Password Auditing

Rainbow Tables

What Are Rainbow Tables?

- Precomputed Table
- Used for reversing cryptographic hash functions
  - Commonly, cracking passwords
- Faster alternative to brute-forcing

A Bit of History

- Originally proposed by Martin Hellman in 1980
  - *A Cryptanalytic Time-Memory Trade-Off*
- Further refined by Philippe Oechslin in 2003
  - *Making a Faster Cryptanalytic Time-Memory Trade-Off*

So, Some Cracking Approaches

- The easy way to crack a hash is through brute forcing
  - We actually compute the hash of all plaintexts one-by-one.
    - This takes time. Lots of CPU time.
- If we have TONS of storage space, we could just compute ALL of the hashes before we need them.
  - Instant lookup process, but uses lots of storage space.

How Do Rainbow Tables Help?

- A sort of middle ground between those two techniques.
- Computing is done in advance.
- Hashes are stored in the rainbow table.
- Hashes can be looked up, but not necessarily instant – but we save on storage space here.

Pros/Cons TL;DR?

- Use less processing time than a brute-force attack
  - But more storage
- Uses less storage than a simple lookup table with one entry per hash
  - But more processing time

Rainbow Tables

- To store every single 7 character password possible, it would use ~417TB of space.
- A rainbow table would need around 80GB of space, depending on generation options.

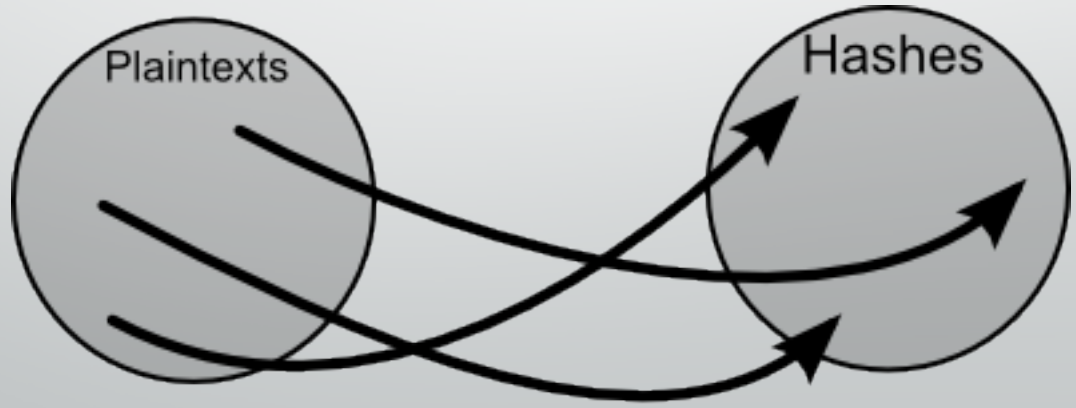
- Middle Ground

How Do Rainbow Tables Work?

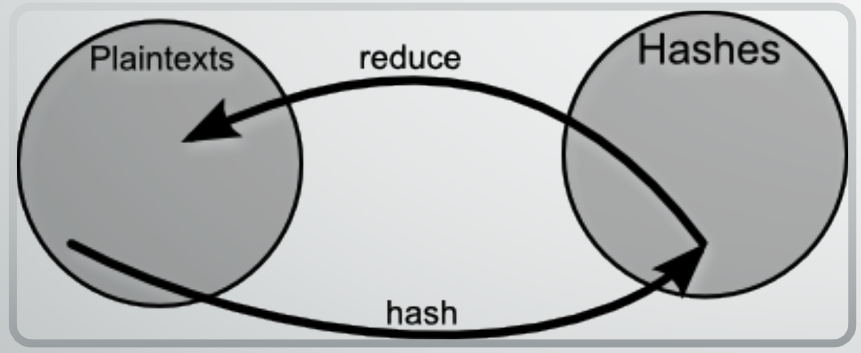
- Good article here: http://kestas.kuliukas.com/RainbowTables/
  - Basis of the upcoming discussion

