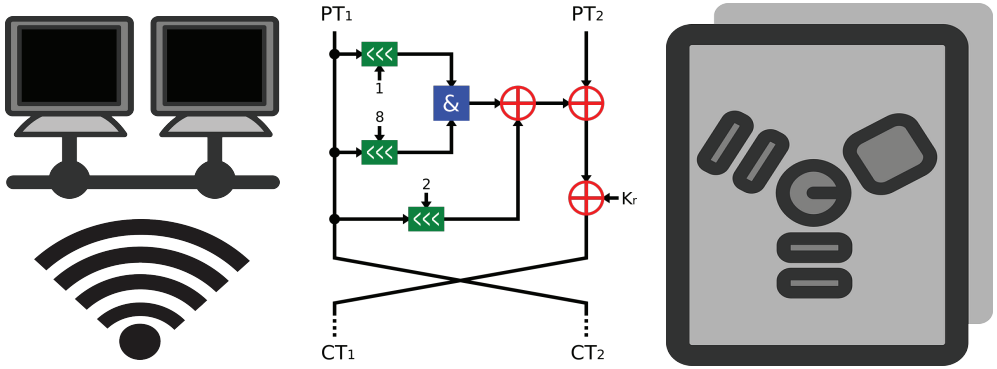
CSE 40567 / 60567: Computer Security



Software Security 1

Homework #3 has been released. It is due 2/18 at 11:59PM

See **Assignments Page** on the course website for details

Midterm Exam: 2/27 (In Class)

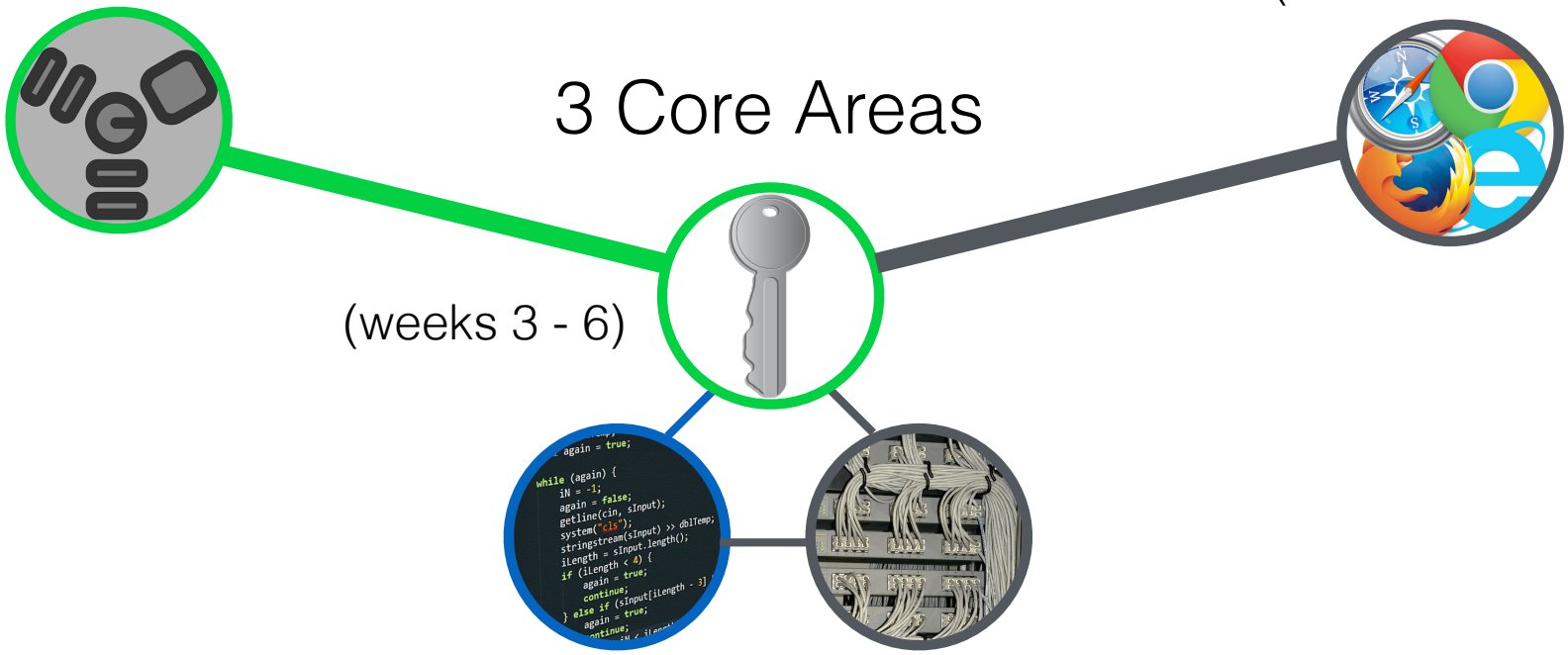
Course Roadmap

Basics

(weeks 1 & 2)

