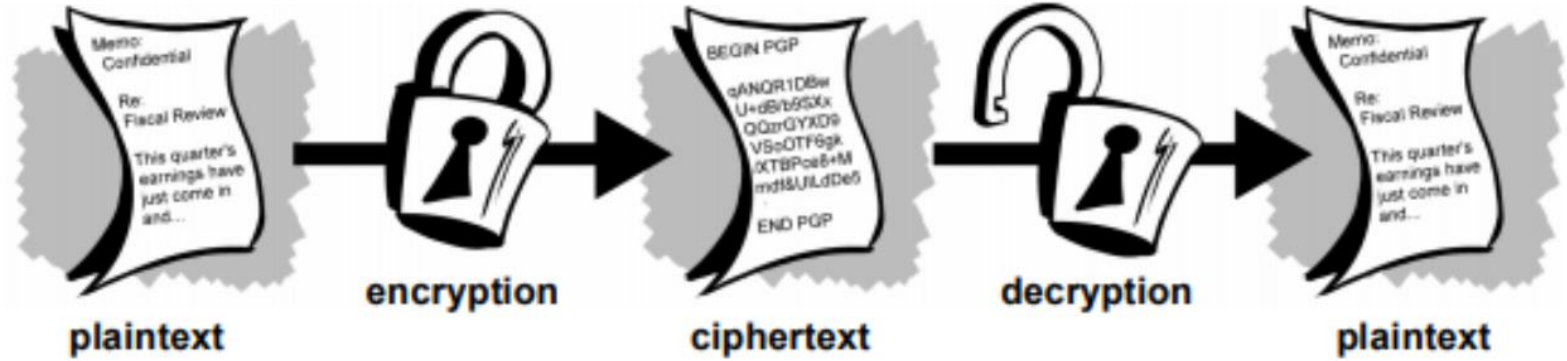
Network Security Workshop

Cryptography

APNIC

Cryptography

- Terminology



  - Cryptography
    - From Greek, “crypto” meaning hidden or secret, “graphy” meaning writing
  - Cryptanalysis
    - From Greek, “crypto” meaning hidden or secret, “analysis” meaning to loosen or untie

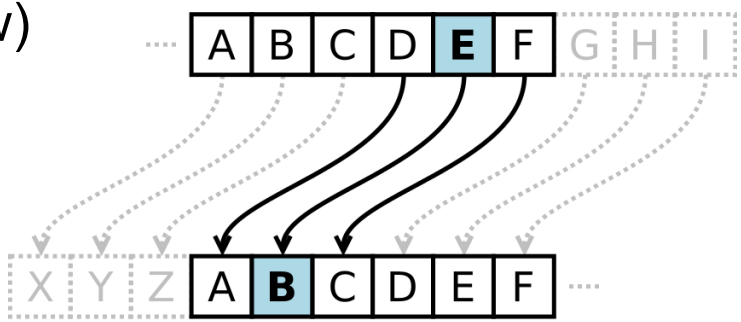
Cryptography

- History
  - Non-standard hieroglyphs in Egypt (1900 BCE)
  - Modified words on clay tablet in Mesopotamia (1500 BCE)
  - Monoalphabetic substitution ciphers
    - Hebrew scholars using Atbash Cipher (500-600 BCE)
    - Indian authors of Karma Sutra document ciphers for messages between lovers (400 BCE to 200 CE)
    - Atbash Cipher:

Plain	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cipher	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

Cryptography

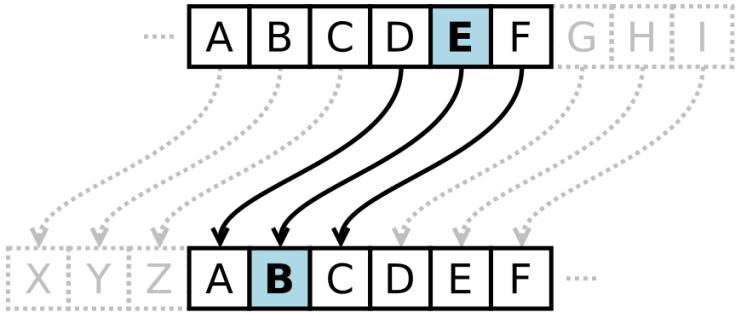
- History
  - Romans used a shift cipher called a Caesar Cipher after it's famous user Julius Caesar (100-44 BCE)
    - He used a left shift of 3 places, known as the key
    - The ROT13 systems uses a shift of 13 places
  - Caesar cipher is also used in secret decoder rings (1930's to now)