Hash Functions

- A hash maps a plaintext (password) to a "garbage" value (hash), and it shouldn't be reversible.
- To figure out the plaintext, we'd either…
  - Hash each plaintext one at a time in real time
  - Hash each plaintext one at a time ahead of time and store it



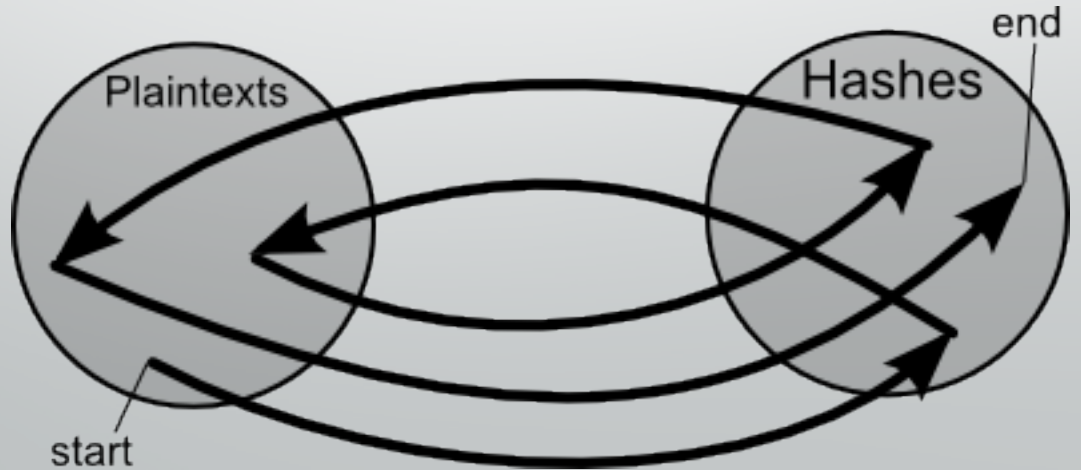
Reduction Function



- This can get a bit confusing
- Does the reverse of a hash (mapping hashes to plaintexts)
- If you take the hash of a plaintext, and take the reduction of the hash, it doesn't give you the original, but some other plaintext.
  - Reduction can be anything. Maybe taking the first 8 chars from the hash, and hashing those.