The Web

(weeks 15 & 16)



Advanced Persistent Threats

Two Facets:

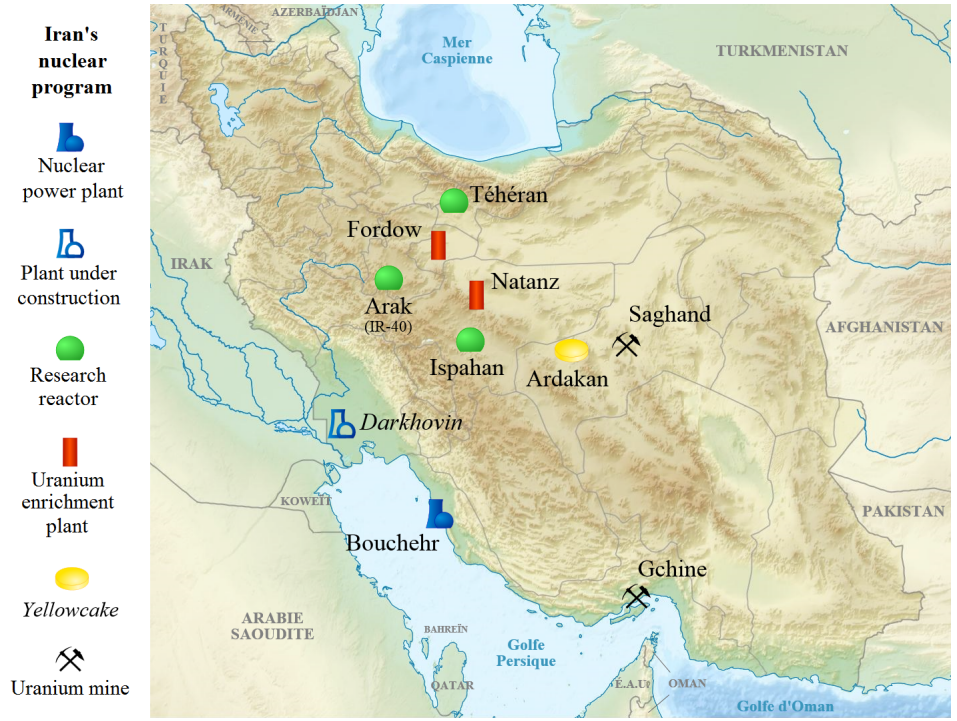
1. Good target intelligence
2. Technical attack that isn't easily deflected

Example: **Stuxnet**, malware targeted at Iranian nuclear facilities

Siemens Simatic S7-300

Stuxnet Background

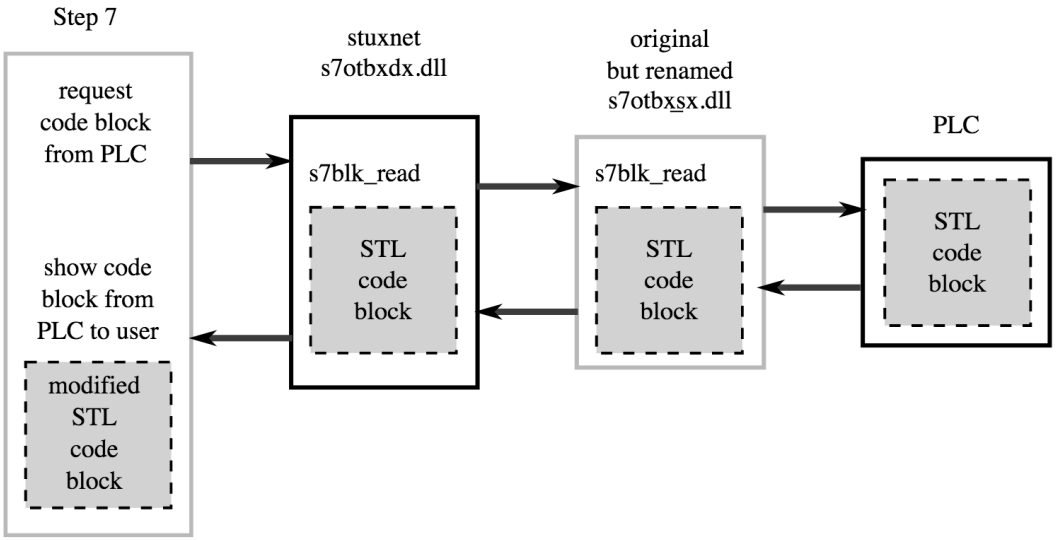
- Objective: vary speeds of centrifuge motors by infecting Siemens S7 PLCs
- Level of sophistication hints at state-sponsored work
  - Israel?
  - United States?



Map of the main sites of Iran's nuclear program CC BY-SA 4.0 Yagasi

Stuxnet's Technical Capabilities

- Exploited four "0-day" vulnerabilities in MS Windows
- Spread via a LAN or USB flash drives
- Relayed status information back to attackers
- Made use of rootkits in Windows and the PLC hardware (a first)



Step7 communicates with a PLC

What can we learn from software like Stuxnet?