Cryptography

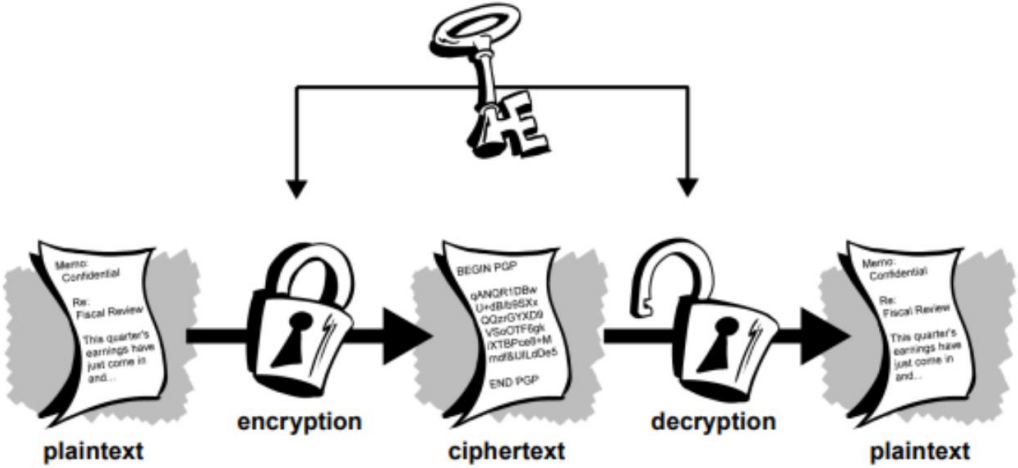
- Exercise: Use Atbash and Caesar ciphers
  - https://gchq.github.io/CyberChef/
  - Look under Encryption/Encoding for Atbash and ROT13
  - Using ROT13 for Caesar cipher, left shift encoding is negative numbers (e.g. -3) and right shift decoding is positive numbers (e.g. 3)

  - Decode Atbash: svool dliow
  - Decode Caesar: zovmql fp crk
    (for this one, use ROT13 with different numbers)



Cryptography

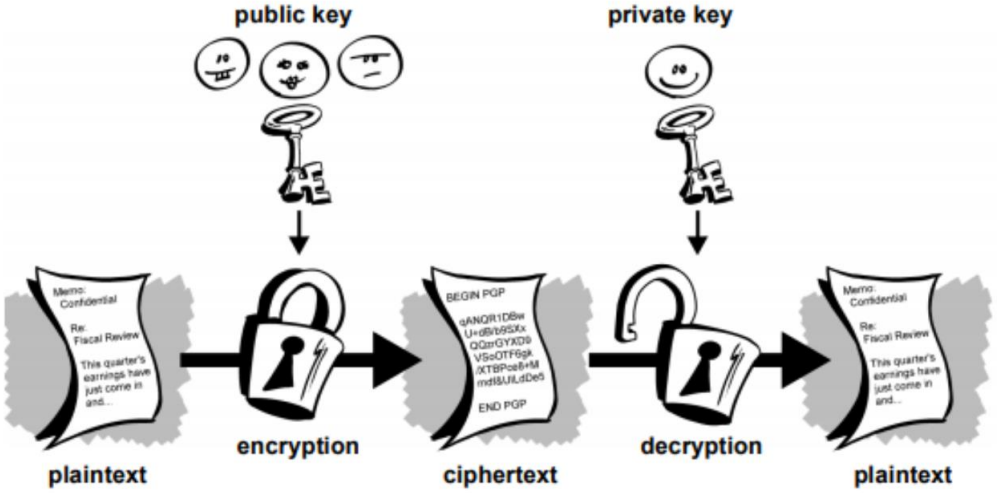
- Symmetric Algorithms
  - aka Private Key Crypto
  - The basic concept
    - Uses the same key



