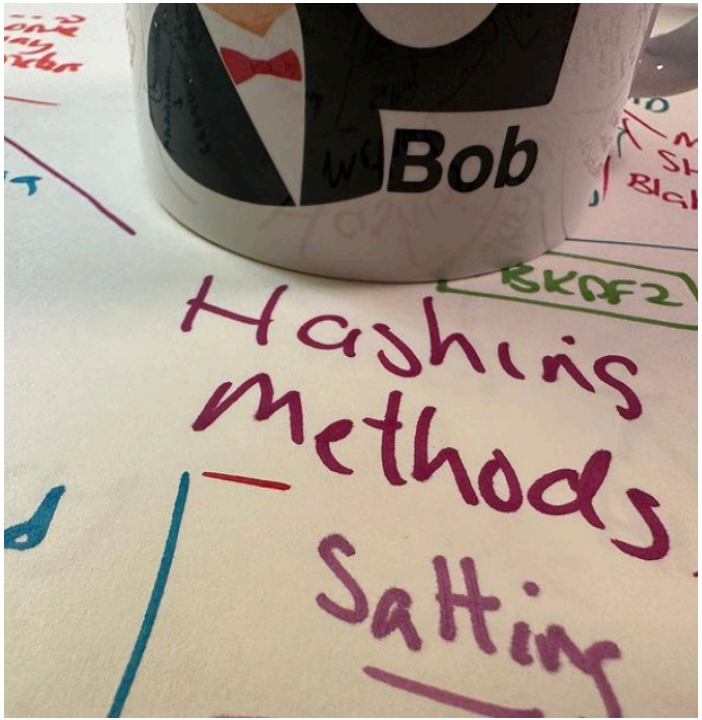
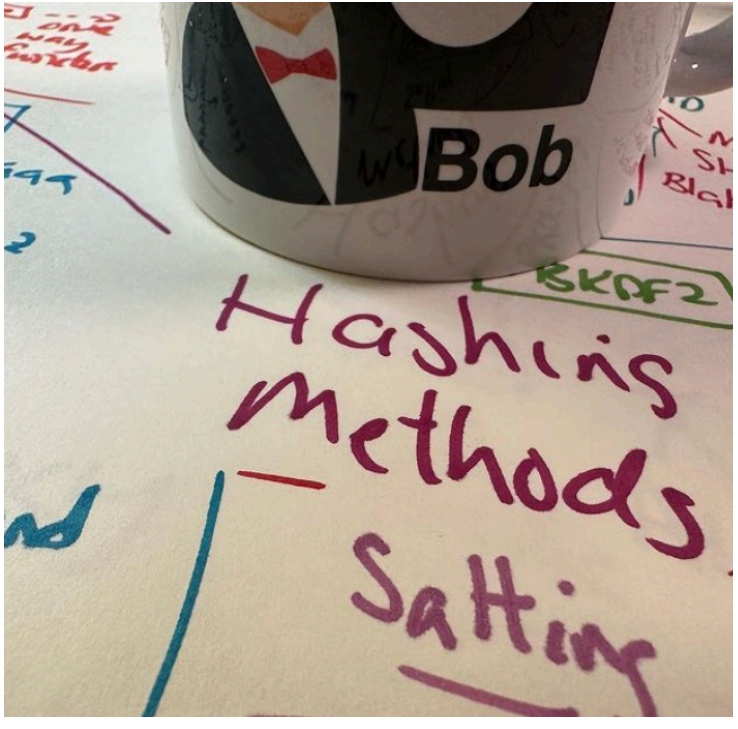
No	Date	Subject	Lab
2	23 Jan 2024	Ciphers and Fundamentals [Unit]	[Lab] [Demo]
3	30 Jan 2024	Symmetric Key [Unit]	[Lab]
4	6 Feb 2024	Hashing and MAC [Unit]	[Lab Bruce Schneier]
5	13 Feb 2024	Asymmetric (Public) Key [Unit]	[Lab]
6	20 Feb 2024	Key Exchange [Unit]	[Lab Whitfield Diffie]
7	27 Feb 2024	Reading Week (Revision lecture)	Mini-project [Here] / Coursework
8	5 Mar 2024	Digital Signatures and Certificates [Unit]	[Lab Vincent Rijmen]
9	12 Mar 2024	Test (Units 1-5) 40% of overall mark [Here]	
10	19 Mar 2024	Tunnelling [Unit]	[Lab Marty Hellman
11	10 Apr 2024	Blockchain [Unit]	[Lab]
12	16 Apr 2024	Future Cryptography [Unit]	[Lab]
13	23 Apr 2024	Host/Cloud Security [Unit]	[Lab]
14	30 Apr 2024		
15	7 May 2024	Coursework Hand-in - 60% of overall mark (15 May) [Coursework]	



Unit 3: Hashing

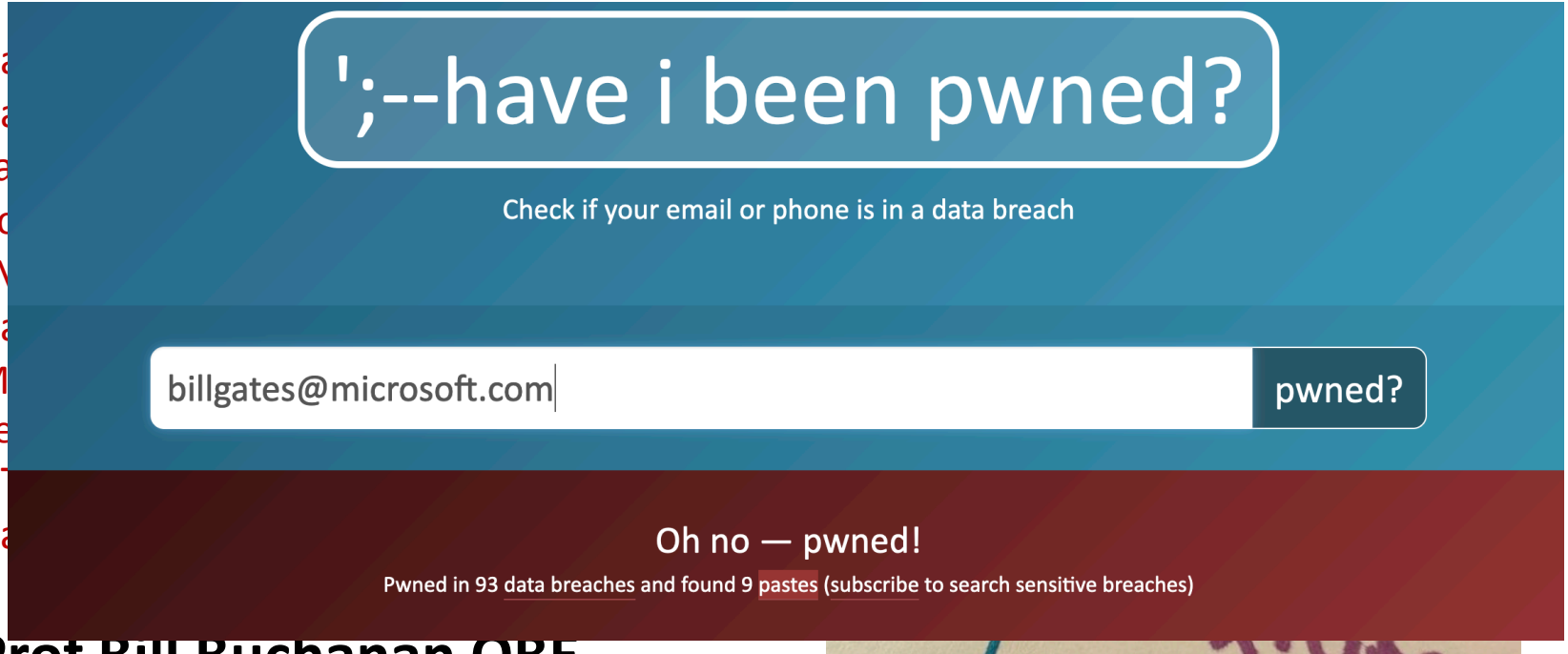
Hashing Types.
Hashing Methods.
Salting.
Collisions.
LM and NTLM Hashes (Windows).
Hash Benchmarks.
Message Authentication Codes (MACs).
Key Derivation Function (KDF)
OTP/HOTP.
Hashcat **(for practical work)**