Remarkable amount of intelligence about the target

- Known organizational links by which a flash drive attack might spread
- Precise information on PLCs and motor speeds that could damage operations

What can we learn from software like Stuxnet?

Significant required resources for pulling something like this off (estimated by Symantec)

- 5-10 developers
- 1/2 a year of development time
- Additional resources for testing, management, and intelligence-gathering

Framing for our discussion of software security

Brute Force Attacks and Password Cracking

Low hanging fruit

- Even reduced keyspace searches are hard
  - Randomly searching an AES keyspace of size 2^{128} will not yield anything useful
- Why conduct an exhaustive brute force attack against the keyspace when users choose bad passwords?
  - Dictionary attack is far more efficient

```
Username: jdoe
Password: ChicagoBulls
```

Password Cracking Tools

- Primarily for Unix, but runs on 15 different platforms
- Supports many different hash algorithms via system's `crypt(3)`

http://www.openwall.com/john/

http://pentestmonkey.net/cheat-sheet/john-the-ripper-hash-formats

```
# cat pass.txt
user:AZl.zWwxIh15Q
# john -w:password.lst pass.txt
Loaded 1 password hash (Traditional DES [24/32 4K])
example          (user)
guesses: 1  time: 0:00:00:00 100%  c/s: 752  trying: 12345 - pookie
```

John's attack modes

1. Dictionary attack
   - Text string is encrypted in the same format as the password being examined
   - Both cipher text representations are compared
   - John can apply a number of "mangling" rules to the dictionary words before encryption

Choose a base word → notredame

"Leetify" → n0tr3d4m3

Append / prepend things → g0n0tr3d4m3!

John’s attack modes

2. Brute force attack

- John attempts to go through all possible plaintexts, encrypting each one in the same format as the password being examined
- Both cipher text representations are compared
- Optimization: John makes use of character frequency tables to try plaintexts containing more frequently used characters first

Brute force attack against SMTP passwords

Password cracking attacks against network services in the wild are common

```
Jan  3 15:19:48 cortex saslauthd[7021]: do_auth         : auth failure:
[user=custsvc] [service=smtp] [realm=mail.vast.uccs.edu] [mech=pam]
[reason=PAM auth error]
Jan  3 15:20:06 cortex saslauthd[7020]: do_auth         : auth failure:
[user=custsvc] [service=smtp] [realm=mail.vast.uccs.edu] [mech=pam]
[reason=PAM auth error]
Jan  3 15:20:17 cortex saslauthd[7019]: do_auth         : auth failure:
[user=custsvc] [service=smtp] [realm=mail.vast.uccs.edu] [mech=pam]
[reason=PAM auth error]
Jan  3 15:20:26 cortex saslauthd[7016]: do_auth         : auth failure:
[user=custsvc] [service=smtp] [realm=mail.vast.uccs.edu] [mech=pam]
[reason=PAM auth error]
```