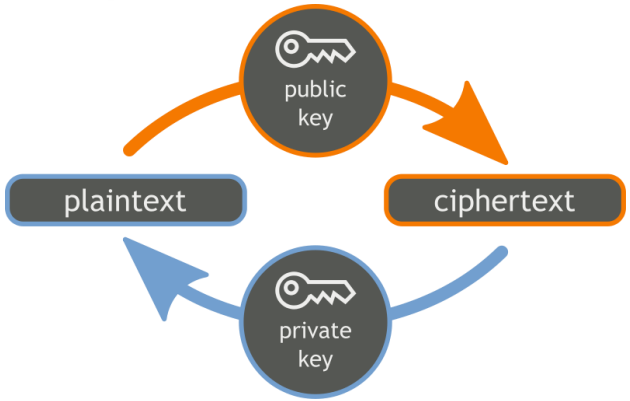
  - Common symmetric algorithms
    - AES
    - DES, 3DES
    - Blowfish
  - Exercise: Using CyberChef again, encrypt/decrypt using Blowfish
    - Demo on-screen

Cryptography

- Asymmetric Algorithms
  - aka Public Key Crypto
  - The basic concept
    - Uses a public key and a private key



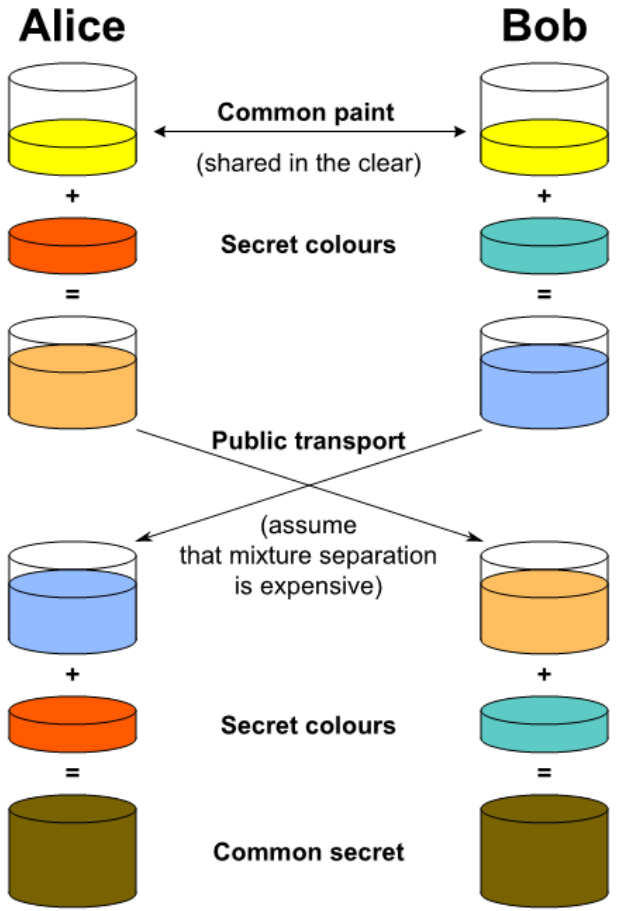
  - Common asymmetric algorithms
    - RSA (shown in the first image)
    - Diffie-Hellman (shown on next slide)
    - ElGamal



Cryptography

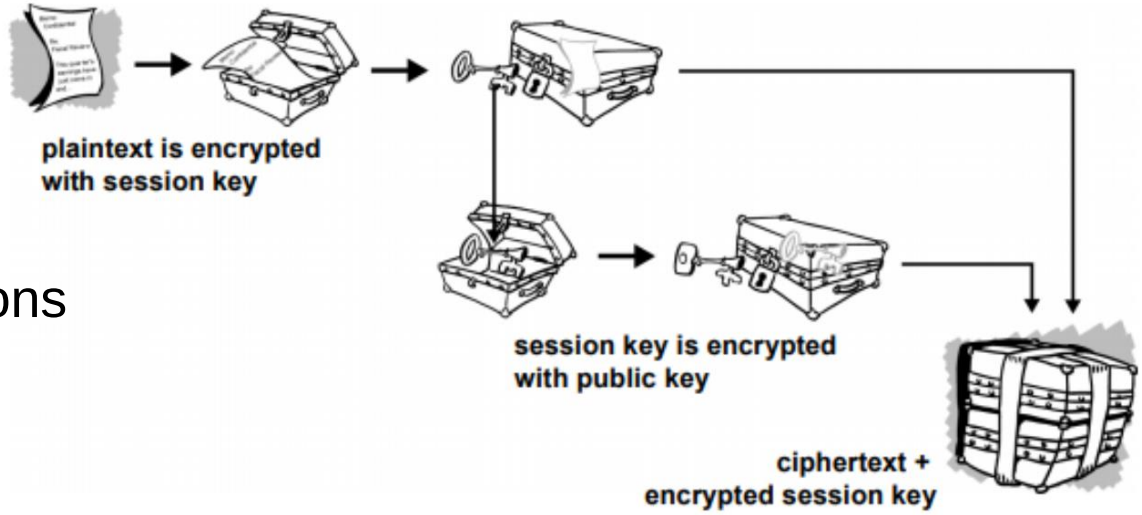
- A simple graphical explanation of how Diffie-Hellman key exchange works

