Introduction to Security
Cryptography

Ming Chow (mchow@cs.tufts.edu)

Twitter: @0xmchow

Tufts UNIVERSITY | SCHOOL OF ENGINEERING Computer Science

Learning Objectives

- By the end of this week, you will be able to:
  - Understand the difference between symmetric and asymmetric cryptography
  - Understand and use one-way hash functions
  - Understand how Transport Layer Security (TLS) works
  - Understand how and how not to store users' passwords

Why Cryptography?

- To put it quite simply, it is the backbone of Cyber Security, critical for protecting information (confidentiality and integrity)
- Week 1 on networking and attacking networks alluded to the importance of Cryptography
  - Imagine a world without Cryptography. Imagine your usernames, passwords, credit card numbers, messages, secrets, account details, personal information, emails, etc. were all transmitted over a computer network in plaintext, unencrypted

Warning

- This week is not a comprehensive study on Cryptography
- Cryptography stands as a course and field of its own
- While Cryptography is critical in Cyber Security, Cryptography and Cyber Security are not the same. Some academic institutions still teach Cyber Security as Cryptography. There is a lot more to Cyber Security than Cryptography.

Definitions

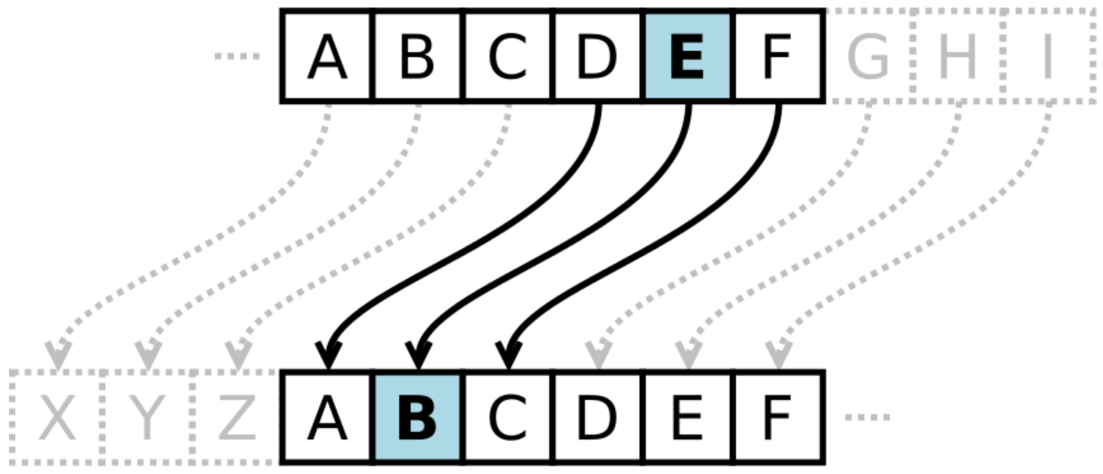
- **Cryptography** - The process of communicating secretly through the use of *cipher*
- **Cryptanalysis** - The process of cracking or deciphering; code breaking
- **Cryptology** - The study of cryptography or cryptanalysis
- **Cleartext / plaintext** - What you are reading now
- **Encrypt** - convert information or data into code to prevent unauthorized access
- **Decrypt** – convert an encoded or unclear message into something intelligible, to plaintext
- **Cipher** - An algorithm to perform encryption and/or decryption
- **Cryptosystem** - Suite of algorithms to perform encryption and/or decryption

The Golden Rule

- **“Don’t roll your own crypto”**
- The reason: https://security.stackexchange.com/questions/18197/why-shouldnt-we-roll-our-own
- “Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break. It's not even hard. What is hard is creating an algorithm that no one else can break, even after years of analysis. And the only way to prove that is to subject the algorithm to years of analysis by the best cryptographers around.” –Bruce Schneier
- Snake oil: https://www.schneier.com/crypto-gram/archives/1999/0215.html#snakeoil

Ancient History: Caesar Cipher

- A substitution cipher
- "Each letter in the original message (plaintext) is replaced with a letter corresponding to a certain number of letters up or down in the alphabet."
- In this way, a message that initially was quite readable, ends up in a form that can not be understood at a simple glance.
- Source: https://learncryptography.com/classical-encryption/caesar-cipher
- The purpose of this: what if a messenger for Julius Caesar got mugged or murdered and the message that was supposed to be delivered to another party got intercepted or stolen by enemy?