Prof Bill Buchanan OBE
https://asecuritysite.com/hash

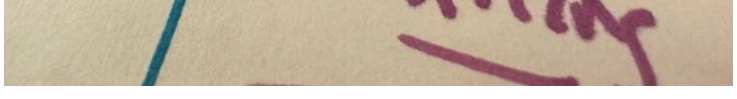
Unit 3: Hashing

Ha
Ha
Sa
Co
LM
Ha
M
Ke
OT
Ha



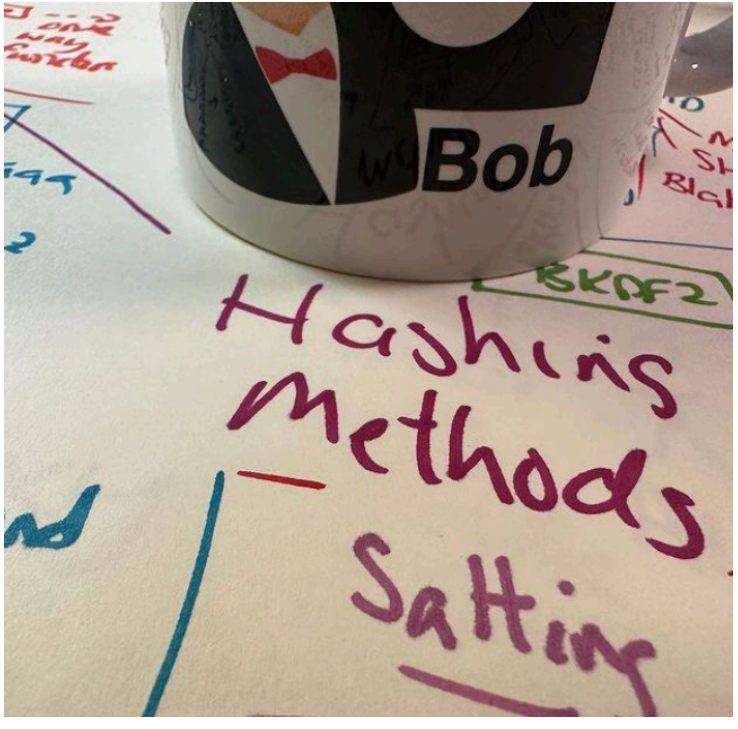
Prof Bill Buchanan OBE

https://asecuritysite.com/hash



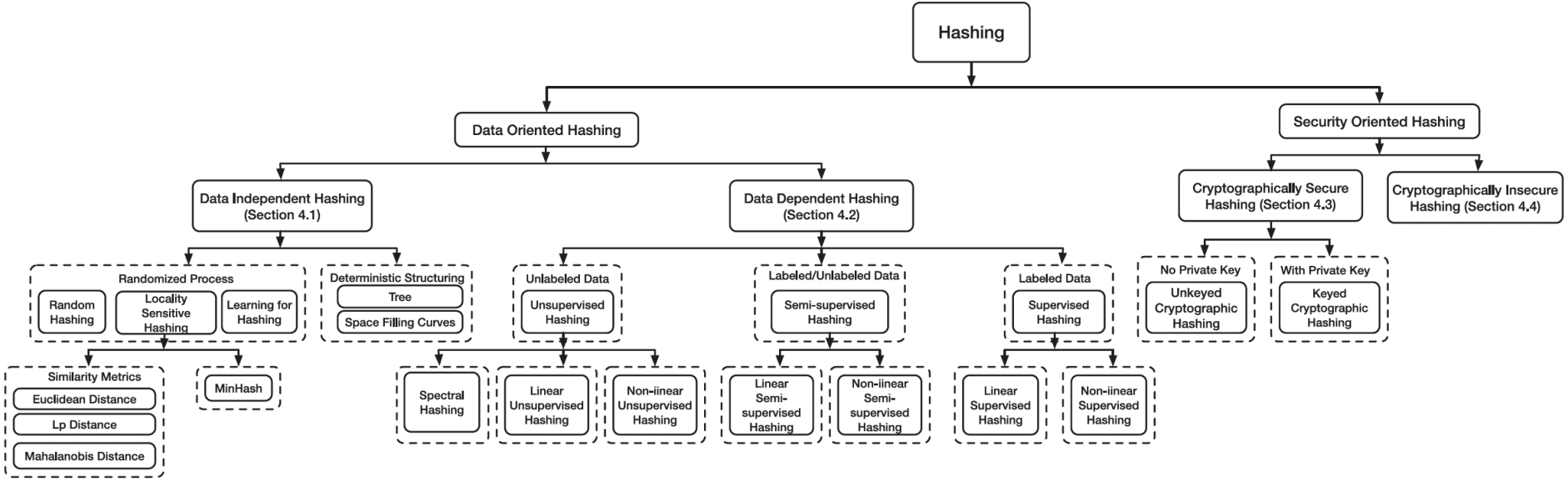
Unit 3: Hashing

Hashing Types.
Hashing Methods.
Salting.
Collisions.
LM and NTLM Hashes (Windows).
Hash Benchmarks.
Message Authentication Codes (MACs).
Key Derivation Function (KDF).
OTP/HOTP.
Hashcat (for practical work)

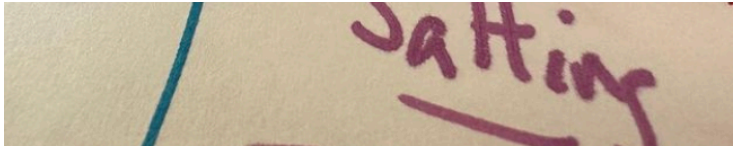


Prof Bill Buchanan OBE
https://asecuritysite.com/hash

Unit 3: Hashing



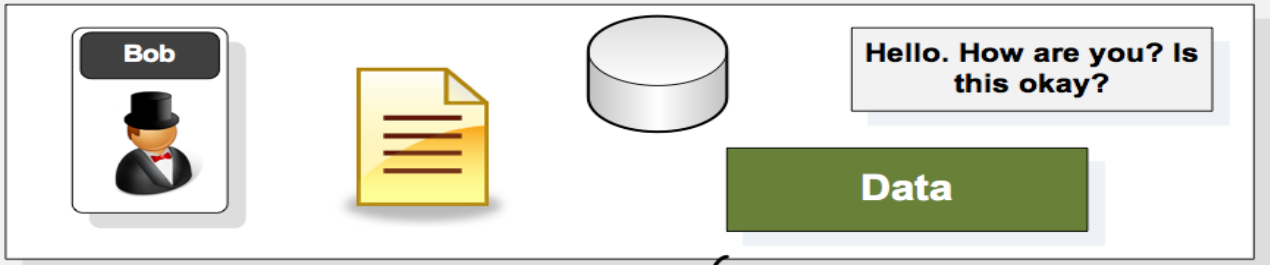
Hashcat (for practical work)



Prof Bill Buchanan OBE

https://asecuritysite.com/hash

How do we get a finger-print for data?



With a fingerprint we can hopefully tell if Eve has modified any of the data

Solved by Prof Ron Rivest with the MD5 hash signature.

Hex

hello	5D41402ABC4B2A76B9719D911017C592
Hello	8B1A9953C4611296A827ABF8C47804D7
How are you?	04E35EB3E4FCB8B395191053C359CA0E
Napier	8F83571F9324AE4E23D773753055C7B6

Base64

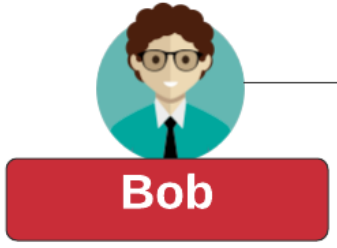
hello	XUFAKrxLKna5cZ2REBfFkg==
Hello	ixqZU8RhEpaoJ6v4xHgE1w==
How are you?	BONes+T8uLOVGRBTw1nKDg==
Napier	j4NXH5Mkrk4j13N1MFXHtg==

Hex

hello	AAF4C61DDCC5E8A2DABEDE0F3B482CD9AEA9434D
Hello	F7FF9E8B7BB2E09B70935A5D785E0CC5D9D0ABF0
How are you?	3031897E282167593FBB4DBE81DC48EBBE9A002D
Napier	BF81B135A5687766F4F464764EAC38CA8A4EBABA

Base64

hello	qvTGHdzF6KLavt4PO0gs2a6pQ00=
Hello	9/+ei3uy4Jtwk1pdeF4MxdnQq/A=
How are you?	MDGJfighZ1k/u02+gdxl676aAC0=
Napier	v4GxNaVod2b09GR2Tqw4yopOuro=



Bob

MD5 Hash
(128-bit hash)

There are two kinds of cryptography in this world: cryptography that will stop your kid sister from reading your files, and cryptography that will **stop** major governments from reading your files. Cryptography is typically bypassed, not penetrated. Security is a process, not a product. Beware of Snake Oil Cryptography.

14375C20F07A9DF2EBDE78F063C5AD7C

FDdcIPB6nfLr3njwY8WtfA==

There are two kinds of cryptography in this world: cryptography that will stop your kid sister from reading your files, and cryptography that will **top** major governments from reading your files. Cryptography is typically bypassed, not penetrated. Security is a process, not a product. Beware of Snake Oil Cryptography.

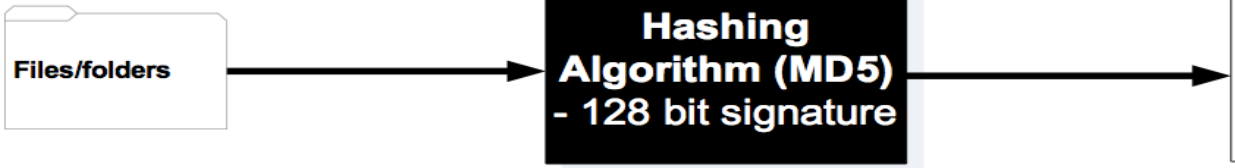
174F3AB19D8047DAE398CE620407B60B

F086sZ2AR9rjmM5iBAe2Cw==

OpenSSL

```
root@kali:~# echo -n “hello” | openssl md5
(stdin)= 5d41402abc4b2a76b9719d911017c592

root@kali:~# echo -n “hello” | md5sum
5d41402abc4b2a76b9719d911017c592  -

root@kali:~# openssl md5 pw
MD5(pw)= 859b6a9be3b45262c4414bd1696ba91b

root@kali:~# md5sum pw
859b6a9be3b45262c4414bd1696ba91b  pw

Hash methods supported:
md2          md4          md5          rmd160       sha
sha1
```