- Good video explaining this: www.youtube.com/watch?v=3QnD2c4Xovk



Cryptography

- Asymmetric algorithms are slower and symmetric, so most implementations use a combination of both to ensure it is both fast and secure



  - Common implementations
    - SSL
    - PGP / GPG

Cryptographic Hashing

- Cryptographic Hash Function, aka One-Way Hashing
- The basic concept
  - Take an input of any length, process it in such a way that it is infeasible to reverse, and produce a fixed length output.
  - Example or changing one character produces a very different output
    - **TestNumber1** put through SHA-256 produces the output 541a6d07ae40626b61456912992b38b7e726d121d1eddfe47719cefe6811385e
    - **TestNumber2** also using SHA-256 produces the output 596924bec1ef084d5173fc18c8ae94e6083c08ecee6d57d83eaafcb83b221b3e
  - This helps protect integrity of data by identifying any changes

Cryptographic Hashing

- Example or changing one character produces a very different output
  - **TestNumber1** put through SHA-256 produces the output 541a6d07ae40626b61456912992b38b7e726d121d1eddfe47719cefe6811385e
  - **TestNumber2** also using SHA-256 produces the output 596924bec1ef084d5173fc18c8ae94e6083c08ecee6d57d83eaafcb83b221b3e
  - EXERCISE - These were generated on a web site implementation of SHA-256, but you can check the same output on your own computers
    - Cyber Chef has the recipe “SHA2” with a strength of 256
    - On Linux (like Kali): `echo -n TestNumber1 | sha256sum`
    - On Mac OS X: `echo -n TestNumber1 | shasum -a 256`
    - On Windows: `certutil -privatekey -hashfile <filename> sha256` (but this is for files, and not text input)
    - Or just use any number of web sites like http://www.fileformat.info/tool/hash.htm

Uses of Hashing

- Password authentication
  - Because the output of a hash is identical for the same input, hashes are commonly used for storing and verifying passwords.
  - When a user creates a new password, the authenticating system stores the hash output (for example, TestNumber1 from the previous slide)
  - When the user logs in later, they supply their password which is put through the same hash function, and the output is compared to the stored output.
  - If the hashes match, then the passwords must match.
  - This has advantages over storing the password in plaintext, in case the server is compromised and the hashes are exposed.

Uses of Hashing

- Verifying file integrity
  - When downloading files from the internet you may see a hash provided.
  - This allows you to put the downloaded file through the same hash and compare outputs.
  - This verifies that the file hasn’t been modified during download, and allows for the file to be available for download from multiple sites while the authoritative web site hosts the hash to be compared against.

Uses of Hashing

- Verifying file integrity
  - File Integrity Monitor (FIM) applications also use hashing of files to confirm integrity. This is done through the creation of a database of the hash output from all (or selected) files on a computer, to be compared against later. If any hashes change, the administrator is alerted. FIM systems are used to find modified files from attackers, or changes in configuration files.

Uses of Hashing

- Verifying file integrity
  - Exercise:
    Run sha256sum on a file, edit the file, then sha256sum it again
    - Open a terminal window on your Linux server
    - `nano fim-test.txt` (create the file with “Hello World!” , then save and exit)
    - `sha256sum fim-test.txt`
    - `nano fim-test.txt` (edit the file to say “Hello World?”)
    - `sha256sum fim-test.txt`

    To see what the output looks like for multiple files being hashed, run this command:
    - `sha256sum nmap/*`

Uses of Hashing

- Proof-of-work
  - BitCoin (and other cryptocurrencies) uses the work to create a partially known hash to prove that the end user (miner) performed the amount of work on their computer. For example, one proof of work is to have a computer try many different input combinations in order to produce an output value that has the first 20 bits being zero.
  - Similar proof of work systems have been created for a way to reduce spam. Legitimate email senders can have their computer perform the proof-of-work function for each email sent, which isn't too onerous for the typical small amount of email sent. Spammers however want to send email much faster and don't have the time or resources to generate the special hash for each email sent.