Tufts UNIVERSITY | SCHOOL OF ENGINEERING Computer Science

Security of a Cryptosystem

- The only perfectly secure algorithm is the *One-Time Pad*
- Is any crypto algorithm perfectly secure?
- Tradeoff 1: the cost of breaking a cipher exceeds the value of the encrypted information
- Tradeoff 2: the time required to break the cipher exceeds the useful lifetime of the information
- Very difficult to estimate cost and time required to break a cipher
  - There is always brute force
  - ...and then there is plain-old stealing or just asking for it

One-Time Pad (OTP)

- Invented in 1917
- Impossible to crack
- The secret key (the cipher), with random data, must be the same length as the plaintext
- Assume "A" = 0, "B" = 1, "C" = 2, etc.
- Simple to use: just XOR, modular addition
    - https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/e/modular-addition
- Encryption: addition, mod 26
- Decryption: subtraction, if result is negative, add 26 and mod 26
- Rather impractical

SCHOOL OF ENGINEERING
Computer Science

Algorithms

- Hash functions - one way encryption, no decryption thus no secret key
  - MD5 (insecure)
  - SHA1 (insecure), SHA256, SHA512
- Symmetric - single key for encryption and decryption
  - DES
  - AES
  - RC4
- Asymmetric a.k.a., public key - uses two different keys: one public (for encryption) and one private (for decryption)
  - Diffie-Hellman
  - RSA

Base64

- NOT encryption!
- An encoding scheme of representing binary data using only printable (text) characters
  - Example: an image (JPG, PNG) contains binary data. Convert the binary data to text characters. https://www.base64-image.de/
- RFC 4648: https://www.ietf.org/rfc/rfc4648.txt
- Operations: encode and decode
  - https://www.base64decode.org/
  - Example: `hello` encoded in Base64 is `aGVsbG8=`. The "=" at the end of encoding serves as padding.
- Many programming languages include a Base64 library
- Has an important place in Cyber Security
  - Used quite a bit, including for HTTP Basic Authentication
- References:
  - https://en.wikipedia.org/wiki/Basic_access_authentication
  - https://stackoverflow.com/questions/201479/what-is-base-64-encoding-used-for
  - https://stackoverflow.com/questions/4070693/what-is-the-purpose-of-base-64-encoding-and-why-it-used-in-http-basic-authentica

Hash Functions

- Maps a variable length string of data to produce a fixed-length output in deterministic, public, and random manner
- No secret key
- Properties of a perfect hash function (recall properties of a hash function for hash tables in a Data Structures course):
  - One-way: cannot decrypt
  - No collisions: two unique strings cannot produce the same result
  - Randomness
  - Unfeasible to produce the whole hash space (pre-image resistance)
  - Given a hash result, unfeasible to produce the string

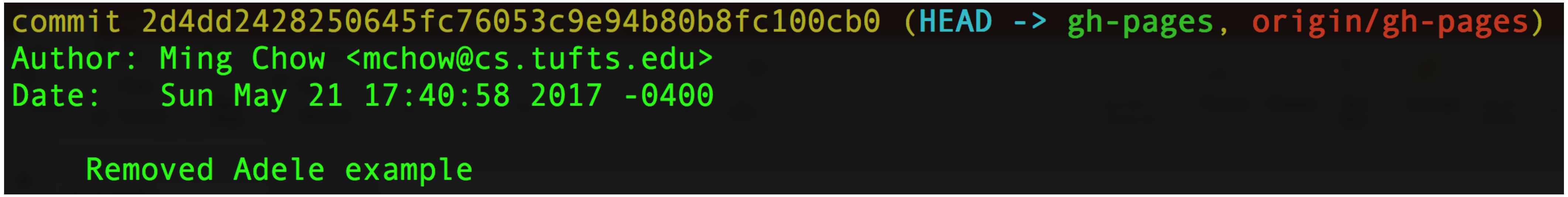
Hash Functions: Tradeoffs

- Strengths:
  - Verifying integrity
- Weaknesses:
  - MD5 (128-bit hash value) is broken --not collision-resistant (two researchers created two files that shared the same hash value). Read:
    - http://www.cs.colorado.edu/~jrblack/papers/md5e-full.pdf
    - http://eprint.iacr.org/2004/199.pdf
  - SHA1 (160-bit hash value) is broken –badly; big news in February 2017
    - http://www.schneier.com/blog/archives/2005/02/sha1_broken.html,
    - http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html
    - https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html
    - https://shattered.io/