Author: Prof Bill Buchanan



Hash signature

- Hash signatures are used to gain a signature for files, so that they can be checked if they have been changed.

```
[Path] / filename                    MD 5 sum
-------------------------------------------------------------------
[C:\Windows\System32\]
12520437.cpx                          0a0feb9eb28bde8cd835716343b03b14
12520850.cpx                         d 69ae057cd82d04ee7d311809abefb2a
8point1.wav                          beab 165fa58ec5253185f32e124685d5
aaclient.dll                         ad 45dedfdcf69a28cbaf6a2ca84b5f1e
AC3ACM.acm                            59683d1e4cd0b1ad6ae32e1d627ae25f
Ac3audio.ax                           4b87d889edf278e5fa223734a9bbe79a
ac3filter.cpl                         10b27174d46094984e7a05f3c36acd2a
accessibilitycpl.dll                 ac 4cecc86eeb8e1cc2e9fe022cff3ac1
ACCTRES.dll                           58f57f2f2133a2a77607c8ccc9a30f73
acledit.dll                           0bcee3f36752213d1b09d18e69383898
  . . .
ZSHP1020.CHM                         c 671ec
ZSHP1020.EXE                          96e45a
ZSHP1020.HLP                         a 07693
ZSPOOL.DLL                           fae 332
ZTAG.DLL                              7ca836
ZTAG32.DLL                            27b026
```

```
[Path] / filename                    MD 5 sum
-------------------------------------------------------------------
[C:\Windows\system32\]
12520437.cpx                         Cg/rnrKL3ozYNXFjQ7A7FA==
12520850.cpx                          1prgV82C0E7n0xGAmr77Kg==
8point1.wav                          vqsWX 6WOxSUxhfMuEkaF1Q==
aaclient.dll                         rUXe 39z2mijLr2osqEtfHg==
AC3ACM.acm                           WWg 9HkzQsa1q4y4dYnriXw==
Ac3audio.ax                          S 4fYie3yeOX6Ijc0qbvnmg==
```

Message Hash

Authentication

MD5 hash algorithm

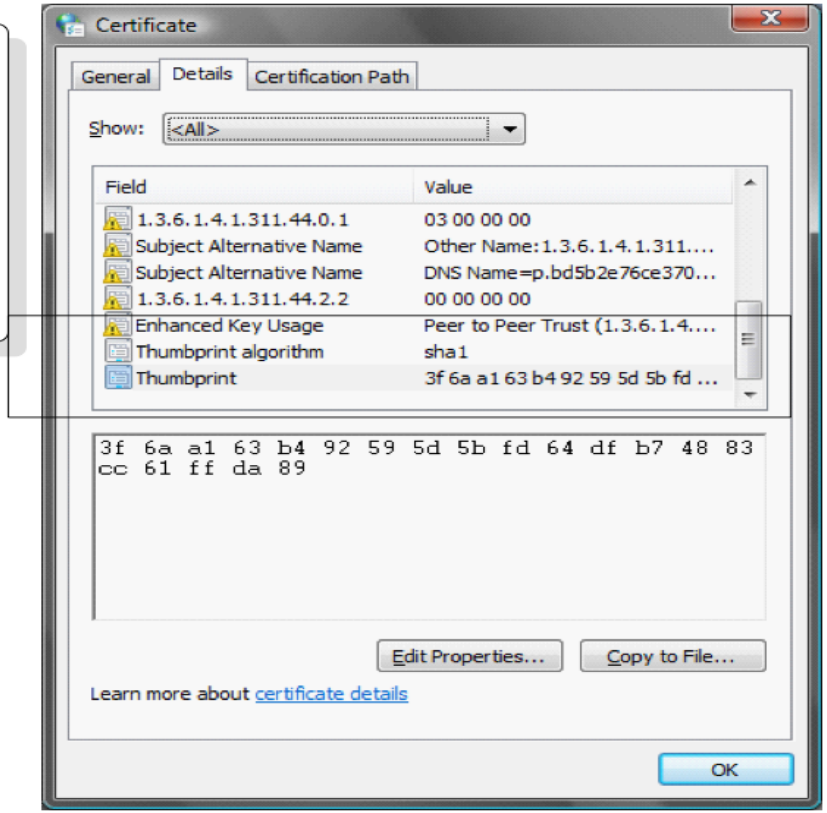
Files/folders

Hashing Algorithm (MD5) - 128 bit signature

Hash signature

- Hash signatures are used to identify that a file/certificate has not been changed.

Bob



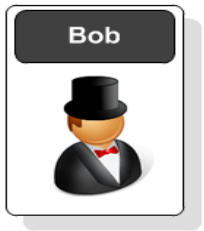
The digital certificate has an SHA-1 hash thumbprint (3f6a...89) which will be checked, and if the thumbprint is different, the certificate will be invalid.

Author: Prof Bill Buchanan

One-way hash

- Hashes are used for digital fingerprints (see the next unit) and for secure password storage.
- Typical methods are NT hash, MD4, MD5, and SHA-1.

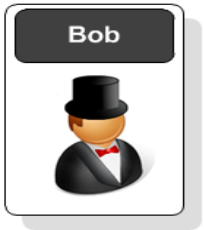
Windows login/ authentication



mypass → NT hash (MD4) → fa1bfa14fa13fa12fa10fa1ffa14fa12

NT-password hash for Windows NT, XP and Vista

Cisco password storage (MD5)



mypass → 

```
# config t
(config)# enable secret test

Current configuration : 542 bytes
!
version 12.1
no service single-slot-reload-enable
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname Router
!
enable secret 5 $1$/Nwk$knsEQYxZVenGjWOGj/TGk0
```

MD5 encoded password →

One-way hash

- Hashing suffers from dictionary attacks, thus it is important that any passwords are not standard words, such as to change **password** for **pA55wOrd**.

Windows login/ authentication

mypass → NT hash (MD4) → fa1bfa14fa13fa12fa10fa1ffa14fa12

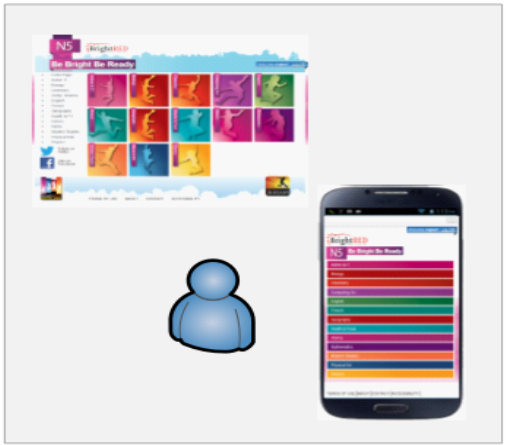
NT-password hash for Windows NT, XP and Vista

Hashing suffers from **dictionary attacks** where the signatures of well know words are stored in a table, and the intruders does a lookup on this

mypast	effahd13fa12fa10fgffa1ffa14fa144
mypass	fa1bfa14fa13fa12fa10fa1ffa14fa12
mypose	ff12189043210954defff0123444512d
test1	aabbfce023215546dfeddd0101001cd

Risk 4: One Password Fits All

Edinburgh Napier UNIVERSITY



Adobe

150 million accounts compromised

#	Count	Ciphertext	Plaintext
1.	1911938	EQ7fIpT7i/Q=	123456
2.	446162	j9p+HwtWWT86aMjgZFLzYg==	123456789
3.	345834	L8qbAD3jl3jioxG6CatHBw==	password
4.	211659	BB4e6X+b2xLioxG6CatHBw==	adobe123
5.	201580	j9p+HwtWWT/ioxG6CatHBw==	12345678
6.	130832	5djv7ZCI2ws=	qwerty
7.	124253	dQi0asWPYvQ=	1234567
8.	113884	7LqYzKVeq8I=	111111
9.	83411	PMDTbP0LZxu03SwrFUvYGA==	photoshop
10.	82694	e6MPXQ5G6a8=	123123

TJ Maxx / Marshalls

47 million accounts

LinkedIn

6.5 million accounts (June 2013)

Windows Live · f · YouTube

One account hack … leads to others

1 million accounts – in plain text. 77 million compromised

Dropbox compromised 2013

citigroup

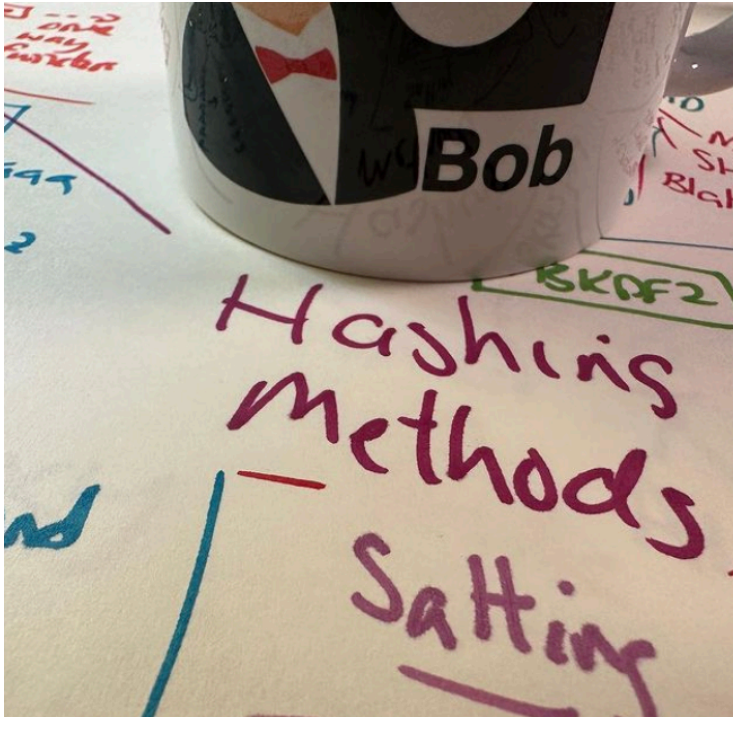
200,000 client accounts

Brute Force - How many hash codes?