Cryptographic Hashing

- Types of Hashing Algorithms
  - MD5 – First published 1992 - Deprecated, do not use
    - Produces a 128-bit hash output, usually represented as 32 hex digits
    - 2109494ae833752b82ba786e7a4d7209
  - SHA-1 – First published 1995 - Deprecated, do not use
    - Produces a 160-bit hash output, usually represented as 40 hex digits
    - a316ec9b579abda6cc712490894619f47f38cbef

Cryptographic Hashing

- SHA-2 – First published 2001
  - Consists of 6 hash functions with outputs of 224, 256, 384, and 512 bits
  - The other 2 are SHA-512/224 and SHA-512/256 which are truncated versions of SHA-512 and are not commonly used. Added in 2012.
  - SHA-256 = 541a6d07ae40626b61456912992b38b7e726d121d1eddfe47719cefe6811385e
  - SHA-512 = 04ce124b492943eb9883cb2af654d89e02548e4f11bf5c9dff35217d63cfbed3b3c4125ecc8bf4270566f51c09c84aed21d4891ce2b1eb6bb2e4ccfc25dd9e35
  - SHA-2 functions are partially vulnerable to a type of attack called a “length extension attack”, so if you are currently using SHA-2 you should start looking at moving to SHA-3, bcrypt, PBKDF2, or scrypt. If you are using something weaker than SHA-2 you should skip SHA-2 and move to something stronger.

Cryptographic Hashing

- PBKDF2, bcrypt, scrypt, Argon2
  - I'm grouping these together as they all work on a similar basic principle.
  - For input you provide the plaintext password, a salt, and the number of iterations.
    - A salt is a random block of random characters that is prepended or appended to the plaintext password, thus making the password stronger before being put through the hash function
      - Salt can be added to other hash algorithms (e.g. SHA-1, SHA-256) to make them more secure.
      - Salts should be used and changed for each password being stored, so that 2 users with the same password won't have the same hash output.

Cryptographic Hashing

- PBKDF2, bcrypt, scrypt, Argon2
  - For input you provide the plaintext password, a salt, and the number of iterations.
    - Iterations are the number of times the hash function is performed before the output it stored. Remember the basic premise of hashes: going forwards is easy, going backwards it extremely difficult.