Tufts UNIVERSITY | SCHOOL OF ENGINEERING Computer Science

Hash Function Applications

- Password storage
- Checksum of software packages
- Digital signatures
- Commits in Git which use SHA-1
  - https://gist.github.com/masak/2415865



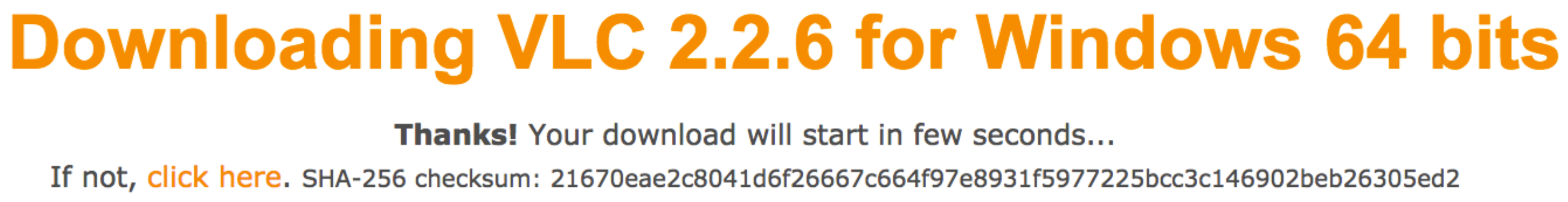
```
commit 2d4dd2428250645fc76053c9e94b80b8fc100cb0 (HEAD -> gh-pages, origin/gh-pages)
Author: Ming Chow <mchow@cs.tufts.edu>
Date:   Sun May 21 17:40:58 2017 -0400

    Removed Adele example
```

Example: Verifying the integrity of a download

1. Let’s download the popular VLC Media Player for Windows (64-bit) at https://get.videolan.org/vlc/2.2.6/win64/vlc-2.2.6-win64.exe

2. Most software downloads will provide a checksum. For this version of VLC, we see SHA-256 checksum: 21670eae2c8041d6f26667c664f97e8931f5977225bcc3c146902beb26305ed2



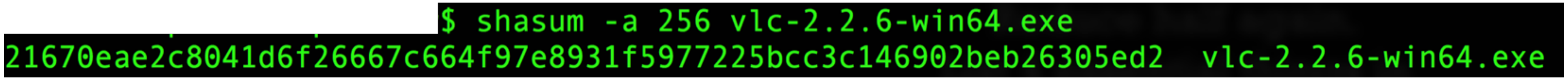
Downloading VLC 2.2.6 for Windows 64 bits

Thanks! Your download will start in few seconds...

If not, click here. SHA-256 checksum: 21670eae2c8041d6f26667c664f97e8931f5977225bcc3c146902beb26305ed2

Example: Verifying the integrity of a download (continued)

3. On Linux or Mac OS X terminal, run the following in the folder where the vlc-2.2.6-win64.exe file is: `shasum -a 256 vlc-2.2.6-win64.exe`. Result should match the checksum provided on the website. If result does NOT match, either a tampered download or corrupted download –both not good!



```
$ shasum -a 256 vlc-2.2.6-win64.exe
21670eae2c8041d6f26667c664f97e8931f5977225bcc3c146902beb26305ed2  vlc-2.2.6-win64.exe
```

Symmetric Algorithms

- One key for encryption and decryption
- Let K = secret key (think password), C = Ciphertext, P = Plaintext, E = Encrypt function, D = Decrypt function
- C = Ek(P)
- P = Dk(C)
- Example: One-Time Pad
- Applications
  - Password protect a ZIP file –which uses AES: https://security.stackexchange.com/questions/35818/are-password-protected-zip-files-secure
- Strengths:
  - Modified key K will result in garbage plaintext in decryption
  - Fast!
- Weaknesses:
  - Those who know K can participate in communications (eavesdropping)
  - Impersonation attack if attacker knows K
  - Not good for authenticity