- 7 digit password with [a-z] … how many?
  - Ans:
  - Time to crack - 100 billion per second:
- 7 digit with [a-zA-z] … how many?
  - Ans:
  - Time to crack – 100 billion per second:
- 7 digit with [a-zA-z!@#$%^&*()] … how many?
  - Ans:
  - Time to crack – 100 billion per second:

Unit 3: Hashing

Hashing Types.
Hashing Methods.
Salting.
Collisions.
LM and NTLM Hashes (Windows).
Hash Benchmarks.
Message Authentication Codes (MACs).
Key Derivation Function (KDF).
OTP/HOTP.
Hashcat



Prof Bill Buchanan OBE
https://asecuritysite.com/hash

LM Hash

LM Hash. LM Hash is used in many version of Windows to store user passwords that are fewer than 15 characters long.

SHA-3

SHA-3. SHA-3 was known as Keccak and is a hash function designed by Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. MD5 and SHA-0 have been shown to be susceptible to attacks, along with theoretical attacks on SHA-1. NIST thus defined there was a need for a new hashing method which did not use the existing methods for hashing, and setup a competition for competing algorithms. In October 2012, Keccak won the NIST hash function competition, and is proposed as the SHA-3 standard.

Tiger

Bcrypt

Bcrypt. This creates a hash value which has salt.

RIPEMD

RIPEMD (RACE Integrity Primitives Evaluation Message Digest) and GOST. RIPEM160. RIPEMD is a 128-bit, 160-bit, 256-bit or 320-bit cryptographic hash function, and was created by Hans Dobbertin, Antoon Bosselaers and Bart Preneel. It is used on TrueCrypt, and is open source. The 160-bit version is seen as an alternative to SHA-1, and is part of ISO/IEC 10118

Tiger. Tiger is a 192-bit hash function, and was designed by Ross Anderson and Eli Biham in 1995. It is often used by clients within Gnutella file sharing networks, and does not suffer from known attacks on MD5 and SHA-0/SHA-1. Tiger2 is an addition, in which the message is padded with a byte of 0x80 (in a similar way to MD4, MD5 and SHA), whereas in Tiger it is 0x01. Otherwise the two methods are the same in their operation.

Author: Prof Bill Buchanan

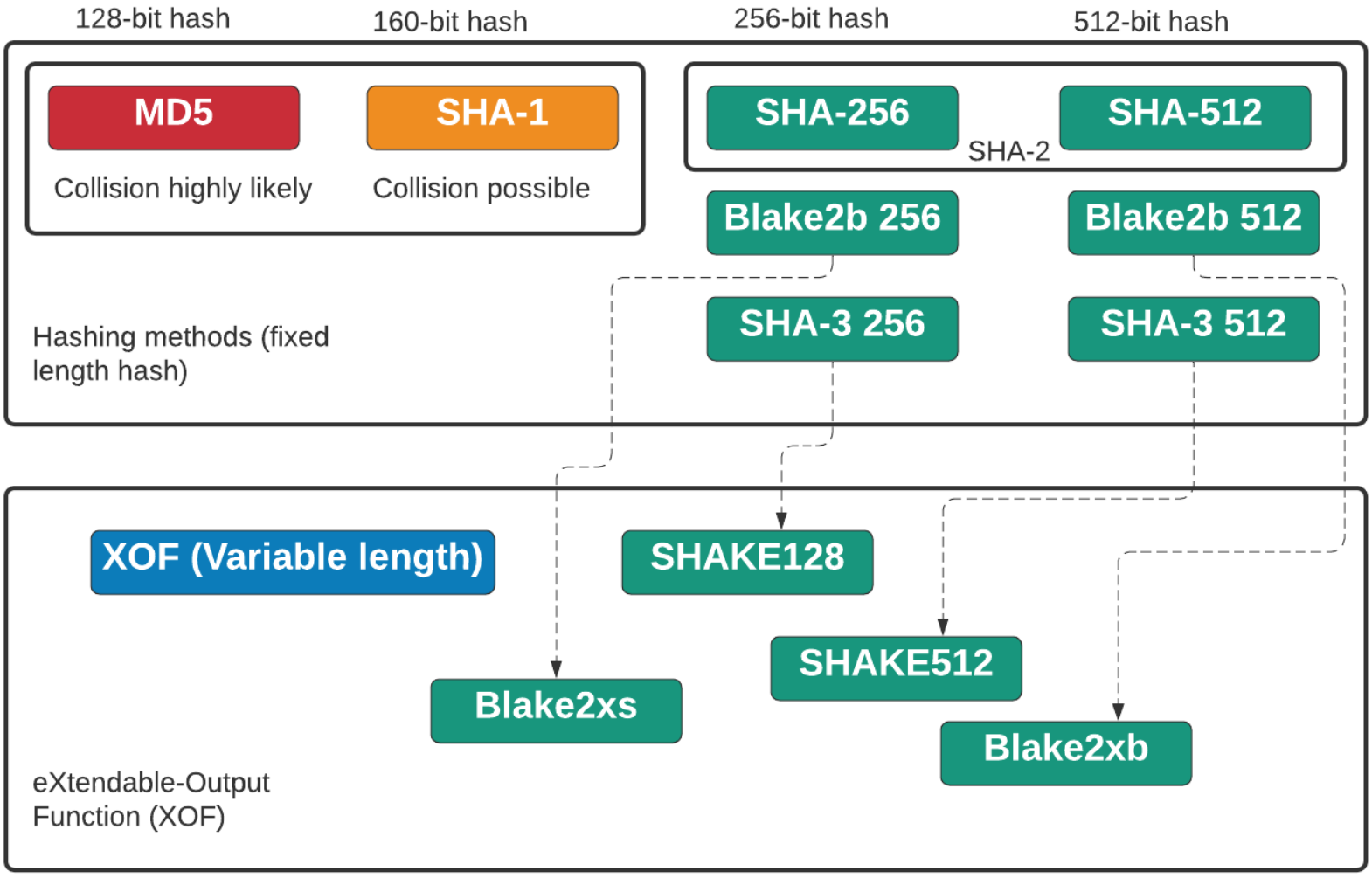
Murmur

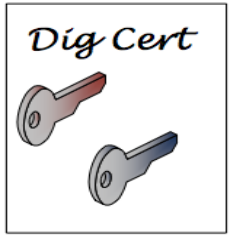
While hashing methods such as MD5 and SHA-1 use cryto methods, the Murmur and FNV hashes uses a non-cryptographic hash function. The Murmur hash, designed by Austin Appleby, uses a non-cryptographic hash function. This can be used for general hash-based lookups. It has a good performance compared with other hashing methods, and generally provide a good balance between performance and CPU utilization. Also it performs well in terms of hash collisions.

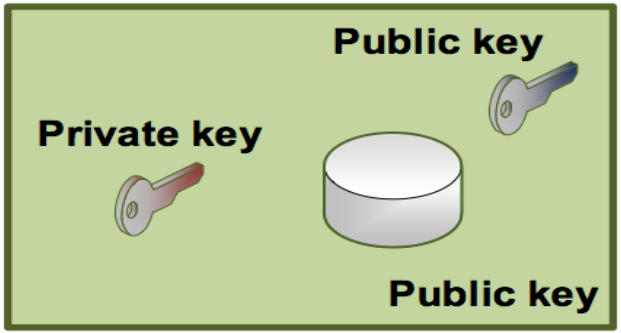
FNV

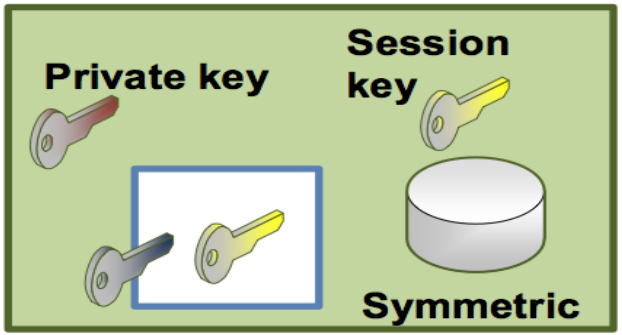
FNV (Fowler–Noll–Vo) is a 64-bit non-cryptographic hash function developed by Glenn Fowler, Landon Curt Noll, and Phong Vo. There are two main versions, of which 1a is the most up-to-date version.

