CSCI 331:
Introduction to Computer Security

Lecture 11: Midterm Exam Review

Instructor: Dan Barowy

Williams

Announcements

- TA applications open; due by Oct 27.
- TA feedback survey Oct 27.

Announcements

- **Midterm exam**, in class, Thursday, Oct 19.
- Colloquium: **What I Did Last Summer (Research Edition)**, 2:35pm in Wege Auditorium.

Your to-dos

1. Study for **Thursday's exam**.
2. Project part 2, **due Sunday by 10pm**.

Person	*Topic 1*	*Topic 2*	*Topic 3*
Ben Wilen	Malware/viruses	XSS	MITM
David Goetze	Sandbox escape	timing attacks	SQL injection
Faisal Alsaif	Malvertising	SQL injection	Reflected XSS
Gregor Remec	Buffer overflows	SQL injection	DDoS
Jack Sullivan	MITM	Botnet/DDoS	Rootkits
Kit Conklin	Race conditions	Format string vuln	Clickjacking
Lee Mabhena	MITM	DoS	Credential stuffing
Michelle Wang	Clickjacking	XSS	Evesdropping
Zach Sturdevant	DoS	XSS	Side channel attacks
Ye Shu	Traffic confirmation attack	Use after free exploit	Privilege escalation
Sarah Fida	SQL injection	Directory traversal	Clickjacking

What topics?

Think about which topics you **do not feel confident about**. Take a few minutes and write them down on a piece of paper.

Everybody needs to **tell me something**.

Things we've covered

The C Programming Language

The C Programming Language

Basics

- Compilation using `gcc`.
- Warnings using `-Wall`
- Programs vs libraries
  - Build program with `-o` and specify name
  - Build library with `-c`

The C Programming Language

C Features

- The pointer as the basic unit of abstraction.
- `struct` as the basic unit of grouping.
- `typedef` as a way to give types useful names.
- Printing using `printf` and format specifiers.
- Memory as a resource that must be manually managed
  - Automatic ("local") memory, allocated on the stack
  - Manual memory, allocated on the heap using `malloc`.

The C Programming Language

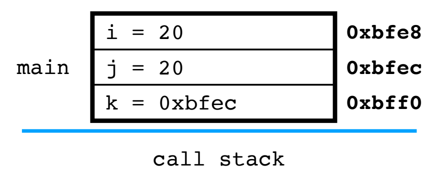
C Rules

0. Pointers are for **referring to** locations in **memory**.
1. When using a variable, **always** ask C to **reserve memory** for some **duration**.
2. **Always allocate** and **deallocate** long duration storage.
3. **Always initialize** variables.
4. **Watch out** for **off-by-one** errors.
5. **Always null-terminate** "C strings."

The C Programming Language

State Diagrams

```
#include <stdio.h>

int main() {
  int i = 10, j = 0, *k;
  k = &i;
  *k = 20;
  k = &j;
  *k = i;
  printf("i = %d,
          j = %d,
         *k = %d\n",
         i, j, *k);
→ return 0;
}
```



(state **just before** the line indicated by the **arrow** is executed)

The C Programming Language

State Diagram Rules

The Rules

1. Initialize diagram with empty stack and heap.
2. When a function is called, put a box on the stack, and label it with the function's name.
3. Put global variables outside the box.
4. Put local (automatic) variables inside the box, including function parameters.
5. Manage allocated variables on the heap.
   (a) `malloc` adds objects.
   (b) `free` removes objects.
6. As the function runs, update values.
7. Returning from a function pops the stack frame and, if the function returns a value, assigns it to the storage awaiting the return value.

Makefiles

```
program: c.c b.o a.o
tab → gcc -o program c.c b.o a.o
```

```
target: dep₁ … depₙ
tab → command
```

`command` should produce `target`.