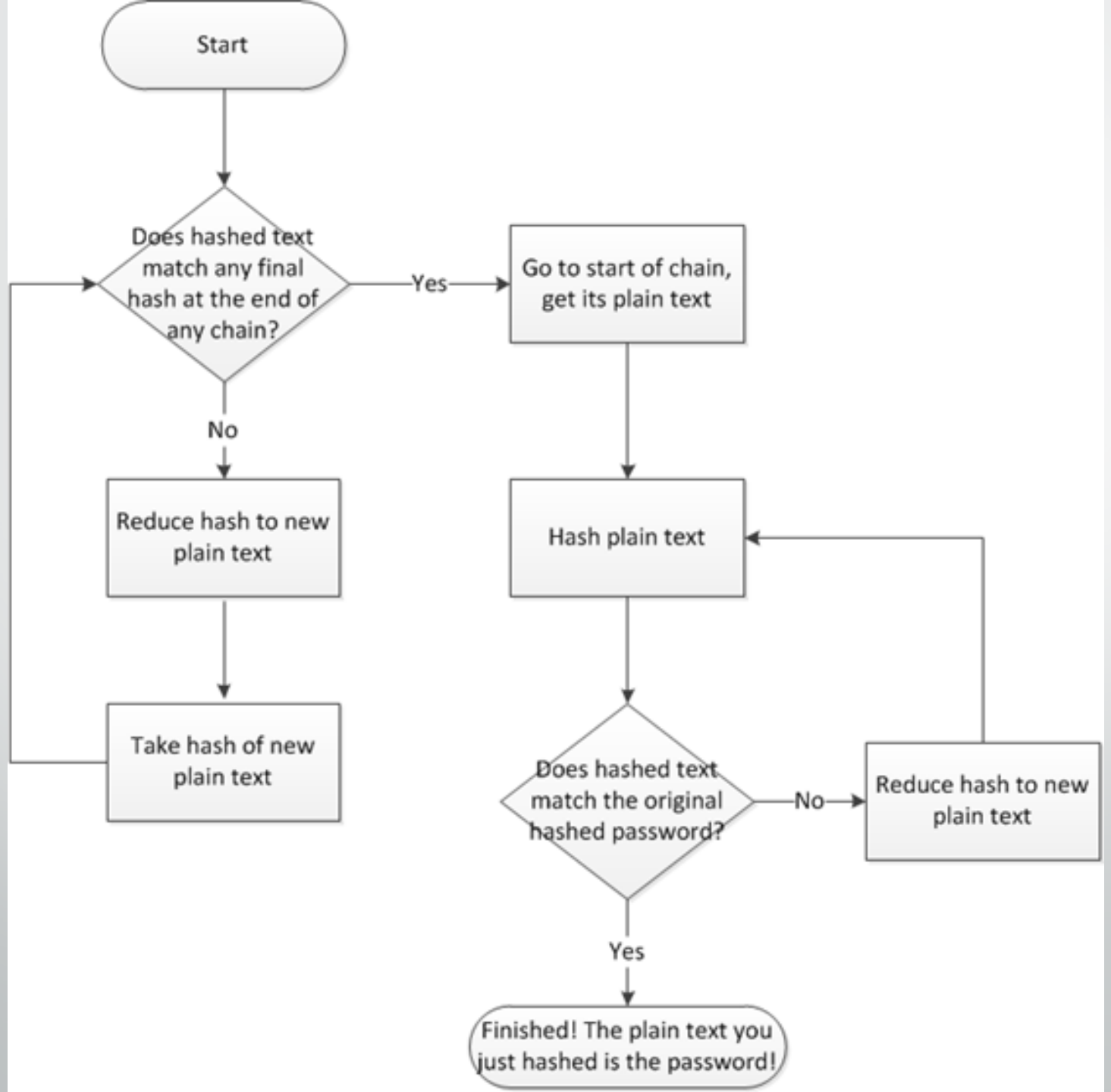
Reduction in Action

- Suppose you want the first 8 characters because that's what many passwords are and you want it in an MD5
- $hash_{MD5}$(12345678) = 25d55ad283aa400af464c76d713c07ad
- $hash_{MD5}$(25d55ad2) = 5c41c6b3958e798662d8853ece970f70
- $hash_{MD5}$(5c41c6b3) = 4e3d15bea09e7455aa362d3cf89838fc
- $hash_{MD5}$(4e3d15be) = f439b57135045b9d365dd867fca1e316



Reduction Explained

- You pick a point to stop at (let's go with 100,000 hashes)
- After the last reduction, you have a chain of 100,000 hashes
  - For example, the last one is 78ba340e23a492704f63e5239be65495
- What the rainbow table stores is:
  - 12345678 –> f439b57135045b9d365dd867fca1e316
  - Just the starting plaintext, and the ending hash
- This represents 100,000 hashes in a very small line of text.


Start
Does hashed text match any final hash at the end of any chain?
Yes
Go to start of chain, get its plain text
No
Reduce hash to new plain text
Hash plain text
Take hash of new plain text
Does hashed text match the original hashed password?
No
Reduce hash to new plain text
Yes
Finished! The plain text you just hashed is the password!

Rainbow Crack

- A general purpose implementation of Oechslin's technique
- Software to do this…
  - http://project-rainbowcrack.com
  - Some tables available for download there too
  - Also: http://ophcrack.sourceforge.net/tables.php

Rainbow Crack

- Three tools are included in RainbowCrack that must be used in sequence
  - rtgen
    - Generates the rainbow tables
  - rtsort
    - Sorts the rainbow tables generated by rtgen
  - rtcrack
    - Table lookup process
    - “hash cracking”