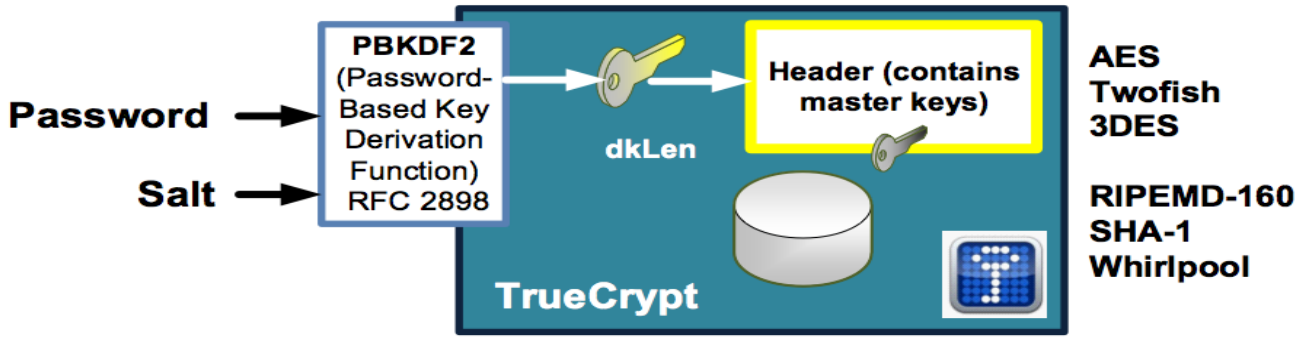
Author: Prof Bill Buchanan


128-bit hash
160-bit hash
256-bit hash
512-bit hash
MD5
SHA-1
Collision highly likely
Collision possible
SHA-256
SHA-512
SHA-2
Blake2b 256
Blake2b 512
SHA-3 256
SHA-3 512
Hashing methods (fixed length hash)
XOF (Variable length)
SHAKE128
SHAKE512
Blake2xs
Blake2xb
eXtendable-Output Function (XOF)









AES
Twofish
3DES

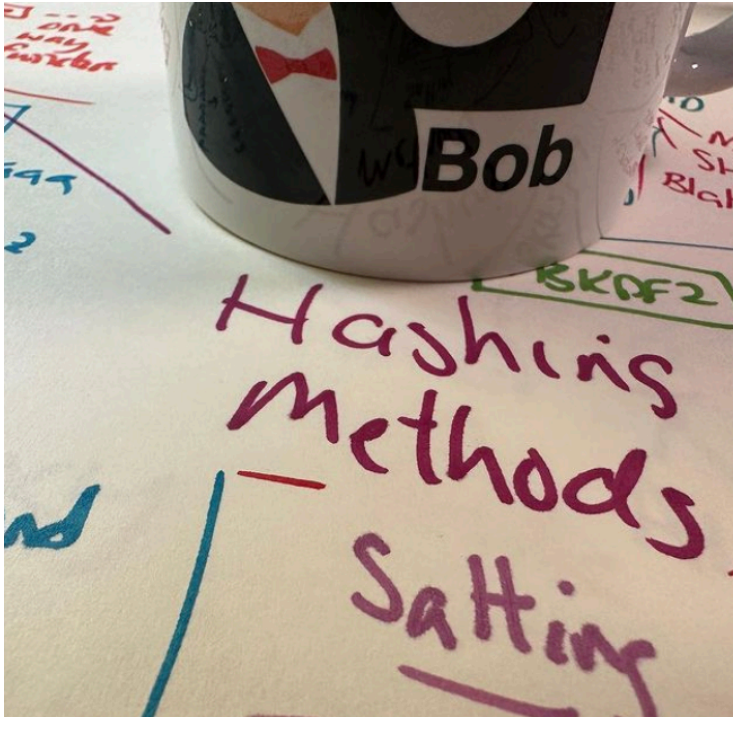
RIPEMD-160
SHA-1
Whirlpool

DK = PBKDF2(PRF, Password, Salt, c, dkLen)
DK = PBKDF2(HMAC−SHA1, passphrase, ssid, 4096, 256)

Encrypting disks

Unit 3: Hashing

Hashing Types.
Hashing Methods.
Salting.
Collisions.
LM and NTLM Hashes (Windows).
Hash Benchmarks.
Message Authentication Codes (MACs).
Key Derivation Function (KDF)
OTP/HOTP.
Secret Shares.



Prof Bill Buchanan OBE
http://asecuritysite.com/crypto03
http://asecuritysite.com/encryption

Adding salt

- Salt increases the range of the possible signatures

mypass → NT hash (MD4) → fa1bfa14fa13fa12fa10fa1ffa14fa12

NT-password hash for Windows NT, XP and Vista

Salt increase the range of the signatures

mypast →

effahd13fa12fa10fgffa1ffa14fa144

caaahdd3fa12ccfae342345500011aff

Ddde43296945031040 3010d0ae000100

password

1fred$bATAk8UUH/IDAp9sd6IUv/

1

fred

bATAk8UUH/IDAp9sd6IUv/

password

bATAk8UUH/IDAp9sd6IUv/

fred

```
C:\openssl>openssl passwd -1 -salt fred password
$1$fred$bATAk8UUH/IDAp9sd6IUv/
```

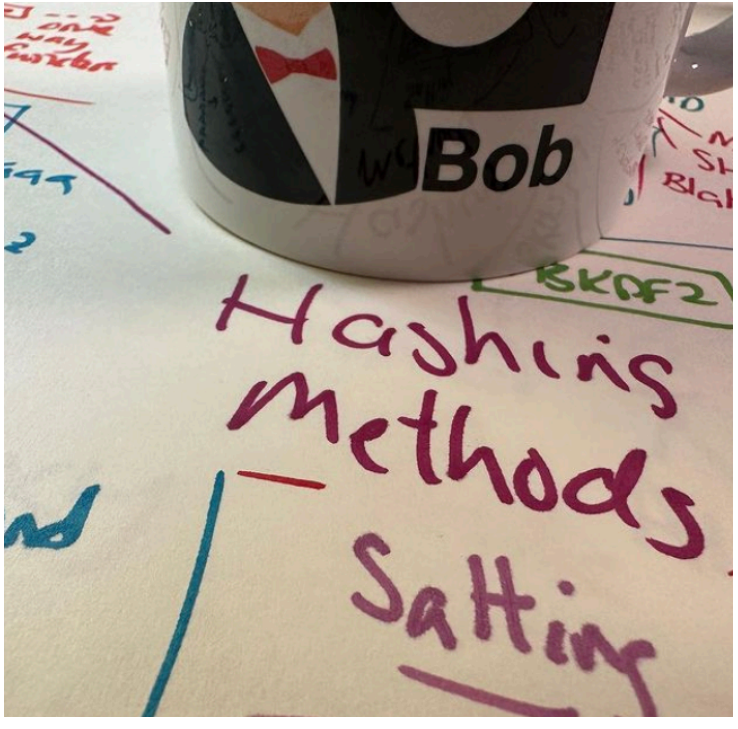
Author: Prof Bill Buchanan


```
# cat /etc/shadow
root:$1$Etg2ExUZ$F9NTP7omafhKIlqaBMqng1:15651:0:99999:7:::
# openssl passwd -1 -salt Etg2ExUZ redhat
$1$Etg2ExUZ$F9NTP7omafhKIlqaBMqng1
```

```
$ openssl version
OpenSSL 1.0.1f 6 Jan 2014

$ openssl dgst -md5  file
MD5(file)= b1946ac92492d2347c6235b4d2611184

$ openssl genrsa -out mykey.pem 1024
Generating RSA private key, 1024 bit long modulus
.....................................................................++++++
...++++++
e is 65537 (0x10001

$ openssl rsa -in mykey.pem -pubout > mykey.pub
writing RSA key

$ cat mykey.pub
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDXv9HSFkpM+ZoOQcpdHBZiUwX8
EzIKm0nsgjc5ZTYVaF9CMLtmKoTzep7aQX9o9nKepFt1kQ73Ta9vOPd6CX61/cgY
Xy2tShw0imrtFaVDFjX+7kLmc0uWbFFCoZMtJxIaXaa9SV2kARxOCTJ2uOjRTCCe
XU09IJGHnIhSNJeIJQIDAQAB
-----END PUBLIC KEY-----

$ cat /etc/shadow
root:$1$Etg2ExUZ$F9NTP7omafhKIlqaBMqng1:15651:0:99999:7:::

$ openssl passwd -1 -salt Etg2ExUZ redhat
$1$Etg2ExUZ$F9NTP7omafhKIlqaBMqng1
```

Unit 3: Hashing

Hashing Types.
Hashing Methods.
Salting.
Collisions.
LM and NTLM Hashes (Windows).
Hash Benchmarks.
Message Authentication Codes (MACs).
Key Derivation Function (KDF)
OTP/HOTP.
Hashcat.



Prof Bill Buchanan OBE
https://asecuritysite.com/hash

A major factor with hash signatures is:

- **Collision.** This is where another match is found, no matter the similarity of the original message. This can be defined as a **Collision attack**.
- **Similar context**. This is where part of the message has some significance to the original, and generates the same hash signature. The can be defined as a Pre-image attack.
- **Full context**. This is where an alternative message is created with the same hash signature, and has a direct relation to the original message. This is an extension to a Pre-image attack.

In 2006 it was shown that MD5 can produce collision within less than a minute.

A 50% probability of a collision is:

$$\sqrt{N(signatures)} = \sqrt{2^n} = 2^{\frac{n}{2}}$$

where n is the number of bits in the signature. For example, for MD5 (128-bit) the number of operations that would be required for a better-than-50% chance of a collision is:

$$2^{64}$$

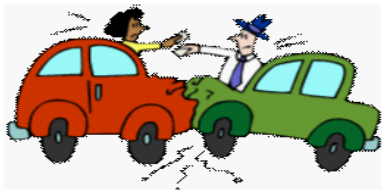
Note, in 2006, for SHA-1 the best time has been 18 hours

d131dd02c5e6eec4693d9a0698aff95c
2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325 71415a
085125e8f7cdc99fd91dbdf280373c5b
d8823e3156348f5bae6dacd436c919c6
dd53e2b487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080a80d1e
c69821bcb6a8839396f9652b6ff72a70

d131dd02c5e6eec4693d9a0698aff95c
2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a
085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6
dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b6708280d1e
c69821bcb6a8839396f965ab6ff72a70