Asymmetric Algorithms a.k.a. Public Key Algorithms

- Two keys
  - Public key: which anyone can have, can be distributed publicly
  - Private key: only you should have
- How it works:
  - Alice and Bob agree on a public-key cryptosystem
  - Alice and Bob have their own public and private keys
  - Alice gives Bob her public key
  - Bob encrypts message with Alice's public key Alice decrypts the message with her private key
  - Even better: key signing (encrypt message with own private key)
- Arguably the most popular algorithm: RSA. Walkthough: http://www.di-mgt.com.au/rsa_alg.html
- Strengths:
  - Public key can be distributed any way possible
  - Confidentiality: only holder of private key can decrypt message
  - Integrity: any modification of the message would be revealed when decrypting
  - Non repudiation: Bob can prove to a third party that Alice is the originator of the message
- Weakness:
  - No authentication: anyone can encrypt a message given a public key
  - Man-in-the-Middle (MitM) attack

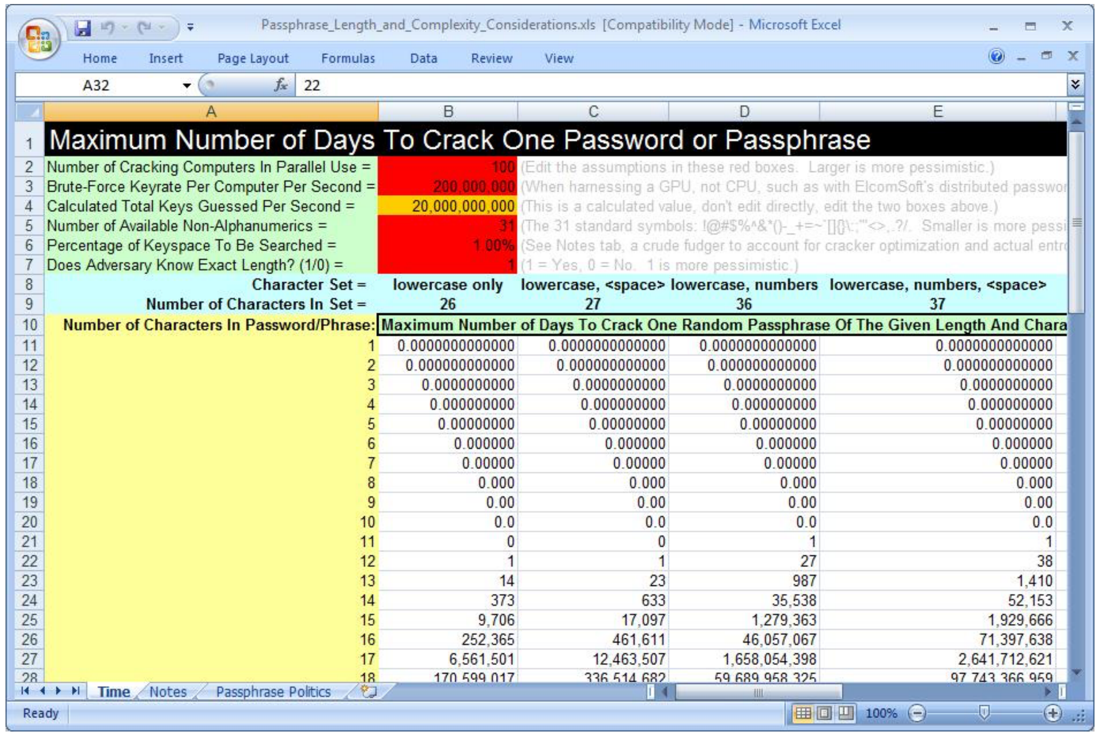
Tufts UNIVERSITY | SCHOOL OF ENGINEERING Computer Science

Asymmetric Algorithms Applications

- Secure Socket Layer (SSL) now Transport Layer Security (TLS)
- SSH and SSH Keys
  - Why: “SSH keys provide a more secure way of logging into a virtual private server with SSH than using a password alone. While a password can eventually be cracked with a brute force attack, SSH keys are nearly impossible to decipher by brute force alone. Generating a key pair provides you with two long string of characters: a public and a private key. You can place the public key on any server, and then unlock it by connecting to it with a client that already has the private key. When the two match up, the system unlocks without the need for a password. You can increase security even more by protecting the private key with a passphrase.” https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys--2
  - Generating a new SSH key for GitHub: https://help.github.com/articles/connecting-to-github-with-ssh/
- PGP (Pretty Good Privacy)
- GPG formerly known as OpenPGP
- Bitcoin

Passwords and Password Cracking

- Question: How long is your password good for? 30 days? 60 days? 90 days?
- Answer: As long as it is not broken
- The bottom line: it will only be a matter of when, not if, your password will be broken
- Source for photo: https://www.sans.org/blog/how-long-to-crack-a-password-spreadsheet/