Installing RainbowCrack

- Already Installed in Kali
- …well that was easy
- Otherwise can download from http://project-rainbowcrack.com

rtgen

- Use rtgen to generate rainbow tables
- Requires several parameters to generate a rainbow table
  - rtgen [hash_algorithm] [charset] [plaintext_len_min] [plaintext_len_max] [table_index] [chain_len] [chain_num] [part_index]
- /usr/share/rainbowcrack/charset.txt

rtgen Example

- rtgen lm alpha-numeric-symbol14-space 1 7 0 2400 33554432 0
  - lm = type of hash (LanMan)
  - alpha-numeric-symbol14-space = character set
  - 1 = minimum characters (password length)
  - 7 = maximum characters (password length)
  - 0 = number of the table
  - 2400 = size of the table (number of chains in a table)
    - Higher number here increases success rate, but increases generation time
  - 33554432 = number of chains in a table
    - Higher number here increases the success rate, but increases the overall size of the table.
  - 0 = appended to the file name

rtgen

- It'll be worth it to do some investigating and benchmarking on this
  - Longer chains take more time to generate
  - Too long chains, too many tables, will take too long to crack to be useful.
- The values you should select depend on what you need. To do it right, you should research more into the time-memory tradeoff algorithm

Your Turn!

- Let's start generating tables!
- rtgen ntlm numeric 1 3 0 2400 100000 0
  - This should finish in under a minute. Ish.
  - Creates a 1.6M file
- Outputs an .rt file in /usr/share/rainbowcrack
  - Ntlm_numeric#1-3_0_2400x10000_0.rt
    - Don't rename this!

Create a few more tables

- ./rtgen ntlm numeric 1 3 1 2400 100000 0
- ./rtgen ntlm numeric 1 3 2 2400 100000 0
- ./rtgen ntlm numeric 1 3 3 2400 100000 0
- ./rtgen ntlm numeric 1 3 4 2400 100000 0

rtsort

- Sort the new chains in the new tables
- Each table is sorted individually
- Makes table lookup more efficient
- DO NOT INTERRUPT!

Your Turn to Sort

- rtsort <filename>

rcrack

- Used to lookup the rainbow tables
- Only accepts sorted tables
- To crack a single hash, use the –h switch followed by the hash.

rcrack

- Generate a hash here: https://www.onlinehashcrack.com/hash-generator.php
- rcrack /usr/share/rainbowcrack/*.rt –h your_hash
  - Or rcrack . –h your_hash if you're already in that dir.
- 9D864475D58B3C733A243D5CC873E73C

rcrack

- To crack multiple hashes, put them all in a text file, one hash per line
- Use the –l switch to specify the hash file