The MD5 signature gives the same result

79054025255FB1A26E4BC422AEF54EB4



Nat McHugh

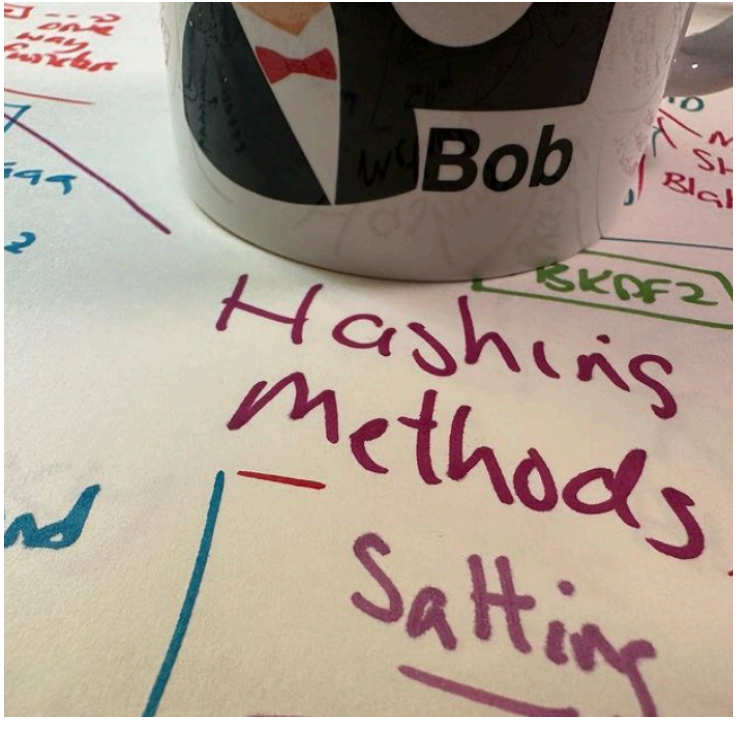
- 10 hours of computing on the Amazon GPU Cloud.
- Cost: 60 cents
- Used: Hashcat (on CUDA)
- Birthday attack: A group size of only 70 people results in a 99.9% chance of two people sharing the same birthday.
- M-bit output there are 2^m messages, and the same hash value would only require 2^(m/2) random messages. 18,446,744,073,709,551,616.

C:\openssl>openssl md5 hash01.jpg
MD5(hash01.jpg)= **e06723d4961a0a3f950e7786f3766338**

C:\openssl>openssl md5 hash02.jpg
MD5(hash02.jpg)= **e06723d4961a0a3f950e7786f3766338**

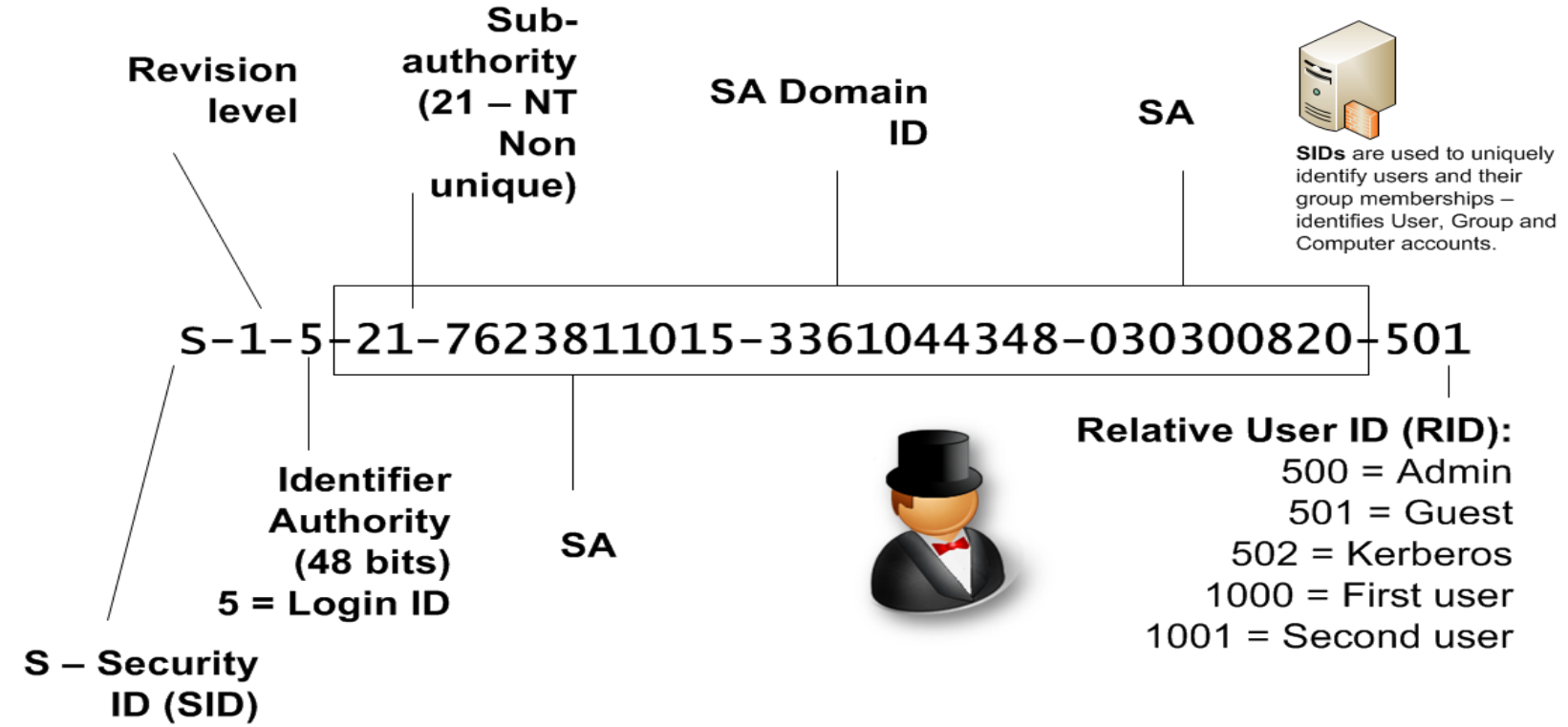
Chapter 3: Hashing

Hashing Types.
Hashing Methods.
Salting.
Collisions.
LM and NTLM Hashes (Windows).
Hash Benchmarks.
Message Authentication Codes (MACs).
Key Derivation Function (KDF)
OTP/HOTP.
Hashcat.

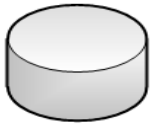


Prof Bill Buchanan OBE
https://asecuritysite.com/hash

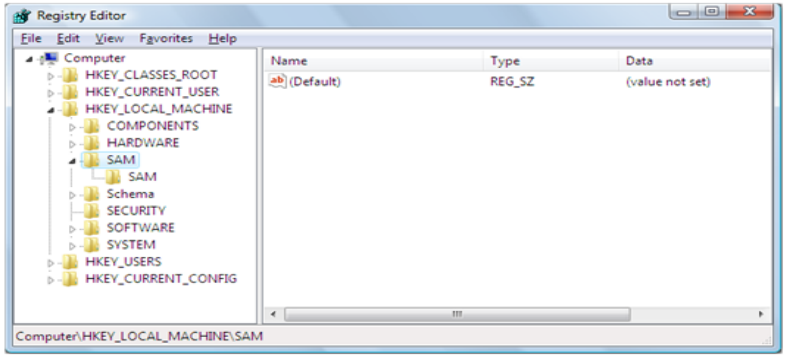
```
C:> user2sid \pluto guest
S-1-5-21-7623811015-3361044348-030300820-501
C:> sid2user 5 21 7623811015 3361044348 030300820 500
Name is Fred
Domain is PLUTO
```

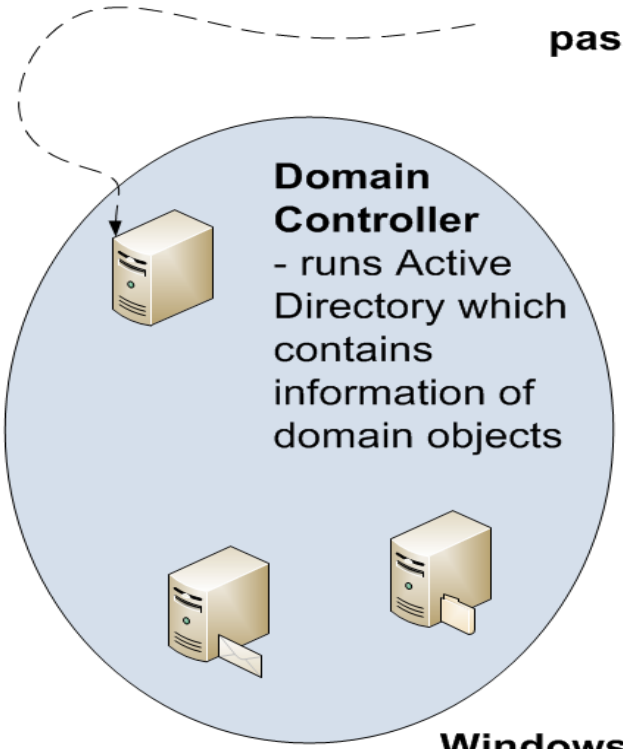


HKLM\SAM



SAM Database (stores usernames and passwords)



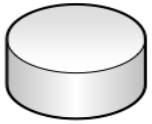


Local Authority Subsystem (Lsass) – Windows Security mechanism – Attached by Sasser Worm which exploited a buffer overflow

Responsible for local security policy

- Controls access.
- Managing password policies.
- User authentication.
- Audit messages.