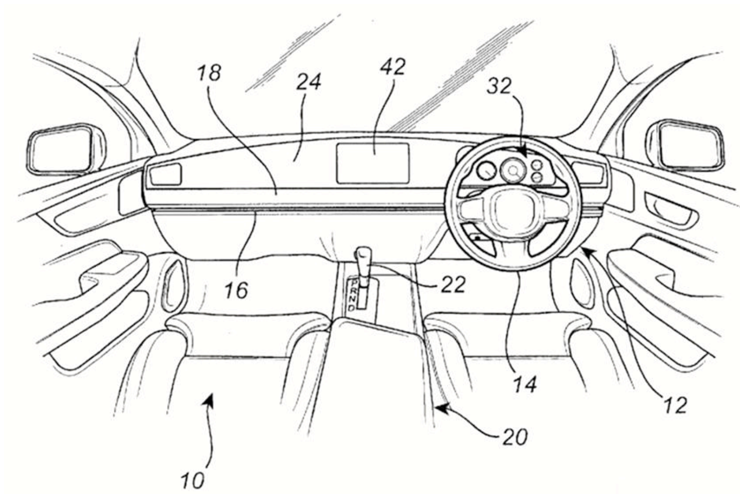
Makefiles

```
CFLAGS=-Wall -g

.PHONY: all
all: dictattack hashchain

database.o: database.h database.c
   gcc $(CFLAGS) -c database.c

crackutil.o: crackutil.h crackutil.c database.h
   gcc $(CFLAGS) -c crackutil.c

dictattack: crackutil.o database.o dictattack.c
   gcc $(CFLAGS) -o dictattack dictattack.c crackutil.o database.o -lmd
```

Libraries: static vs shared

- Static library: compile with `-c`
- Shared library: link with `-l<whatever>`

.h files are **interfaces**



Building with libraries

```
CFLAGS=-Wall -g

.PHONY: all
all: dictattack

database.o: database.h database.c
	gcc $(CFLAGS) -c database.c

crackutil.o: crackutil.h crackutil.c database.o
	gcc $(CFLAGS) -c crackutil.c

dictattack: crackutil.o database.o dictattack.c
	gcc $(CFLAGS) -o dictattack dictattack.c crackutil.o database.o -lmd

.PHONY: clean
clean:
	rm -f *.o
	rm -f dictattack
	rm -rf *.dSYM
```

static library (→ database.o)

shared library (→ -lmd)

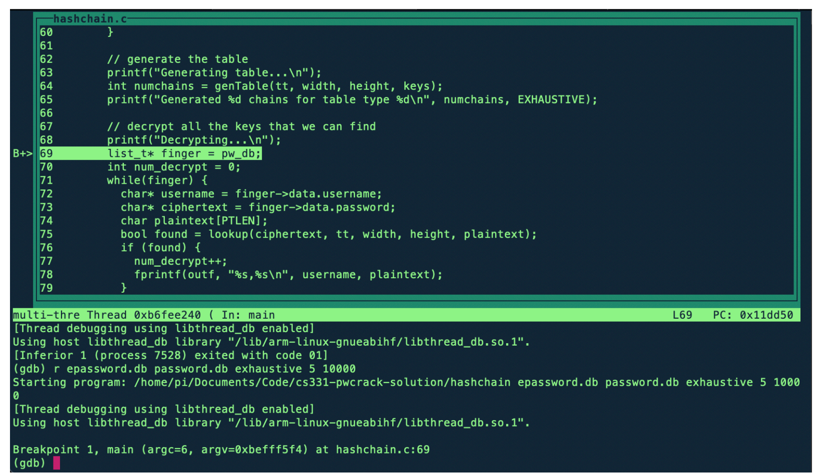
Finding memory errors with ASan

```
-g --fsanitize=address -static-libasan
```