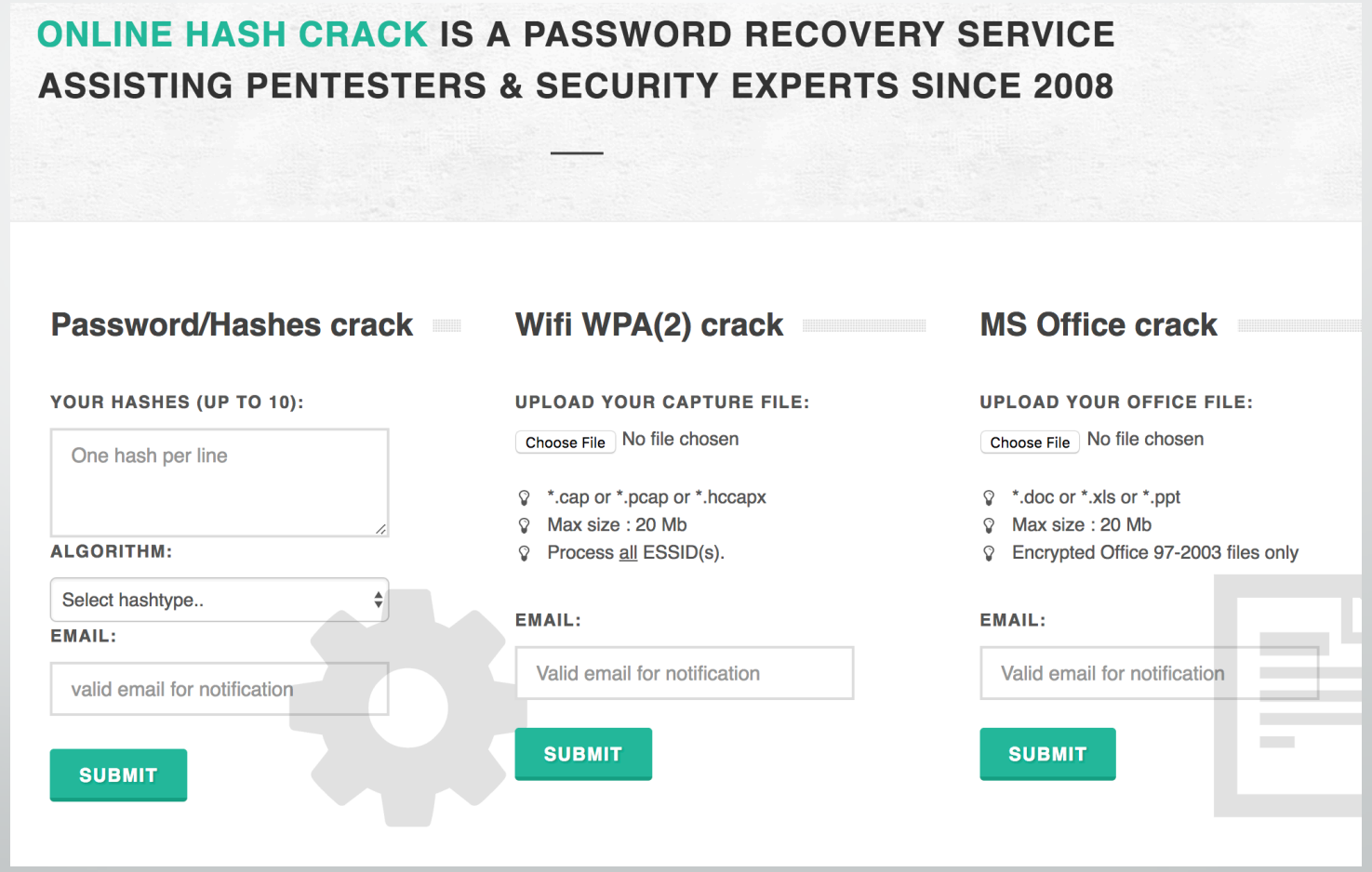
- rcrack . –l hash_file

Rainbow Tables

- Very powerful
- Will need to dedicate a lot of time and resources to build good tables
  - Hardware and time
    - Consider multiple machines
    - Use a UPS
- Do your homework!
  - Nothing worse than spending months building bad or inefficient tables

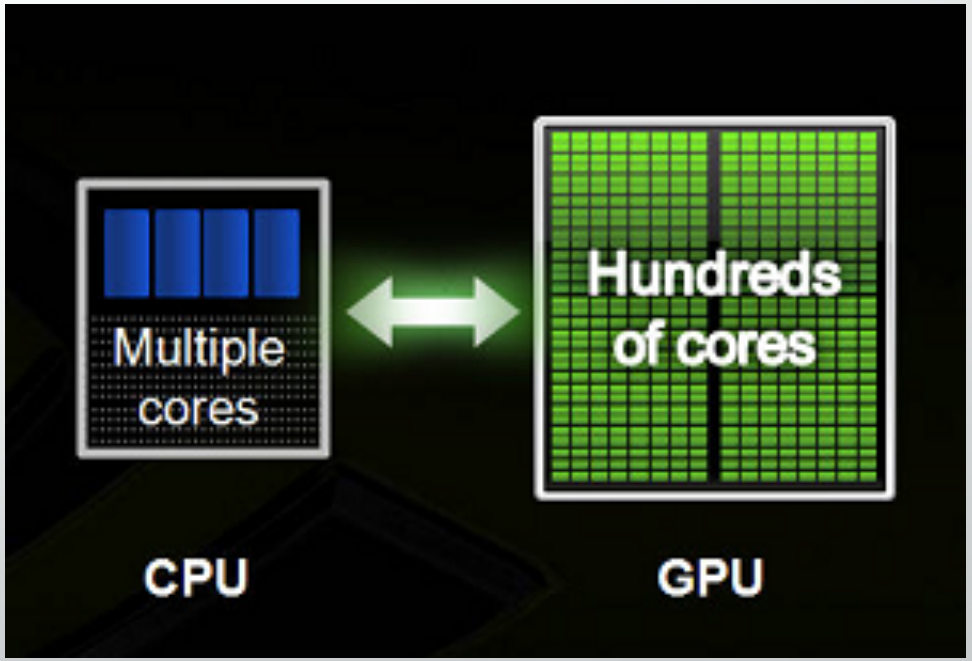
Another Option?