…

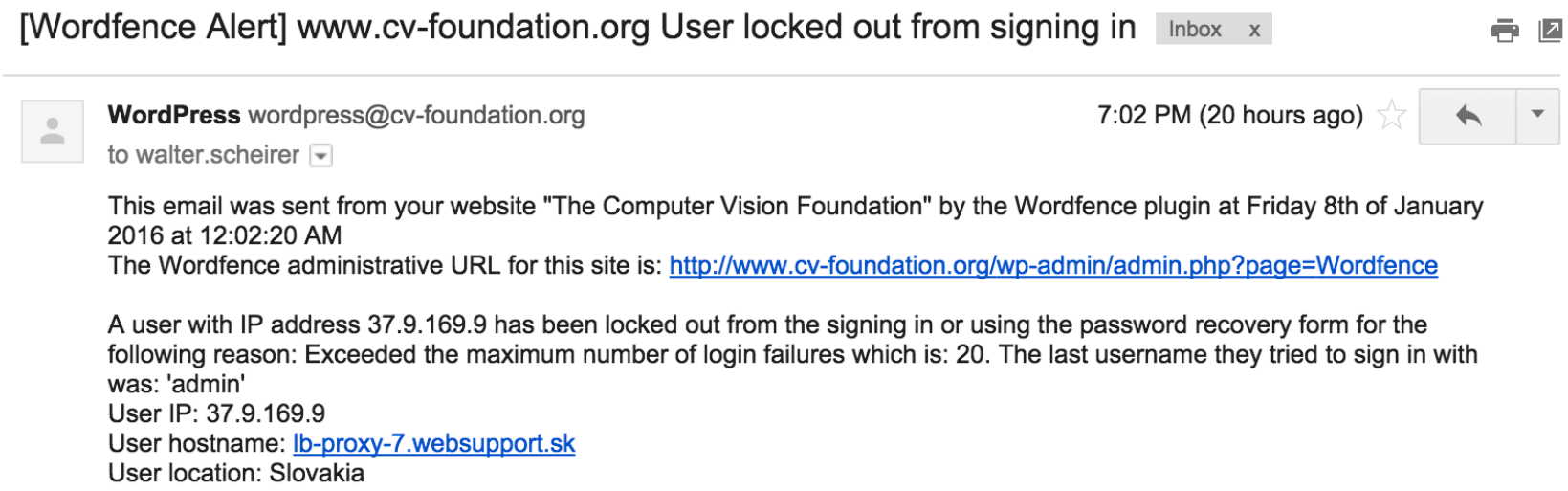
fail2ban

http://www.fail2ban.org/

- Monitors log files for brute force attacks
  - `/var/log/auth.log, /var/log/apache/access.log`
- Blocking mechanisms
  - firewall rules, updates to TCP Wrapper's hosts.deny table, email notifications, or any user-defined action that can be carried out by a Python script.

Common Software to monitor: Apache, sshd, Postfix

Brute force attack against Wordpress passwords



[Wordfence Alert] www.cv-foundation.org User locked out from signing in Inbox x

WordPress wordpress@cv-foundation.org 7:02 PM (20 hours ago)

to walter.scheirer

This email was sent from your website "The Computer Vision Foundation" by the Wordfence plugin at Friday 8th of January 2016 at 12:02:20 AM
The Wordfence administrative URL for this site is: http://www.cv-foundation.org/wp-admin/admin.php?page=Wordfence

A user with IP address 37.9.169.9 has been locked out from the signing in or using the password recovery form for the following reason: Exceeded the maximum number of login failures which is: 20. The last username they tried to sign in with was: 'admin'
User IP: 37.9.169.9
User hostname: lb-proxy-7.websupport.sk
User location: Slovakia

- Web software with authentication mechanisms is also prone to attack
- Limit logins and implement 2-factor authentication

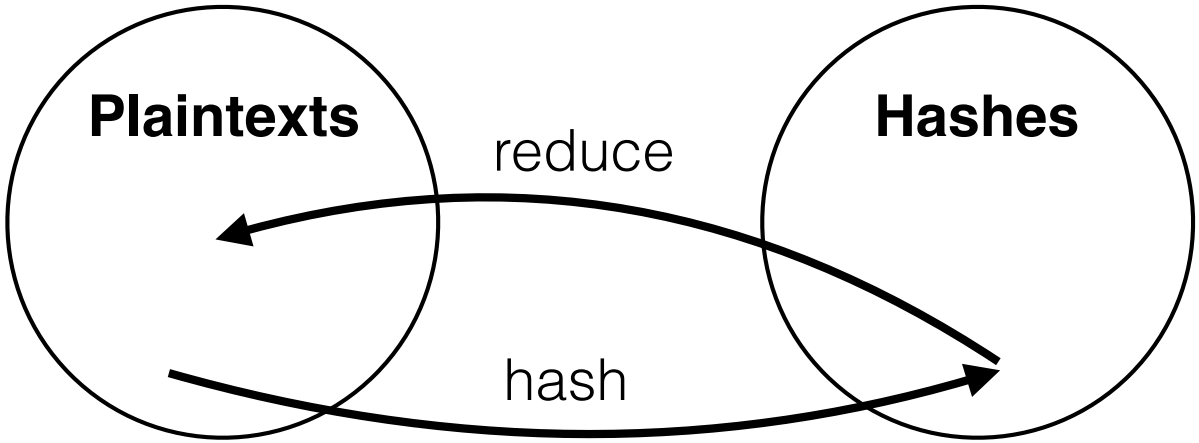
Rainbow Tables

- From an attacker's perspective, password cracking is expensive:
  - Each password needs to be hashed, which is slow
  - Each pre-computed hash needs to be stored, which isn't feasible for large sets of passwords

  Rainbow Tables are a compromise between pre-computation and low memory usage

Reduction Functions

- A hash function maps plaintexts to hashes
- A reduction function maps hashes to plaintexts



- The reduction function does the reverse of a hash function, but it isn't its inverse

Example Reduction Function

Assume:

Set of plaintexts is `[0123456789]{6}` (all numeric passwords of length 6)