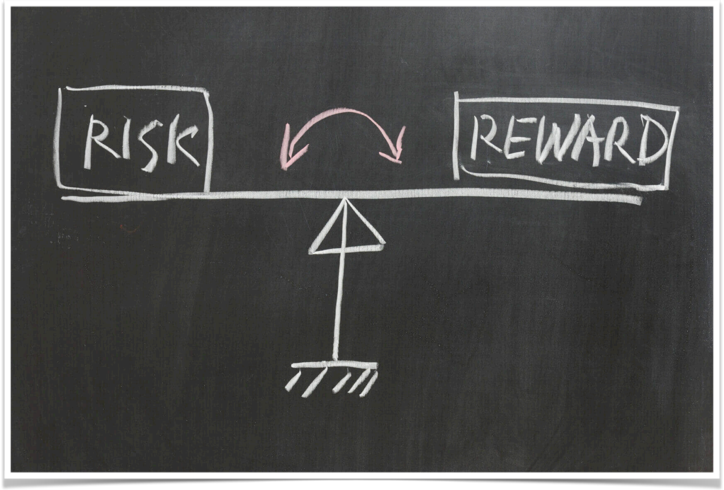
Kinds of memory errors:

- Segmentation fault
- Memory leak
- Out-of-bounds read
- Buffer overflow (OOB write)
- Use-after-free
- Uninitialized read

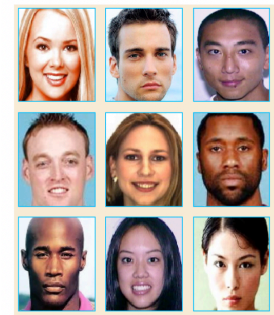
Debugging with gdbtui



Security as a tradeoff



Security as a tradeoff



e.g., memorability vs guessability

Security as a tradeoff

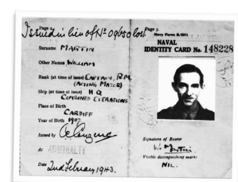
How to quantify risk-reward tradeoff

- Enumerate potential vulnerabilities
- Assign exploit probabilites
- Estimate cost of exploit
- Compute expected cost
- Rational expenses for mitigation do not exceed the expected cost of the exploit

Security properties

Confidentiality

Integrity



Authenticity

Availability

Security properties

Non-repudiation

Crypto!

Encryption is the **process of encoding a message** so that it can be read only by the sender and the **intended recipient**.

- A **plaintext** p is the original, unobfuscated data. This is information you want to protect.
- A **ciphertext** c is encoded, or encrypted, data.
- A **cipher** f is an algorithm that converts **plaintext** to **cipertext**. We sometimes call this function an **encryption function**.
  - More formally, a cipher is a function from plaintext to ciphertext, $f(p)=c$. The properties of this function determine what kind of encryption scheme is being used.
- A **sender** is the person (or entity) who enciphers or encrypts a message, i.e., the party that converts the plaintext into cipertext. $f(p)=c$
- A **receiver** is the person (or entity) who deciphers or decrypts a message, i.e., the party that converts the ciphertext back into plaintext. $f^{-1}(c)=p$

Cryptographic hash functions

Suppose we have:

$f(p)=c$, a **cipher** that maps **plaintexts** to **ciphertexts**; in this case, a **hash function**.

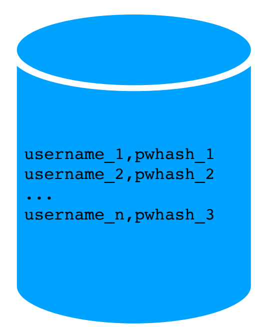
> Because f is a hash function, there is **no inverse function** such that $f^{-1}(f(p))=p$.

A cryptographic hash function is **bitwise independent**, meaning that seeing one or more bits of output **does not help an attacker** predict the values of the remaining outputs.

Brute Force Password Attacks

Online, using a pseudoterminal.

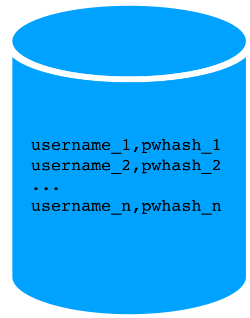
Offline, using a password cracking algorithm.