Cryptographic Hashing

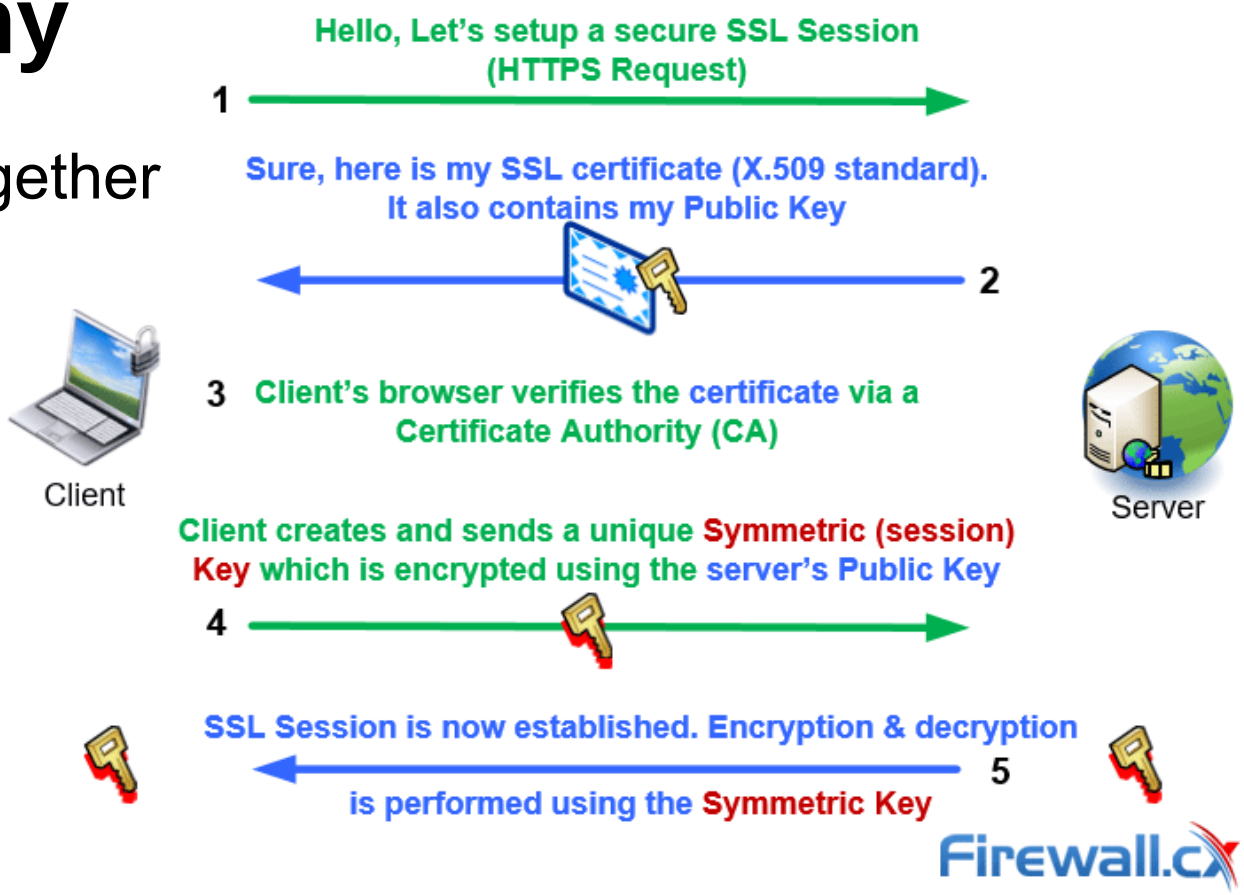
- PBKDF2, bcrypt, scrypt, Argon2
  - PBKDF2 (Password-Based Key Derivation Function 2) and bcrypt are older and more trusted, but use a fixed amount of memory to execute.
  - Scrypt is newer (so less trusted) but require larger amounts of memory, thus making it harder to attack using custom hardware or GPUs.
  - Argon2 was chosen as the winner from the Password Hashing Competition and is similarly resistant to attack using GPUs, but is also a newer algorithm and hasn't been tested or trusted as much yet.
    - Ref: https://password-hashing.net/

Cryptography

- SSL / TLS
  - Certificates are issued as part of a Public Key Infrastructure (PKI) with a hierarchical structure of Certificate Authorities (CA) and Intermediate CAs

Cryptography

- How it's all put together
  - SSL / TLS



Cryptography

- PGP
  - Each user creates their own certificate/key-pair and uses a web-of-trust model
  - Web-of-trust benefits from users signing each others' public keys, usually after verifying the person's identity and public key ID in person
  - Exercise: Create a PGP key pair
    - https://pgpkeygen.com/

- Reminder, the private key is supposed to be kept private

Cryptography

Adobe accidentally releases private PGP key

The firm's security team failed in a spectacular fashion.

By Charlie Osborne for Zero Day | September 25, 2017 -- 08:35 GMT (16:35 GMT-08:00) | Topic: Security

```
ZlRwd77Vmfc=
=QOc7
-----END PGP PUBLIC KEY BLOCK-----

-----BEGIN PGP PRIVATE KEY BLOCK-----
Version: Mailvelope v1.8.0
Comment: https://www.mailvelope.com

xcaGBFm/2KMBEADbwToJM3BCVE1OeC22HgVEqNEDppXzuD2dgfKuy0M4tx2L
De7GkPjo6AOsw4yi8bakLiidpw5B0J/AR1VtIjIDEmS0F9MRZIcV0UKyA5qV
```

Archived copy of the original leak is at: http://archive.is/h7qQ2

Attacks Against Cryptography

- Rainbow tables
  - Hashing algorithms create a fixed length output.
  - What if you pre-compute hashes for all known inputs (e.g. passwords 1-8 characters long, with upper/lower case letters and numbers). Then when you are presented with a hash, you can look it up to see what the plaintext input is.
  - Now you're got a Rainbow Table!
    - Note: rainbow tables are actually more complex, but the above description is good enough