Hash function is `md5()`

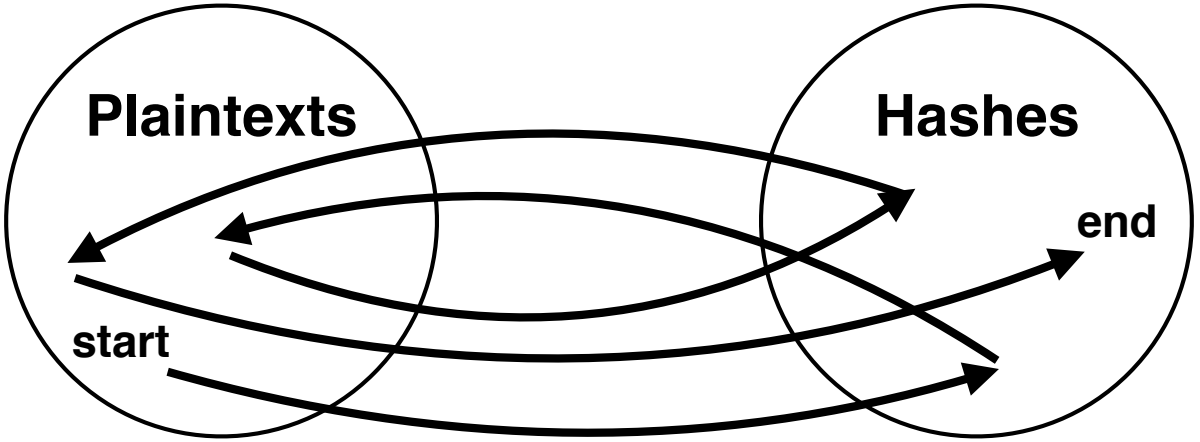
Reduction function `R()` returned the first six numbers of the hash

```
md5("493823") = "222f00dc4b7f9131c89cff641d1a8c50"
R("222f00dc4b7f9131c89cff641d1a8c50") = "222004"
```

This process generated another plaintext from the hash of the previous plaintext — the purpose of the reduction function.

Rainbow table chain

- The tables are made up of chains of hash and reduction functions
- A table only stores the starting plaintext and final hash



After generating many chains the table might look something like:

```
iaisudhiu = 4259cc34599c530b1e4a8f225d665802
oxcvioix = c744b1716cbf8d4dd0ff4ce31a177151
9da8dasf = 3cd696a8571a843cda453a229d741843
[...]
sodifo8sf = 7ad7d6fa6bb4fd28ab98b3dd33261e8f
```

Rainbow table algorithm

- Assume Mallory has a hash with an unknown plaintext
- She checks to see whether it is inside any of the generated chains

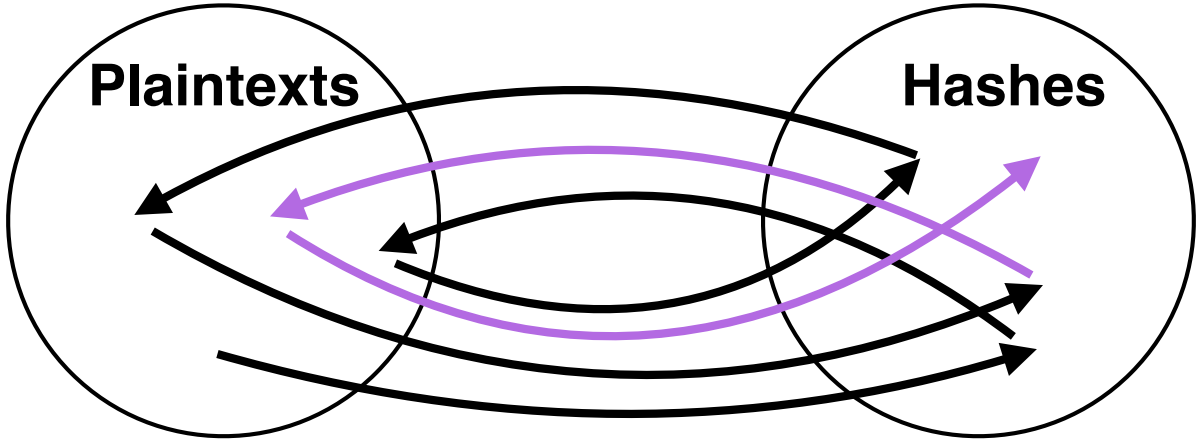
```
while(hash not found)
```

1. Look for the hash in the list of final hashes, if it is there break out of the loop.
2. If it isn't there reduce the hash into another plaintext, and hash the new plaintext.
3. Goto the start.
4. If the hash matches one of the final hashes, the chain for which the hash matches the final hash contains the original hash.