Offline password database attacks

- Random guessing attack
- Enumeration attack
- Dictionary attack
- Precomputed hash chain attack
- Rainbow table attack

Random guessing: complexity (one pw)



m = # of possible passwords

p = probability that random guess is correct

= 1/**m**

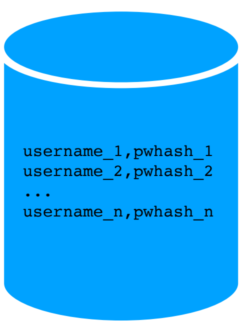
X = # guesses until success

E[**X**] = (1-**p**)/**p** (geometric dist)

= **m** - 1

O(**m**) average **per pw** O(**mn**) average for **all pw**

Enumeration: complexity



m = # of possible passwords

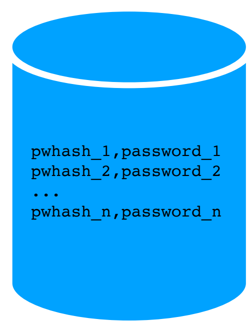
Average guesses to find **one pw**:

O(**m**/2)

Average guesses to find **all pw**:

O(**n** x **m**/2)

Dictionary attack: complexity



m = # of possible passwords

Time to compute dictionary:

O(**m**)

Time to lookup **one pw**:

O(**log m**)

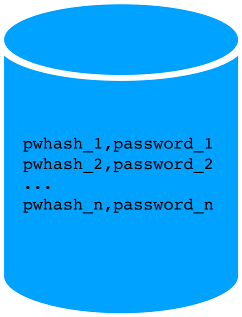
Time to lookup **all pws**:

O(**n log m**)

Space needed:

O(**m**)

PCHC/rainbow attack: complexity



m = # of possible passwords

Time to compute data structure:

O(**m**)

Time to lookup **one pw**:

O(**k**)

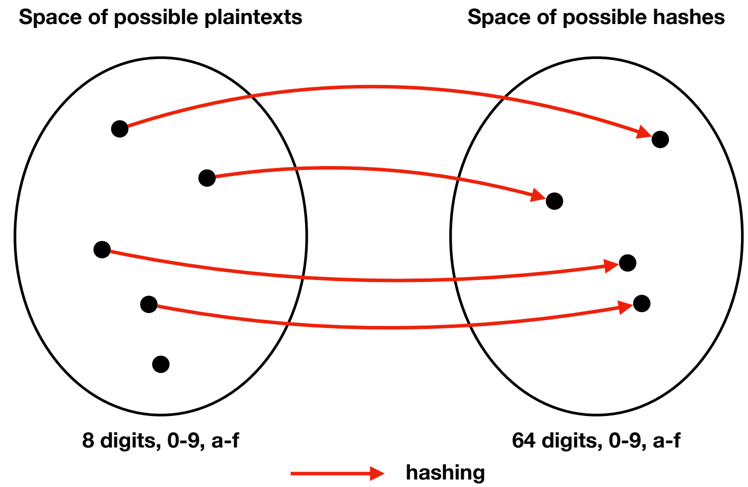
Time to lookup **all pws**:

O(**mk**)

Space needed:

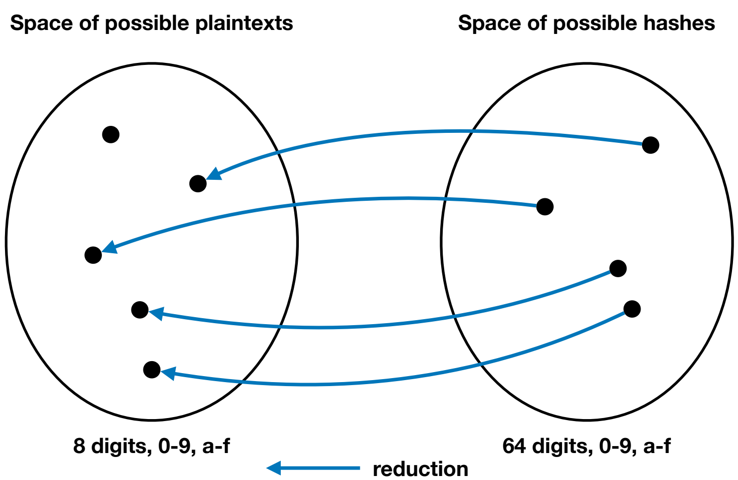
O(**m**/**k**)

Hash function



plaintext: "9a55302d" **ciphertext:** "4651f1799e5e36c878f3d980c59e94ae"

Reducer function

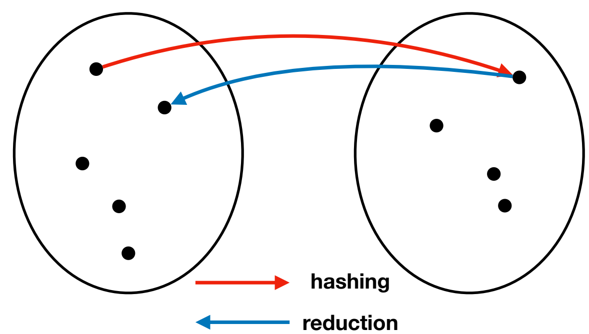


ciphertext: "4651f1799e5e36c878f3d980c59e94ae" **plaintext:** "4651f179"

Reducer function properties

A reducer $r(c)=p$ only needs to satisfy a couple properties.

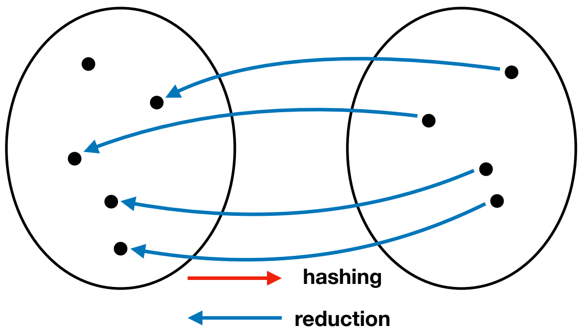
1. A reducer's output, p, should map to the same domain as the *input* of the hash function, $f(p)=c$ (i.e,. plaintexts)



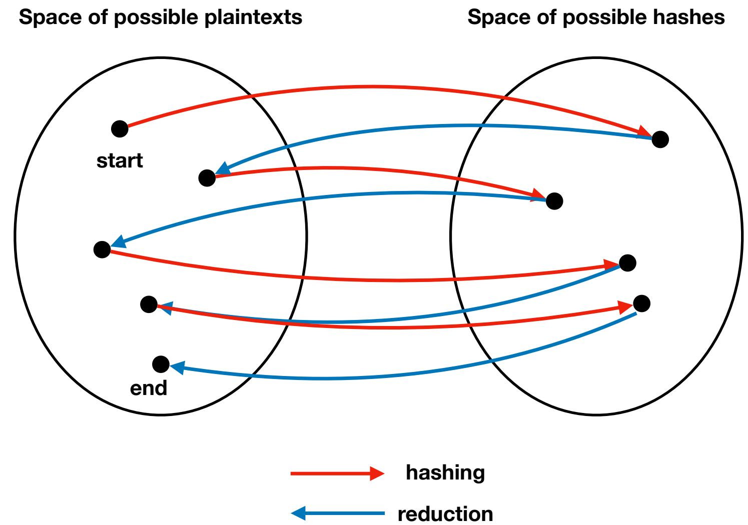
Reducer function properties

A reducer $r(c)=p$ only needs to satisfy a couple properties.