- onlinehashcrack.com



GPU vs. CPU

- CPUs have a few cores with lots of cache memory that can handle a few software threads at the same time.
- GPUs have hundreds or thousands of cores that handle thousands of threads simultaneously.



GPU vs. CPU

- GPUs are highly specialized in number crunching, something that graphics processing desperately needs
- Multiple GPUs can be employed for one single task
- GPUs are fast, but designed for specific applications.


DSU


SARCASTIC
COMMENT

Some Challenges… And Defenses

- What about non-English passwords?
- Those that use special character sets
  - Wingdings anybody?
  - ✠♓■♑♎♓■♑’♦ ♋■♲♌❑♎♲✍
- Changing passwords
- Locking accounts out

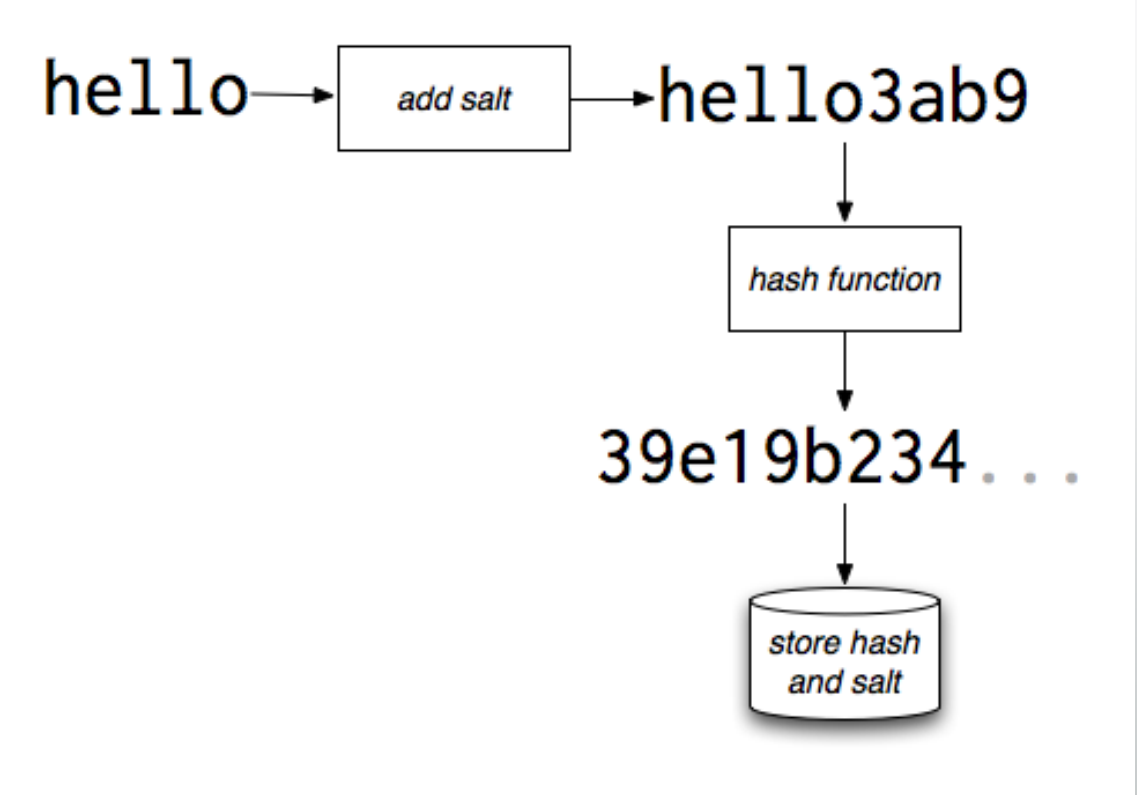
Defense Method – More Iterations

- Running a hash through the hashing algorithm multiple times
- Running a password through SHA256, for example, then feeding that hashed output back through the algorithm 500 additional times.
  - Would make wordlist and brute force attacks slower.