SAM



Registry:
HKEY_LOCAL_MACHINE\SAM

- LM Hash (Windows XP, 2003)
- NTLMv2 (Windows 7, 8, etc) – connect to Active Directory
- NTLM (Windows 7, 8, etc) – No salt

```
C:\Windows\System32\config>dir
 Volume in drive C has no label.
 Volume Serial Number is A2B3-7C7A

 Directory of C:\Windows\System32\config
05-Oct-14  05:52 PM           262,144 SAM
05-Oct-14  05:56 PM           262,144 SECURITY
05-Oct-14  08:39 PM       149,946,368 SOFTWARE
05-Oct-14  08:40 PM        15,728,640 SYSTEM
```

- bkhive - dumps the syskey bootkey from a Windows system hive.
- samdump2 - dumps Windows 2k/NT/ XP/Vista password hashes.

hashme gives: FA-91-C4-FD-28-A2-D2-57-AA-D3-B4-35-B5-14-04-EE FF2A43841C84518A18795AB6E3C8A62E (NTLM)

napier gives: 12-B9-C5-4F-6F-E0-EC-80-AA-D3-B4-35-B5-14-04-EE 307E40814E7D4E103F6A69B04EA78F3D (NTLM)

<user>:<id>:<LM hash>:<NTLM hash>:<comment>:<homedir>:

```
Root@kali:~# cat pw
myuser:500:12B9C54F6FE0EC80AAD3B435B51404EE:307E40814E7D4E103F6A69B04EA78F3D:::
Root@kali:~# john  pw
Loaded 1 password hash (LM DES [128/128 BS SSE2])
NAPIER           (napier)
guesses: 1  time: 0:00:00:00 100% (1)  c/s: 4850  trying: NAPIER - N4PI3R
Use the "--show" option to display all of the cracked passwords reliably
```

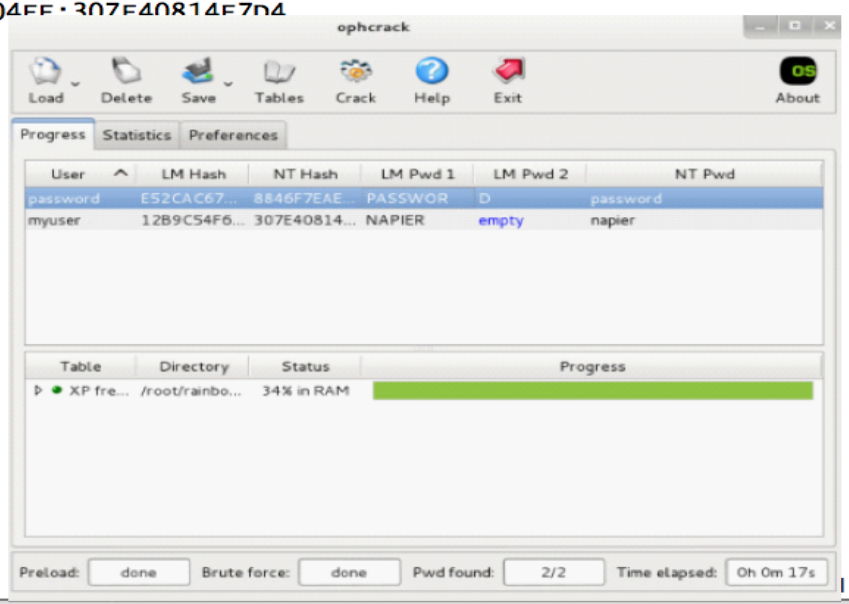
Author: Prof Bill Buchanan

SAM

Registry:
HKEY_LOCAL_MACHINE\SAM

```
Root@kali:~# cat pw
myuser:500:12B9C54F6FE0EC80AAD3B435B51404EE:307E40814E7D4E103F6A69B04EA78F3D:::
Root@kali:~# john  pw
Loaded 1 password hash (LM DES [128/128 BS SSE2])
NAPIER           (napier)
guesses: 1  time: 0:00:00:00 100% (1)  c/s: 4850  trying: NAPIER - N4PI3R
Use the "--show" option to display all of the cracked passwords reliably
```

```
<user>:<id>:<LM hash>:<NTLM hash>:<comment>:<homedir>:
password:500:E52CAC67419A9A224A3B108F3FA6CB6D:8846F7EAEE8
FB117AD06BDD830B7586C:$
myuser:500:12B9C54F6FE0EC80AAD3B435B51404EE:307E40814E7D4
E103F6A69B04EA78F3D:::
```



Hash Crackers/Bit Coin Miners

Fast Hash One

- 1.536TH/s – Cost 3-5,000 dollars.

25 GPU Hash Cracker

- An eight character NTLM password cracked in 5.5 hours. 14 character LM hash cracked in six minutes. 350 billion hashes per second.

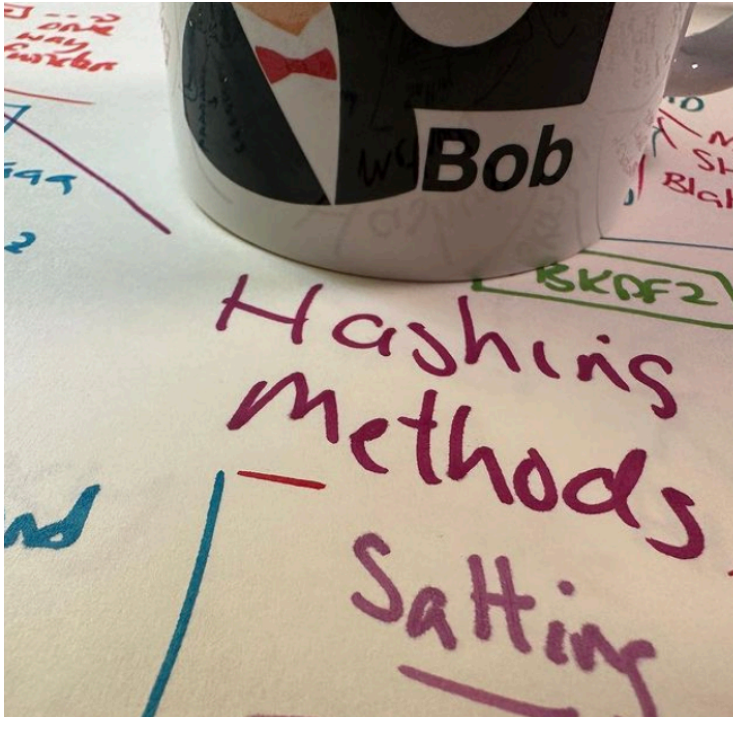
Estimated Password Recovery Times — 1x Terahash Brutalis, 44x Terahash Inmanis (448x Nvidia RTX 2080)

Full US keyboard mask attack with Terahash Hashstack

	Speed	Length 4	Length 5	Length 6	Length 7	Length 8	Length 9	Length 10	Length 11	Length 12	Length 13
NTLM	31.82 TH/s	Instant	Instant	Instant	Instant	3 mins 29 secs	5 hrs 30 mins	3 wks 0 day	5 yrs 7 mos	538 yrs 1 mo	51.2 mil
MD5	17.77 TH/s	Instant	Instant	Instant	Instant	6 mins 14 secs	9 hrs 50 mins	1 mo 1 wk	10 yrs 1 mo	963 yrs 4 mos	91.6 mil
NetNTLMv1 / NetNTLMv1+ESS	16.82 TH/s	Instant	Instant	Instant	Instant	6 mins 35 secs	10 hrs 24 mins	1 mo 1 wk	10 yrs 8 mos	1 mil	96.8 mil
LM	15.81 TH/s	Instant	Instant	Instant	Instant						
SHA1	5.89 TH/s	Instant	Instant	Instant	Instant	18 mins 47 secs	1 day 5 hrs	3 mos 3 wks	30 yrs 7 mos	2.9 mil	276.3 mil
SHA2-256	2.42 TH/s	Instant	Instant	Instant	Instant	45 mins 39 secs	3 days 0 hr	9 mos 1 wk	74 yrs 4 mos	7.1 mil	671.9 mil
NetNTLMv2	1.22 TH/s	Instant	Instant	Instant	Instant	1 hr 30 mins	5 days 23 hrs	1 yr 6 mos	147 yrs 10 mos	14.1 mil	1335.5 mil
SHA2-512	801.9 GH/s	Instant	Instant	Instant	1 min 28 secs	2 hrs 17 mins	1 wk 2 days	2 yrs 4 mos	224 yrs 9 mos	21.4 mil	2029.7 mil
descrypt, DES (Unix), Traditional DES	647.59 GH/s	Instant	Instant	Instant	1 min 48 secs	2 hrs 50 mins	1 wk 4 days	2 yrs 11 mos	278 yrs 3 mos	26.5 mil	2513.3 mil
Kerberos 5, etype 23, TGS-REP	206.97 GH/s	Instant	Instant	Instant	5 mins 38 secs	8 hrs 54 mins	1 mo 0 wk	9 yrs 2 mos	870 yrs 10 mos	82.8 mil	7864 mil
Kerberos 5, etype 23, AS-REQ Pre-Auth	206.78 GH/s	Instant	Instant	Instant	5 mins 38 secs	8 hrs 54 mins	1 mo 0 wk	9 yrs 2 mos	871 yrs 8 mos	82.9 mil	7871.2 mil
md5crypt, MD5 (Unix), Cisco-IOS 1 (MD5)	7.61 GH/s	Instant	Instant	1 min 37 secs	2 hrs 33 mins	1 wk 3 days	2 yrs 7 mos	249 yrs 5 mos	23.7 mil	2252.6 mil	213995.1 mil
LastPass + LastPass sniffed	1.78 GH/s	Instant	Instant	6 mins 52 secs	10 hrs 52 mins	1 mo 1 wk	11 yrs 2 mos	1.1 mil	101.1 mil	9600.8 mil	912079.6 mil
macOS v10.8+ (PBKDF2-SHA512)	335.09 MH/s	Instant	Instant	36 mins 34 secs	2 days 9 hrs	7 mos 2 wks	59 yrs 7 mos	5.7 mil	538.2 mil	51127.7 mil	4857134 mil
WPA-EAPOL-PBKDF2	277.23 MH/s					9 mos 0 wk	72 yrs 0 mo	6.8 mil	650.5 mil	61799.3 mil	5870931.8 mil
TrueCrypt RIPEMD160 + XTS 512 bit	211.78 MH/s	Instant	Instant	57 mins 52 secs	3 days 19 hrs	11 mos 3 wks	94 yrs 3 mos	9 mil	851.6 mil	80899.5 mil	7685455.6 mil
7-Zip	181.51 MH/s	Instant	Instant	1 hr 7 mins	4 days 10 hrs	1 yr 1 mo	110 yrs 0 mo	10.5 mil	993.6 mil	94389.2 mil	8966975.1 mil
sha512crypt 6, SHA512 (Unix)	119.46 MH/s	Instant	1 min 5 secs	1 hr 42 mins	6 days 18 hrs	1 yr 9 mos	167 yrs 2 mos	15.9 mil	1509.7 mil	143419.6 mil	13624861.4 mil
DPAPI masterkey file v1	47.23 MH/s	Instant	2 mins 44 secs	4 hrs 19 mins	2 wks 3 days	4 yrs 5 mos	422 yrs 10 mos	40.2 mil	3818.1 mil	362723.1 mil	34458696.1 mil
RAR5	28.15 MH/s	Instant	4 mins 35 secs	7 hrs 15 mins	4 wks 0 day	7 yrs 5 mos	709 yrs 7 mos	67.4 mil	6407.6 mil	608720.6 mil	57828453.9 mil
DPAPI masterkey file v2	27.82 MH/s	Instant	4 mins 39 secs	7 hrs 20 mins	4 wks 1 day	7 yrs 6 mos	717 yrs 10 mos	68.2 mil	6482.1 mil	615797.6 mil	58500769.5 mil
RAR3-hp	20.84 MH/s	Instant	6 mins 12 secs	9 hrs 47 mins	1 mo 1 wk	10 yrs 1 mo	958 yrs 2 mos	91.1 mil	8652.3 mil	821972.3 mil	78087367.8 mil
KeePass 1 (AES/Twofish) and KeePass 2 (AES)	17.8 MH/s	Instant	7 mins 15 secs	11 hrs 28 mins	1 mo 2 wks	11 yrs 9 mos	1.1 mil	106.7 mil	10131.9 mil	962529.5 mil	91440305.8 mil
bcrypt $2*$, Blowfish (Unix)	11.37 MH/s	Instant	11 mins 21 secs	17 hrs 57 mins	2 mos 1 wk	18 yrs 5 mos	1.8 mil	167 mil	15860.3 mil	1506727.9 mil	143139150.9 mil
Bitcoin/Litecoin wallet.dat	3.55 MH/s	Instant	36 mins 18 secs	2 days 9 hrs	7 mos 2 wks	59 yrs 1 mo	5.6 mil	534.1 mil	50743.7 mil	4820655.6 mil	457962282.7 mil