Passphrase_Length_and_Complexity_Considerations.xls [Compatibility Mode] - Microsoft Excel

A32 | 22

	A	B	C	D	E
1	Maximum Number of Days To Crack One Password or Passphrase				
2	Number of Cracking Computers In Parallel Use =	100	(Edit the assumptions in these red boxes. Larger is more pessimistic.)		
3	Brute-Force Keyrate Per Computer Per Second =	200,000,000	(When harnessing a GPU, not CPU, such as with ElcomSoft's distributed passwor		
4	Calculated Total Keys Guessed Per Second =	20,000,000,000	(This is a calculated value, don't edit directly, edit the two boxes above.)		
5	Number of Available Non-Alphanumerics =	31	(The 31 standard symbols: !@#$%^&*()-_+=~`[]{}\:;"'<>,.?/. Smaller is more pessi		
6	Percentage of Keyspace To Be Searched =	1.00%	(See Notes tab, a crude fudger to account for cracker optimization and actual entro		
7	Does Adversary Know Exact Length? (1/0) =	1	(1 = Yes, 0 = No. 1 is more pessimistic.)		
8	Character Set =	lowercase only	lowercase, <space>	lowercase, numbers	lowercase, numbers, <space>
9	Number of Characters In Set =	26	27	36	37
10	Number of Characters In Password/Phrase:	Maximum Number of Days To Crack One Random Passphrase Of The Given Length And Chara			
11	1	0.0000000000000	0.0000000000000	0.0000000000000	0.0000000000000
12	2	0.000000000000	0.000000000000	0.000000000000	0.000000000000
13	3	0.0000000000	0.0000000000	0.0000000000	0.0000000000
14	4	0.000000000	0.000000000	0.000000000	0.000000000
15	5	0.00000000	0.00000000	0.00000000	0.00000000
16	6	0.000000	0.000000	0.000000	0.000000
17	7	0.00000	0.00000	0.00000	0.00000
18	8	0.000	0.000	0.000	0.000
19	9	0.00	0.00	0.00	0.00
20	10	0.0	0.0	0.0	0.0
21	11	0	0	1	1
22	12	1	1	27	38
23	13	14	23	987	1,410
24	14	373	633	35,538	52,153
25	15	9,706	17,097	1,279,363	1,929,666
26	16	252,365	461,611	46,057,067	71,397,638
27	17	6,561,501	12,463,507	1,658,054,398	2,641,712,621
28	18	170,599,017	336,514,682	59,689,958,325	97,743,366,959

Cracking Passwords on a Linux System

- Two files of interest on a typical Linux box:
  - `/etc/passwd` - Contains users' information but no encrypted password; required for login
  - `/etc/shadow` - Contains users' passwords (encrypted) with additional details relating to the password (see http://tldp.org/LDP/lame/LAME/linux-admin-made-easy/shadow-file-formats.html for more details)
- Methods:
  - Brute-force
  - Wordlists
  - Rainbow tables. See http://project-rainbowcrack.com/ for example.

Tufts UNIVERSITY | SCHOOL OF ENGINEERING Computer Science

Tool: John the Ripper (JtR)

- http://www.openwall.com/john/
- Step 1: merge the `/etc/passwd` and `/etc/shadow` files via `unshadow` tool provided with JtR, save result to a text file (e.g., `crackme.txt`)
  - `unshadow passwd shadow > crackme.txt`
- Format of a password hash in an unshadowed file:
  - `$algorithm$salt$hash` where `$algorithm$` means one of the following:
    - $1$ = MD5
    - $2$ = Blowfish
    - $5$ = SHA-256
    - $6$ = SHA-512
  - Example: `msfadmin:`**`$1$XN10Zj2c$Rt/zzCW3mLtUWA.ihZjA5`**`/:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash`
  - **Password Salt** - random data as an additional input to a one-way function that hashes a password or passphrase. The primary function of a salt is to defend against dictionary attacks or a pre-computed rainbow table attack. In other words, to make a common password uncommon.
- Step 2: run `john crackme.txt` to start cracking passwords
  - "If valid password files are specified but no options are given, John will go through the default selection of cracking modes with their default settings." http://www.openwall.com/john/doc/OPTIONS.shtml