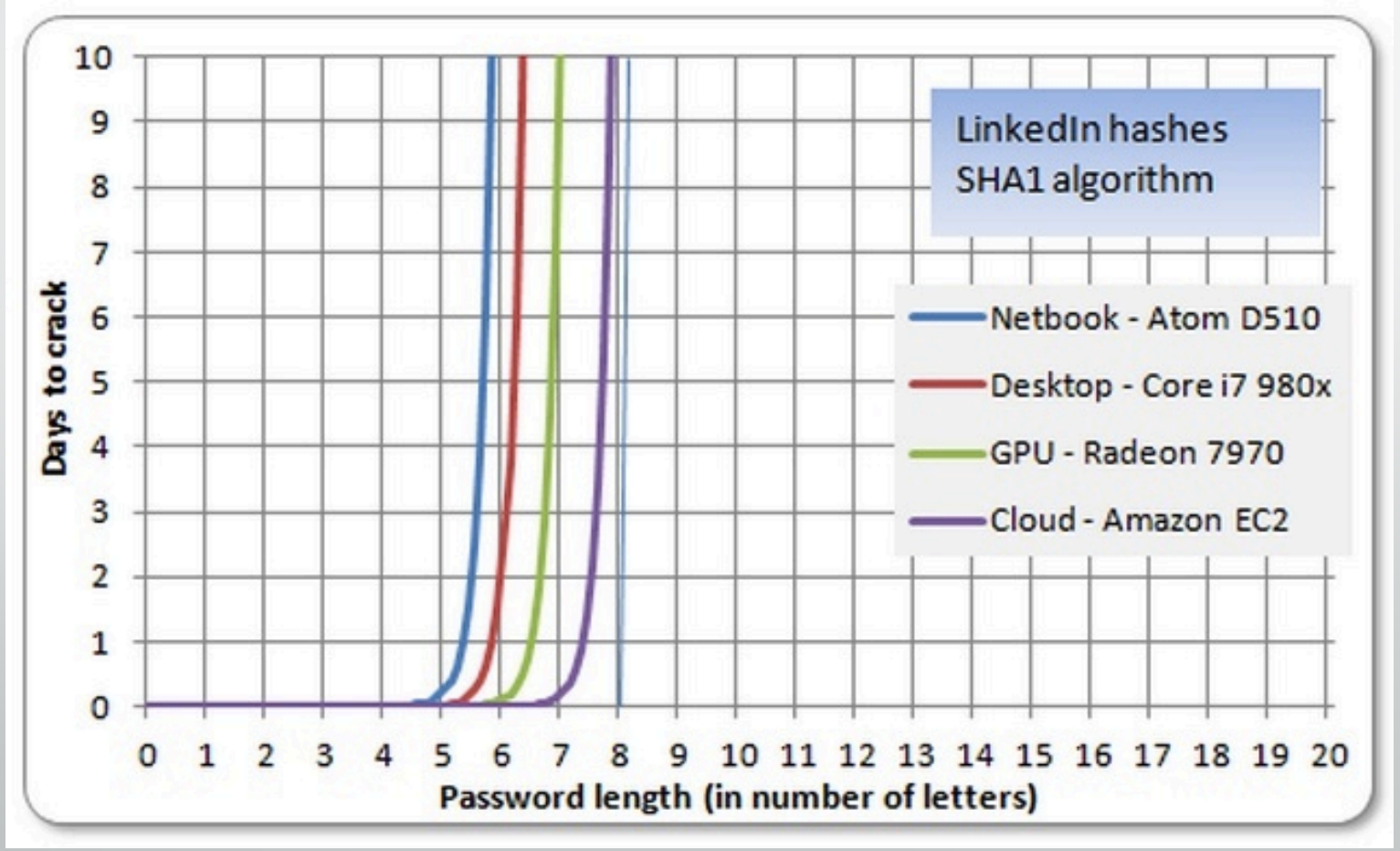
Defense Method - Salting

- Salting attempts to defeat a table lookup by adding several characters to the password before hashing it
- Characters don't have to remain secret – though then an attacker could build a table with those characters
- Salting thwarts cracking techniques that rely on rainbow tables

Salting

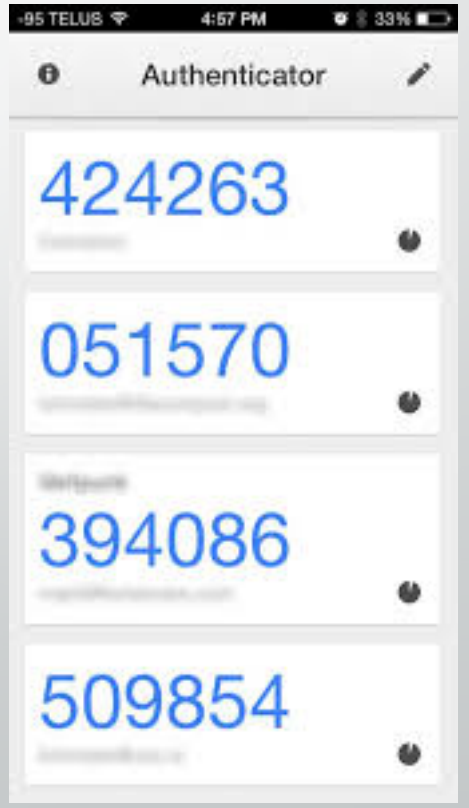

hello
add salt
hello3ab9
hash function
39e19b234 . . .
store hash and salt

Defense Method - Length

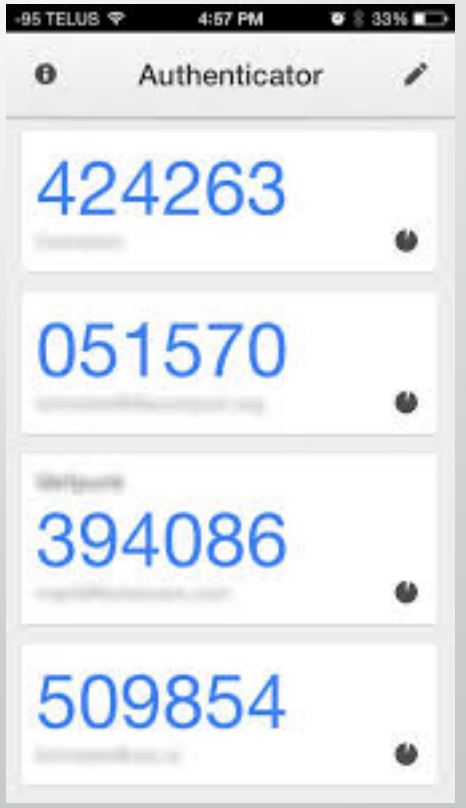


Defense Method – Two Factor Authentication

- Something you **know**
- Something you **have**
- Something you **are**



Two Factor Authentication



It might be easier just to use Kali_iNet

In Class Lab

- Helpful Hints: Previous Lecture and http://www.project-rainbowcrack.com/generate.pdf
- Choose a hashing algorithm to crack (LM, NTLM, SHA are good if you don't know which to choose)
- Use: https://www.onlinehashcrack.com/hash-generator.php to generate a small hash containing the 1st 3 letters of your first name (i.e. Cod)
- Use rtgen to create a rainbow table to crack the hash (HINT: keep your table small! Alpha, 1-3 chars, lowercase, etc…)
- Use rtsort to sort your tables
- Use rcrack to crack your hash
- show me your successfully cracked hash BEFORE LEAVING!