Attacks Against Cryptography

- Rainbow tables
  - Generating rainbow tables requires a lot of computer effort, and a moderate amount of storage. But once created, they can be used again and again (and shared)
    - 1-8 characters long, using a-z.A-Z.0-9 = 127GB
      - Keyspace is 221,919,451,578,090 (221 trillion)
    - 1-9 characters long, using a-z.A-Z.0-9 = 690GB
      - Keyspace is 13,759,005,997,841,642 (13.8 quadrillion)

Cryptography

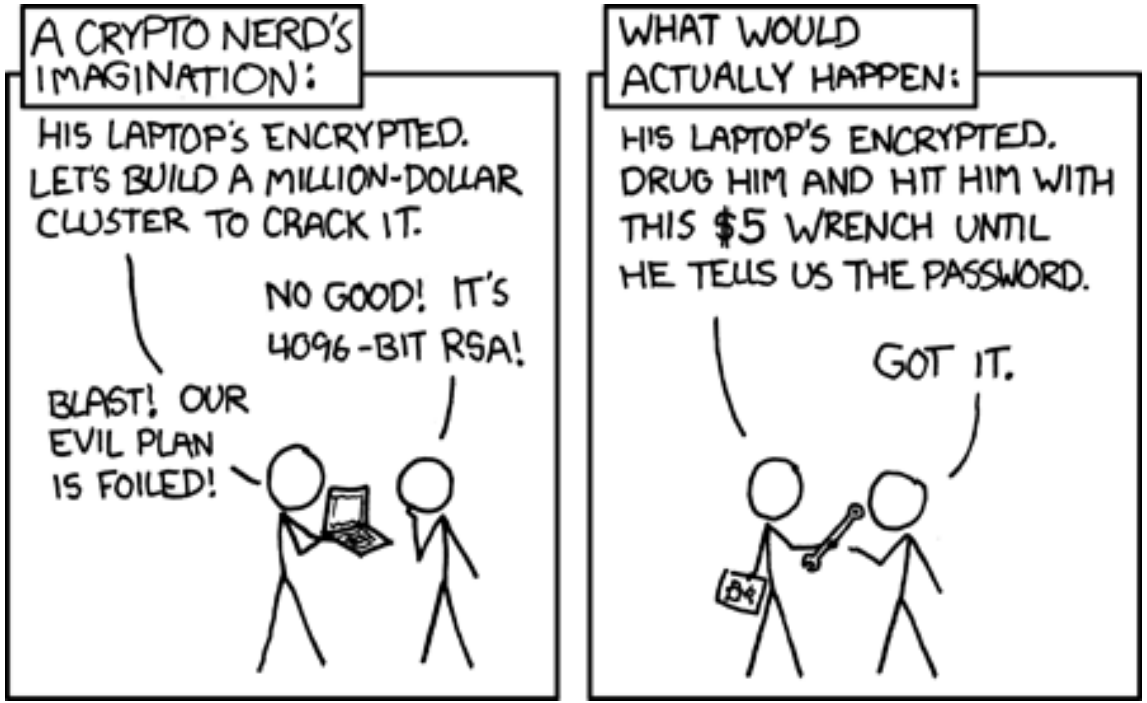
- Brute-force
  - If you have a fast-enough computer, you can just try running every possible input through an algorithm and seeing the output matches the ciphertext or hash you're trying to break.
  - CPUs have gotten faster, but GPUs with 100's of cores are faster
  - Brute force attacking a RAR file
    - 64 passwords per second using Intel Xeon E5 2603
    - 25,300 passwords per second using NVIDIA GeForce GTX 1080
    - Ref: https://www.elcomsoft.com/edpr.html

Cryptography

- Brute-force
  - Combine 10 x GTX 1080 Ti's in one machine and you can do the following:
    - SHA-1 – 113.5 billion hashes per second
    - SHA-512 – 15 billion hashes per second
    - SHA-3 – 11.7 billion hashes per second
    - MSSQL (2012, 2014) – 14.2 billion hashes per second
    - bcrypt – 218.9 thousand hashes per second
    - All this for $16,500 USD (not including the massive power bill for burning 4 kilowatts of power!)
    - Ref: https://www.servethehome.com/deeplearning11-cracking-passwords-with-10x-nvidia-geforce-gtx-1080-ti-gpus/