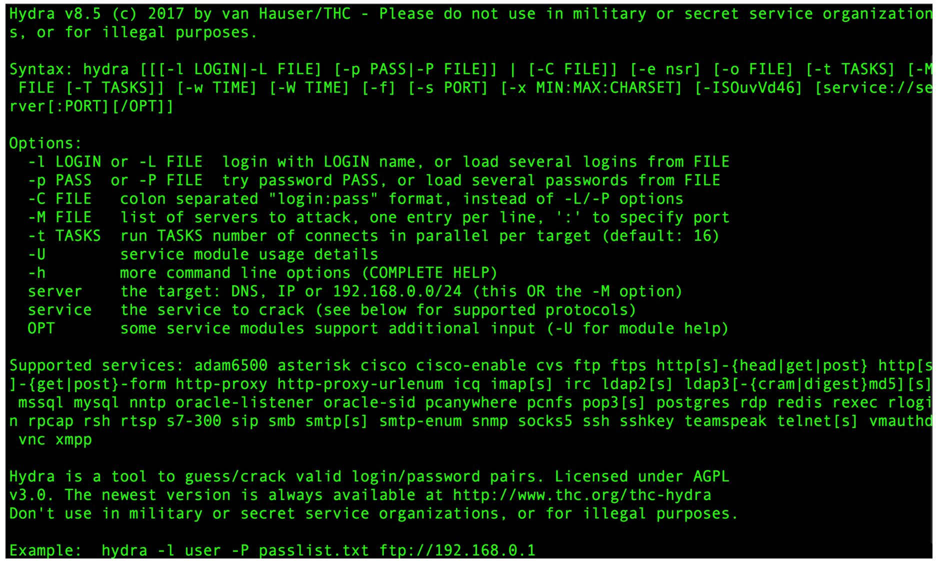
Tufts UNIVERSITY | SCHOOL OF ENGINEERING Computer Science

Cracking Passwords Using Wordlists

- Daniel Miessler maintains a fantastic set of wordlists and other lists at https://github.com/danielmiessler/SecLists
- Wordlist mode in JtR: `john --wordlist=<FILE> crackme.txt`

Additional Passwords Crackers

- L0phtCrack (commercial; http://www.l0phtcrack.com/)
- Cain & Abel (http://www.oxid.it/cain.html)
- Hashcat (https://hashcat.net/hashcat/)
- THC Hydra (free and open source)



```
Hydra v8.5 (c) 2017 by van Hauser/THC - Please do not use in military or secret service organization
s, or for illegal purposes.

Syntax: hydra [[[-l LOGIN|-L FILE] [-p PASS|-P FILE]] | [-C FILE]] [-e nsr] [-o FILE] [-t TASKS] [-M
 FILE [-T TASKS]] [-w TIME] [-W TIME] [-f] [-s PORT] [-x MIN:MAX:CHARSET] [-ISOuvVd46] [service://se
rver[:PORT][/OPT]]

Options:
  -l LOGIN or -L FILE  login with LOGIN name, or load several logins from FILE
  -p PASS  or -P FILE  try password PASS, or load several passwords from FILE
  -C FILE   colon separated "login:pass" format, instead of -L/-P options
  -M FILE   list of servers to attack, one entry per line, ':' to specify port
  -t TASKS  run TASKS number of connects in parallel per target (default: 16)
  -U        service module usage details
  -h        more command line options (COMPLETE HELP)
  server    the target: DNS, IP or 192.168.0.0/24 (this OR the -M option)
  service   the service to crack (see below for supported protocols)
  OPT       some service modules support additional input (-U for module help)

Supported services: adam6500 asterisk cisco cisco-enable cvs ftp ftps http[s]-{head|get|post} http[s
]-{get|post}-form http-proxy http-proxy-urlenum icq imap[s] irc ldap2[s] ldap3[-{cram|digest}md5][s]
 mssql mysql nntp oracle-listener oracle-sid pcanywhere pcnfs pop3[s] postgres rdp redis rexec rlogi
n rpcap rsh rtsp s7-300 sip smb smtp[s] smtp-enum snmp socks5 ssh sshkey teamspeak telnet[s] vmauthd
 vnc xmpp

Hydra is a tool to guess/crack valid login/password pairs. Licensed under AGPL
v3.0. The newest version is always available at http://www.thc.org/thc-hydra
Don't use in military or secret service organizations, or for illegal purposes.

Example:  hydra -l user -P passlist.txt ftp://192.168.0.1
```

