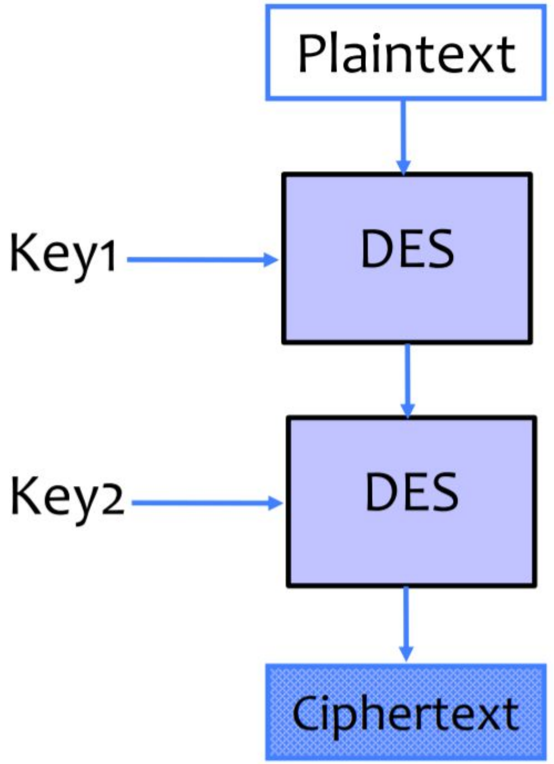


Key Size (bits)	Number of Alternative Keys	Time required at 1 encryption/μs	Time required at 10^6 encryptions/μs
32	$2^{32} = 4.3 \times 10^9$	$2^{31}\ \mu s = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55}\ \mu s = 1142$ years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127}\ \mu s = 5.4 \times 10^{24}$ years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167}\ \mu s = 5.9 \times 10^{36}$ years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26}\ \mu s = 6.4 \times 10^{12}$ years	6.4×10^6 years

- 1999: EFF DES Crack + distributed machines
  - < 24 hours to find DES key
- DES ---> 3DES
  - 3DES: DES + inverse DES + DES (with 2 or 3 diff keys)

2DES

- Key1 and key2 are 56-bit keys
- Adversary knows the plaintext and the ciphertext
- Strategy 1: brute force attack - 2^{112} possibilities
- Strategy 2: meet-in-the-middle attack - precompute 2 tables for Encrypt (P, Key1) and Decrypt (C, Key2) and find the matching output, $2^{56} * 2 = 2^{57}$ possibilities



Meet-in-the-middle attack

$$\begin{tikzcd} P \arrow[r, "\text{DES Key1}"'] & X \arrow[r, "\text{DES Key2}"'] & C \end{tikzcd}$$

K1	Encrypt(P, K1)
1	Y_1
2	Y_2
...	...
2^{56}	$Y_{2^{56}}$

Decrypt(C, K2)	K2
Z_1	1
Z_2	2
...	...
$Z_{2^{56}}$	2^{56}

If Y□ = Z□, We have found X. K1 = K□ and K2 = K□

Tips on HW2 Q9

- Shorter key length 2^{14}
- You are given a plaintext/ciphertext pair for finding the key, and another ciphertext to decrypt and obtain the message
- Use des package with the function provided to you

```python
from des import DesKey
def expandkey(val):
    if(val >= (2**14)):
        print("Key too large! Must fit in 14 bits")
        exit()
    k = val | (val << 14) | (val << 28) | (val << 42)
    return DesKey(bytearray.fromhex("{v:016X}".format(v=k)))
```

- Other functions that might be helpful from des:
  encrypt(plaintext), decrypt(ciphertext), bytearray.fromhex()

Good luck with the rest of lab 1!