Cryptography

- Brute-force

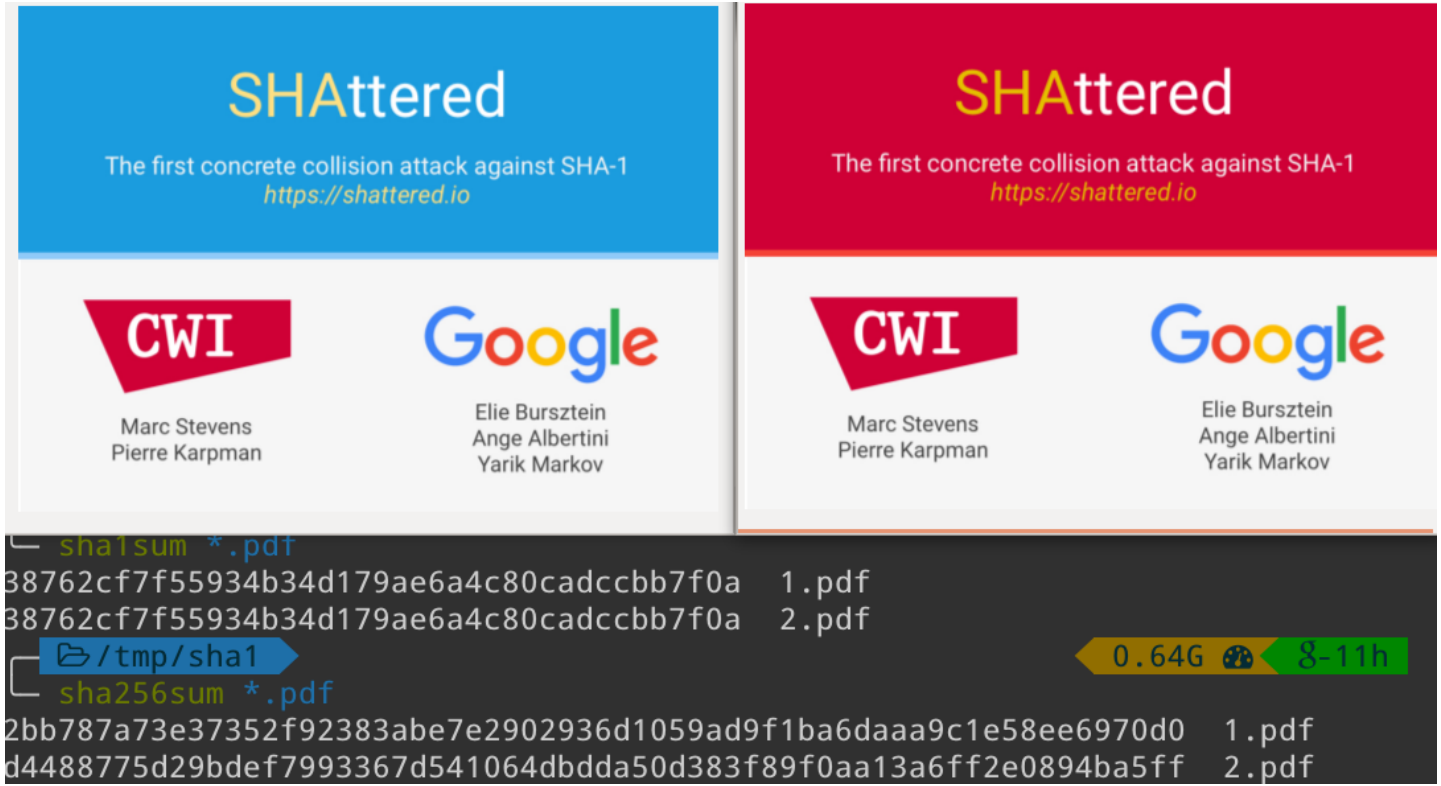


Ref: https://xkcd.com/538/

Cryptography

- Hash collisions
  - Because hashes have a fixed length output, it is mathematically possible for 2 inputs to produce the same output.
    A good hashing algorithm makes this extremely hard to do.
  - MD5 had a weakness found in 1996, and a collision attack published in 2004
  - SHA-1 had theoretical attacks published in 2005, and the NIST officially deprecated SHA-1 in 2011

Cryptography



Ref: https://shattered.io/