Tufts UNIVERSITY | SCHOOL OF ENGINEERING Computer Science

Transport Layer Security (TLS)

- Why? HTTPS is HTTP inside of a TLS session
- Uses BOTH symmetric and asymmetric crypto
- Secure communications between two parties over a network
- On top of TCP
- Different port numbers used for TLS connection. Port 443 for HTTPS
- Part 1: Data between two parties encrypted via symmetric crypto. Why? Speed
- Part 2: Identity of communicating parties identified via asymmetric crypto
- Connection integrity via message integrity check using a message authentication code
- **Digital certificates** - assert the online identities of individuals, computers, and other entities on a network
    - They are issued by certification authorities (CAs) that must validate the identity of the certificate-holder both before the certificate is issued and when the certificate is used.
    - Specification: https://technet.microsoft.com/en-us/library/cc776447(v=ws.10).aspx

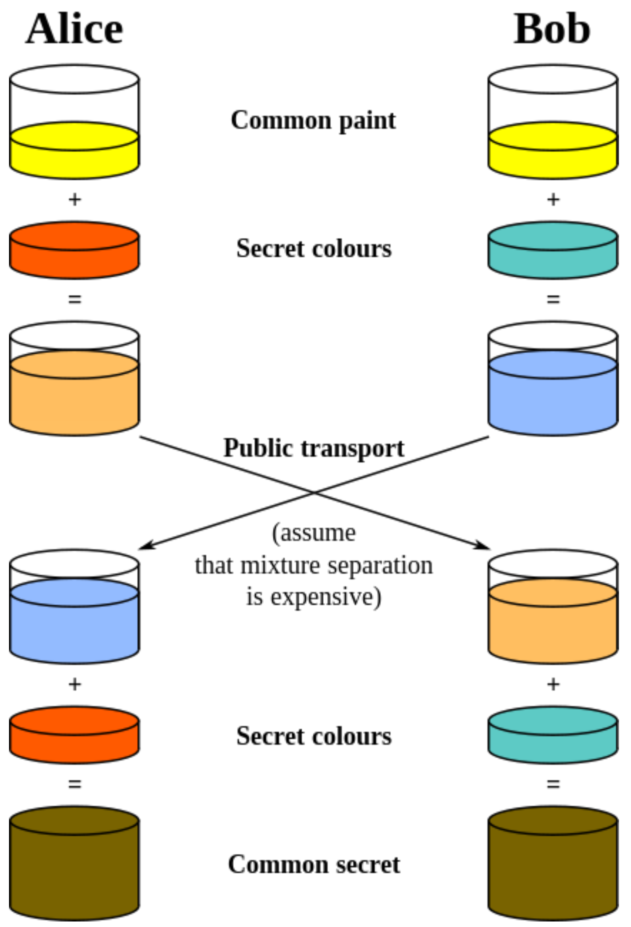
TLS Process

1. Client connects to TLS-enabled server. Client requesting a secure connection and presents a list of supported cipher suites (ciphers and hash functions).
2. The server checks what the highest SSL/TLS version is that is supported by them both, picks a ciphersuite from one of the client's options (if it supports one), and optionally picks a compression method.
3. The server sends back its identification via digital certificate (THIS MAY NOT HAPPEN)
4. Client confirms validity of certificate --or NOT!
5. Both the server and the client can now compute the session key (or shared secret) for the symmetric encryption and decryption of the data. This computation of the session key is known as **Diffie-Hellman key exchange**.
6. "The client tells the server that from now on, all communication will be encrypted, and sends an encrypted and authenticated message to the server."

Tufts UNIVERSITY | SCHOOL OF ENGINEERING Computer Science

Diffie-Hellman Key Exchange

- The idea: two parties (say Alice and Bob --tradition) exchanging cryptographic keys (or shared secret) via public and insecure channel to use to do further encryption
- Explanation in plain English: https://security.stackexchange.com/questions/45963/diffie-hellman-key-exchange-in-plain-english
- Explanation via "paint analogy" (image source: Wikipedia)



Tufts UNIVERSITY | SCHOOL OF ENGINEERING Computer Science

References on TLS

- https://security.stackexchange.com/questions/20803/how-does-ssl-tls-work
- https://stackoverflow.com/questions/788808/how-do-digital-certificates-work-when-used-for-securing-websites-using-ssl
- http://security.stackexchange.com/questions/45963/diffie-hellman-key-exchange-in-plain-english