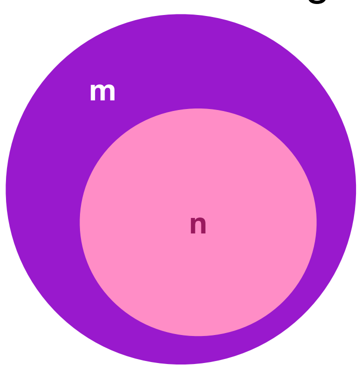
2. All plaintexts should be selected, given the space of ciphertexts, with equal probability.



Hash chain



Hashes are guaranteed to collide

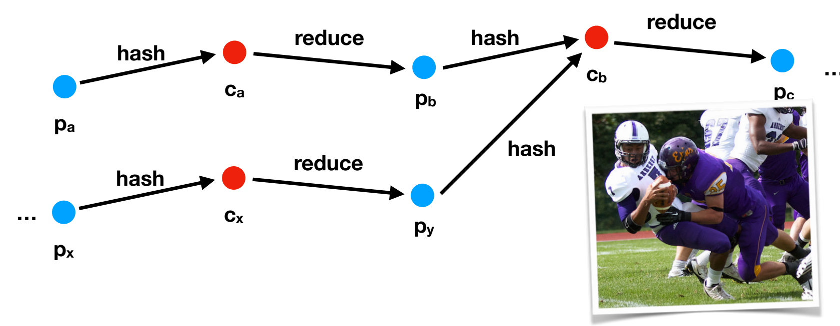


m: # of passwords **n**: # of hashes

If **m > n**, we know that **at least (m-n)/m** must collide.

"pigeonhole principle"

Collisions in a hash chain

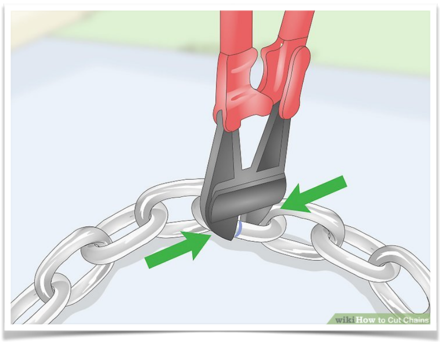


After the **collision**, the chain "**loops**."

Collisions prevent us from enumerating the **entire space**!

Hash chain of length k

We are going to chop up our long chain into **smaller chains** of length **k**.



Store only **start** and **end**

```
start, end
pm   , pm-3
…
p5   , p3
p3   , p1
```

Store it **backward**

```
end, start
pm-3  , pm
…
p3   , p5
p1   , p3
```

Hash function lookup table:

plaintext	Hash of plaintext
♥♥♥♥	4A7D1ED414474E4033AC29CCB8653D9B
♥♥♥★	25BBDCD06C32D477F7FA1C3E4A91B032
♥♥★♥	FC1198178C3594BFDDA3CA2996EB65CB
♥♥★★	AE2BAC2E4B4DA805D01B2952D7E35BA4
♥★♥♥	DB2F40F24260BC41DB48D82D5E7ABF1D
♥★♥★	814F06AB7F40B2CFF77F2C7BDFFD3415
♥★★♥	2A66ACBC1C39026B5D70457BB71B142B
♥★★★	7D7C45B9A935CF9D845FC75679A41559
★♥♥♥	A9B7BA70783B617E9998DC4DD82EB3C5
★♥♥★	B8C37E33DEFDE51CF91E1E03E51657DA
★♥★♥	1E48C4420B7073BC11916C6C1DE226BB
★♥★★	7F975A56C761DB6506ECA0B37CE6EC87
★★♥♥	1E6E0A04D20F50967C64DAC2D639A577
★★♥★	C6BFF625BDB0393992C9D4DB0C6BBE45
★★★♥	2CBCA44843A864533EC05B321AE1F9D1
★★★★	B59C67BF196A4758191E42F76670CEBA

```
func reducer(c,i):

Convert the ith hexadecimal
digit of c into a plaintext
using the following table:
```

hex	plaintext
0	♥♥♥♥
1	♥♥♥★
2	♥♥★♥
3	♥♥★★
4	♥★♥♥
5	♥★♥★
6	♥★★♥
7	♥★★★
8	★♥♥♥
9	★♥♥★
A	★♥★♥
B	★♥★★
C	★★♥♥
D	★★♥★
E	★★★♥
F	★★★★

Find the first three rainbow chains of length 3.