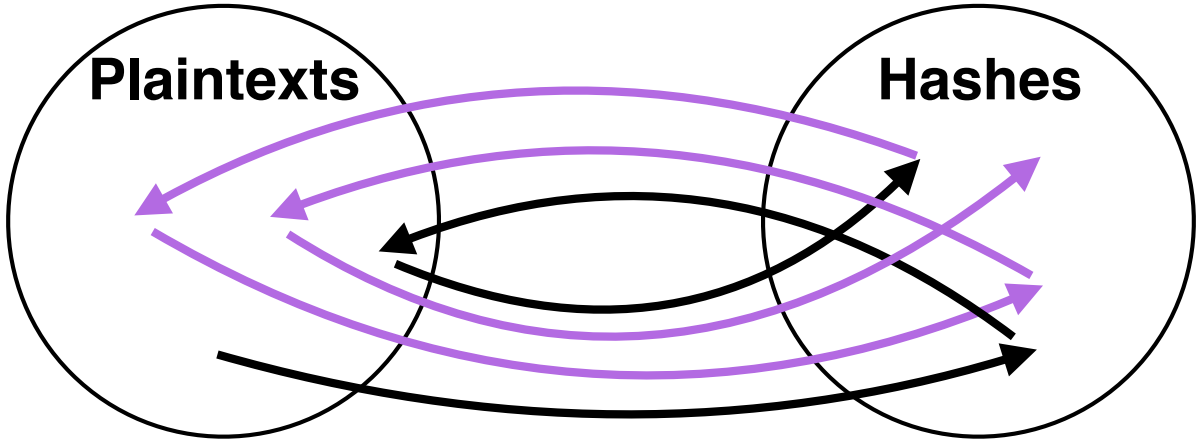
Mallory can now get that chain's starting plaintext, and start hashing and reducing it, until she comes to the known hash along with its secret plaintext.

Checking the chains

Check the last column of the table: reduce and hash once

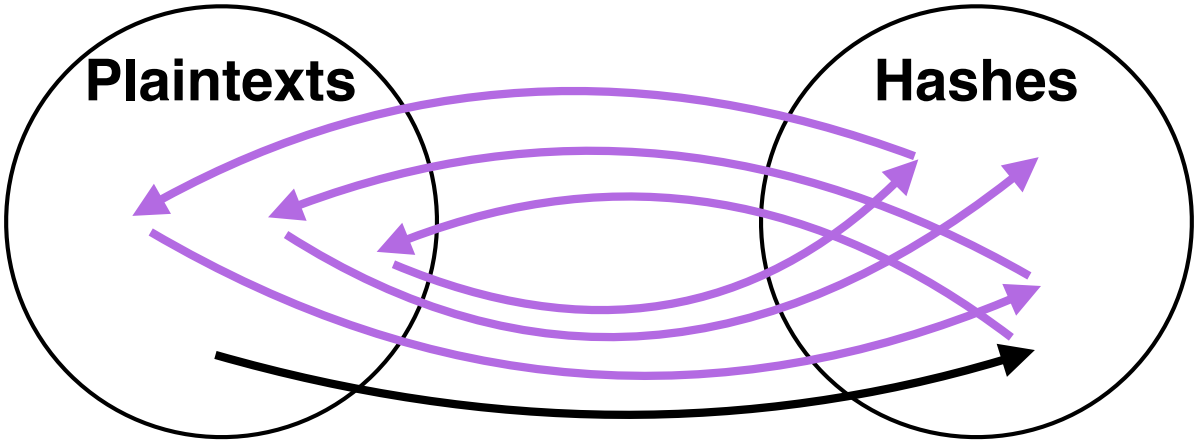


Check the second to last column of the table: reduce and hash twice

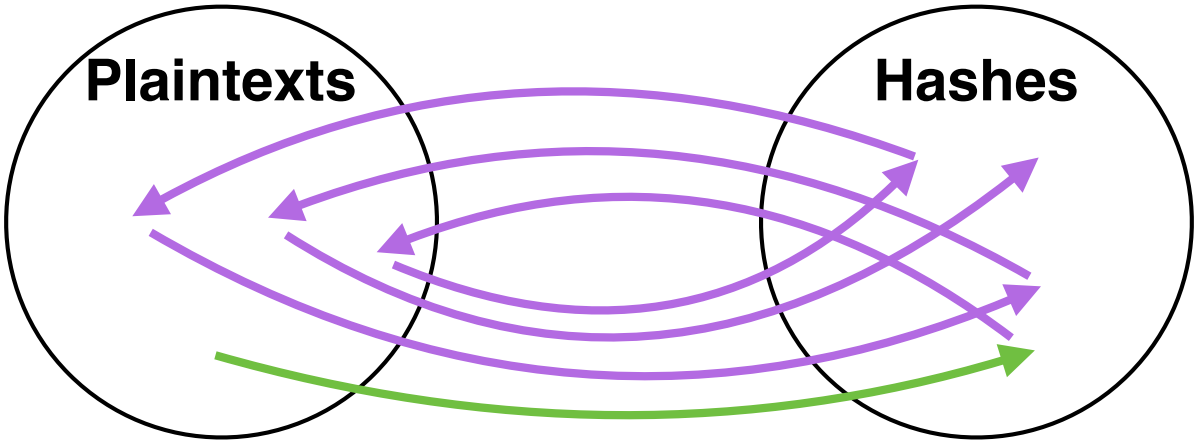


Checking the chains

Check the third to last column of the table: reduce and hash three times

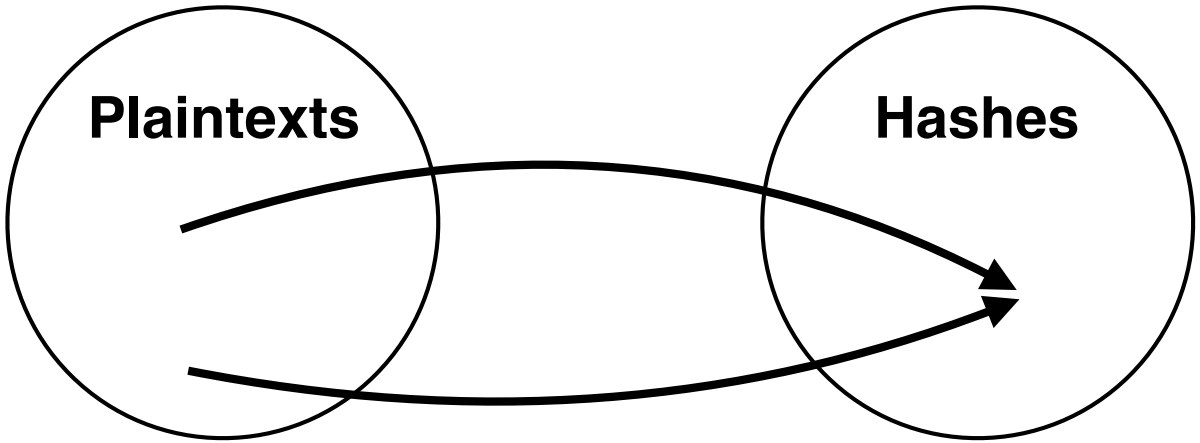


Match: The starting plaintext (stored with the ending hash) is reduced and hashed until the correct plaintext is found within the chain

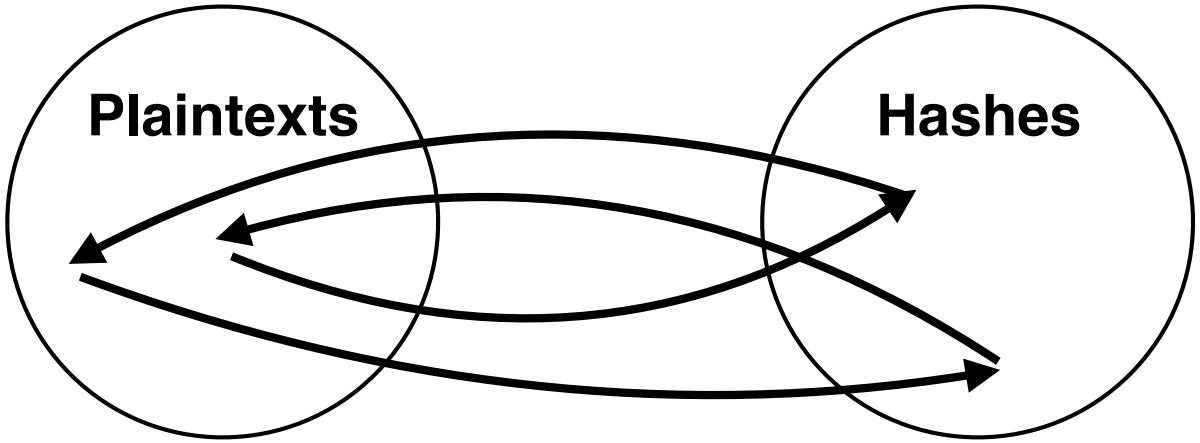


One problem: collisions

Two plaintexts hash to the same value:



This causes cycles in the table:

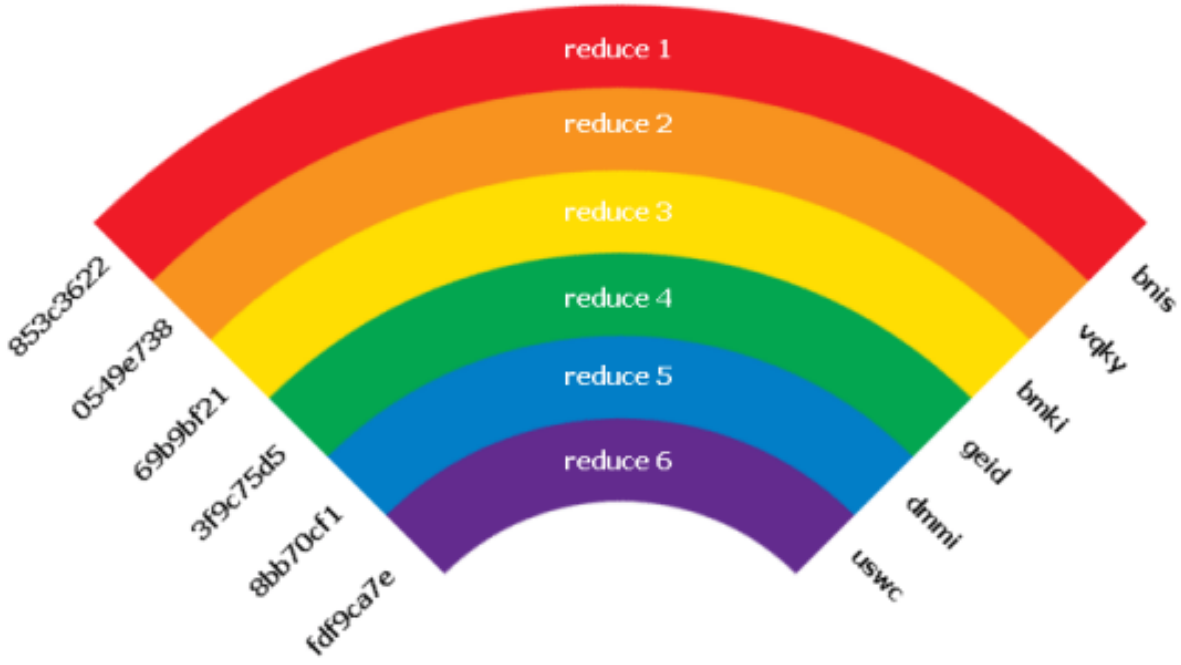


No guarantee that there will be a hash of a plaintext that will reduce to some other given plaintext.

Solution: use a different reduction function in each column

- Origin of the name “Rainbow Table”
  - If each reduction function is a different color, with starting plaintexts at the top and final hashes at the bottom, the table would look like a rainbow
- Chain merges become rare, because collisions have to occur on the same column (chance of collision is `1 / chain length`)
- Loops are also solved: if a hash in a chain is the same as a previous hash it won't reduce to the same plaintext.

Color-coded reduction functions

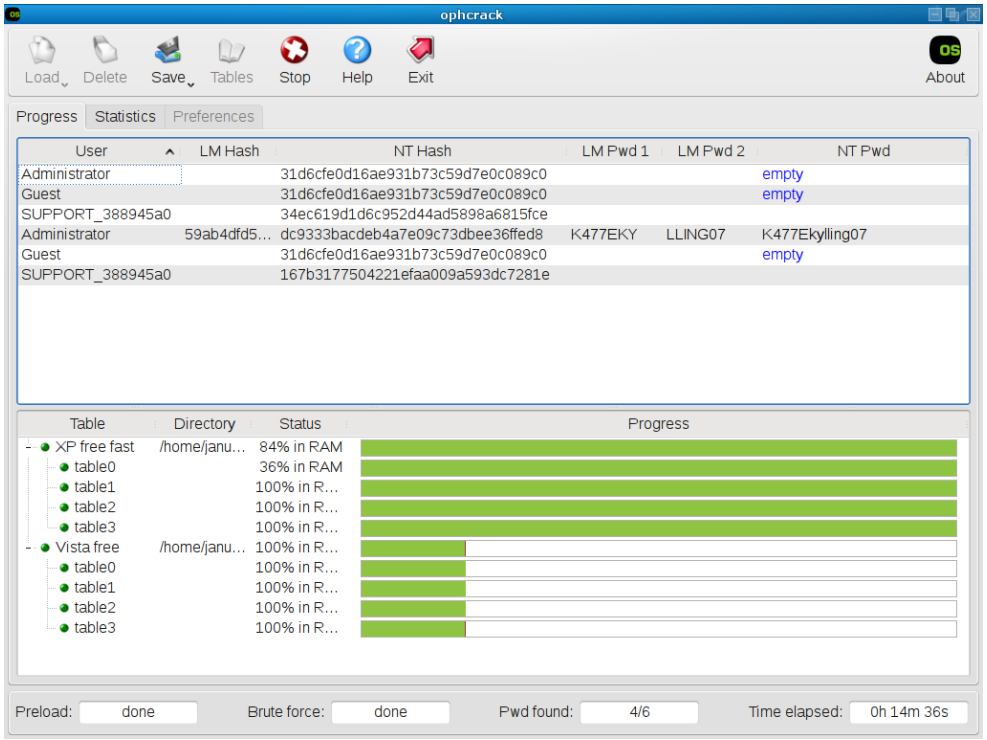


http://www.thesecurityblogger.com/understanding-rainbow-tables/

Password Cracking Tools

Ophcrack (http://ophcrack.sourceforge.net/)

Windows password cracker based on rainbow tables



Screenshot of w:Ophcrack version 3.2.0

- Runs on Windows, Linux/Unix, Mac OS X, ...
- Cracks LM and NTLM hashes
- Free tables available for Windows XP and Vista/7
- Live image available to simplify the cracking

Password Salts

- With a dictionary, it's possible to pre-compute a hash for every word, for all known algorithms
- How do we defend against this?

  Calculate a different hash:

  $$H'(\texttt{username, site, password})$$

  and use the high-order 64 bits as the salt and the low-order 18-24 bits as the *iteration count.*

  Iterations slow down attacks, e.g., if every password is hashed 100,00 times, guessing is slowed down to 1/100,000 the previous rate. (Slows down legitimate use as well.)