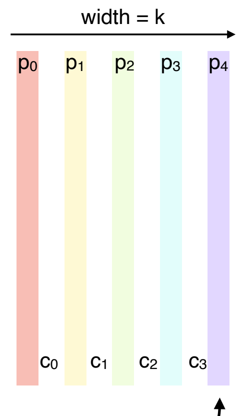
First three rainbow chains

♥♥♥♥ —h→ 4A7D —r_0→ ♥★♥♥ —h→ DB2F —r_1→ ★♥★★ —h→ 7F97 —r_2→ ★♥♥★

♥♥♥★ —h→ 25BB —r_0→ ♥♥★♥ —h→ FC11 —r_1→ ★★♥♥ —h→ 1E6E —r_2→ ♥★★♥

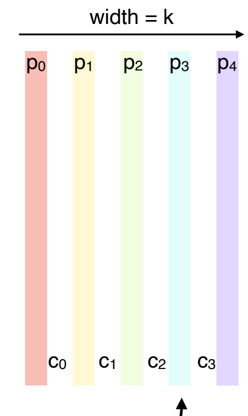
♥♥★♥ —h→ FC11 —r_0→ ★★★★ —h→ B59C —r_1→ ♥★♥★ —h→ 814F —r_2→ ♥★♥♥

end	start
★♥♥★	♥♥♥♥
♥★★♥	♥♥♥★
♥★♥♥	♥♥★♥



I hypothesize that c reduces to p_4

What reducer should I use? `reduce(c,3)`



I hypothesize that c reduces to p_{k-2}

What reducer should I use? `reduce(c,2)`

Then: `reduce(c,3)`

Rainbow table (for first 3 chains)

end	start
★♥♥★	♥♥♥♥
♥★★♥	♥♥♥★
♥★♥♥	♥♥★♥

Decrypt `FC11`.

Hypothesis: `FC11` is the third link in the chain.

`FC11` $\xrightarrow{r_2}$ ♥♥♥★ Is ♥♥♥★ an **end**? **No.**

Hypothesis: `FC11` is the second link in the chain.

`FC11` $\xrightarrow{r_1}$ ★★♥♥ $\xrightarrow{h}$ `1E6E` $\xrightarrow{r_2}$ ♥★★♥ Is ♥★★♥ an **end**? **Yes.**

Decrypt from **start** ♥♥♥★:

♥♥♥★ $\xrightarrow{h}$ `25BB` $\xrightarrow{r_0}$ ♥♥★♥ $\xrightarrow{h}$ `FC11`

plaintext

Countermeasures Against Cracking Attacks

- Password salts.
- Uniformly-distributed passwords.
- Two-factor authentication.
- Last-known IP address.
- Make hashing expensive.

Key Stretching

Key stretching is a technique used to make password decryption attacks **computationally expensive**. Unlike an ordinary user, an attacker must invoke a hash function many times. Key stretching **amplifies the cost of a hash function** using a **stretch factor s**.

$f^s(p) = c^s$ is an iterated hash function, where

$$f^1(p) = f(p) = c^1$$
$$f^2(p) = f(f(p)) = c^2$$
$$f^3(p) = f(f(f(p))) = c^3$$
$$\ldots$$
$$f^n(p) = c^n$$

Practice exam solutions

Q&A

Recap & Next Class

Today we learned:

Midterm review

Next class:

Midterm exam