Unit 3: Hashing

Hashing Types.
Hashing Methods.
Salting.
Collisions.
LM and NTLM Hashes (Windows).
Hash Benchmarks.
Message Authentication Codes (MACs).
Key Derivation Function (KDF)
OTP/HOTP.
Hashcat.



Prof Bill Buchanan OBE
https://asecuritysite.com/hash

Benchmark

```
Ulta fast:
Murmur:          545,716 hashes per second

Fast:
SHA-1:           134,412
SHA-256:         126,323
MD5:             125,741
SHA-512:          76,005
SHA-3 (224-bit): 72,089

Medium speed:
LDAP (SHA1):     13,718
MS DCC:           9,582
NT Hash:          7,782
MySQL:            7,724
Postgres (MD5):   7,284

Slow:
PBKDF2 (SHA-256): 5,026
Cisco PIX:        4,402
MS SQL 2000:      4,225
LDAP (MD5):       4,180
Cisco Type 7:     3,775
PBKDF2 (SHA1):    2,348

Ulta-slow:
LM Hash:            733
APR1:               234
Bcrypt:             103
DES:                 88
Oracle 10:           48
```

Hashes “The quick brown fox jumps over the lazy dog:

SHA-1: 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
SHA-256: d7a8fbb307d7809469ca9abcb0082e4f8d5651e46d3cdb762d02d0bf37c9e592
SHA-512: 07e547d9586f6a73f73fbac0435ed76951218fb7d0c8d788a309d7
85436bbb642e93a252a954f23912547d1e8a3b5ed6e1bfd7097821233fa0538f3db854fee6

MD-5: 9e107d9d372bb6826bd81d3542a419d6
DES: ZDeS94Lcq/6zg
Bcrypt: $2a$**05**$2czCv5GYgkx3aobmEyewB.ejV2hePMdbvTdCyNaSzWtlGPPjB2xx6
APR1: $apr1$ZDzPE45C$3PvRanPycmNc6c2G9wT9b/
PBKDF2 (SHA1): $pbkdf2$5$WkR6UEU0NUM$0RB2bimWrMY.EPYibpaBT2q3HFg
PBKDF2 (SHA-256): $pbkdf2-sha256$5$WkR6UEU0NUM$yrJz2oJix7uBJZwZ/50vWUgdE
l/i0ffqeU4obqC0pk4

LM Hash: a7b07f9948d8cc7f97c4b0b30cae500f
NT Hash: 4e6a076ae1b04a815fa6332f69e2e231
MS DCC: efa9778bbc94a7360f664eb7d7144725
LDAP (MD5): {MD5}9e107d9d372bb6826bd81d3542a419d6
LDAP (SHA1): {SHA}2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
MS SQL 2000: 0x0100BF77CE595DCD1FC87A37B3DEBC27A8C97355CB96B8BAB
63E602662BA5D5D33B913E422499BE72FF3D9BB65DE

MySQL: *A4E4D26FD0C6455E23E2187C3AABE844332AA1B3
Oracle 10: 4CDA2299FCAD0499
Postgres (MD5): md5d44c15daa11770f25c5350f7e5408dd1
Cisco PIX: kGyKN5CqdFQ1qJUs
Cisco Type 7: 15260309443B3E2D2B3875200108010D41505640135E1B0E080
519574156401540035E460B594D1D53020B5C

Benchmark

BLAKE 3

Hash function	MiB/sec	cycl./hash	cycl./map	Quality problems
blake3_c	1285.91	340.01	552.63	no 32bit portability
blake2b-256_64	356.97	1222.76	1435.0	
blake2b-224	356.59	1228.5	1425.87	
blake2b-160	356.08	1236.84	1458.15	
blake2b-256	355.97	1232.22	1443.31	Sparse high 32-bit
md5-128	353.76	638.29	803.39	
md5_32a	353.64	629.85	799.56	Sparse high 32-bit
sha1_32a	353.03	1385.8	1759.9	Cyclic low32, 36.6% distrib
rmd128	334.36	659.03	838.32	Bad seeds
blake2s-128	295.3	698.09	1059.2	
pearsonhash64	287.95	174.11	196.50	Avalanche, Seed, SSSE3 only. broken MSVC
pearsonhash128	287.95	171.72	194.61	Avalanche, Seed, SSSE3 only. broken MSVC
pearsonhash256	264.51	184.87	218.79	Avalanche, Seed, SSSE3 only. broken MSVC
blake2s-256	215.28	1014.88	1230.38	
blake2s-160	215.01	1026.74	1239.54	
blake2s-256_64	211.52	1044.22	1228.43	
blake2s-224	207.06	1063.86	1236.50	
rmd160	202.16	1045.79	1287.74	Bad seeds, Cyclic hi32
sha2-256_64	148.01	1376.34	1624.71	Bad seeds, Moment Chi2 7
sha2-256	147.8	1374.9	1606.06	Bad seeds, Moment Chi2 4
sha2-224_64	147.6	1360.1	1620.93	Bad seeds, Cyclic low32
sha2-224	147.13	1354.81	1589.92	Bad seeds, Comb low32
asconhashv12	144.98	885.02	1324.23	
sha3-256	100.58	3877.18	4159.79	PerlinNoise
sha3-256_64	100.57	3909	4174.63	PerlinNoise
asconhashv12_64	86.73	684.02	606.93	
floppsyhash	35.72	1868.92	1411.07	
tifuhash_64	35.6	1679.52	1212.75	Cyclic low32

BLAKE3

one function, fast everywhere

Jack O'Connor (@oconnor663)
Jean-Philippe Aumasson (@veorq)
Samuel Neves (@sevenps)
Zooko Wilcox-O'Hearn (@zooko)

https://blake3.io

We present BLAKE3, an evolution of the BLAKE2 cryptographic hash that is both faster and also more consistently fast across different platforms and input sizes. BLAKE3 supports an unbounded degree of parallelism, using a tree structure that scales up to any number of SIMD lanes and CPU cores. On Intel Cascade Lake-SP, peak single-threaded throughput is 4× that of BLAKE2b, 8× that of SHA-512, and 12× that of SHA-256, and it can scale further using multiple threads. BLAKE3 is also efficient on smaller architectures: throughput on a 32-bit ARM1176 core is 1.3× that of SHA-256 and 3× that of BLAKE2b and SHA-512. Unlike BLAKE2 and SHA-2, with different variants better suited for different platforms, BLAKE3 is a single algorithm with no variants. It provides a simplified API supporting all the use cases of BLAKE2, including keying and extendable output. The tree structure also supports new use cases, such as verified streaming and incremental updates.

View

Slowing a hash

```
Ulta fast:
Murmur:          545,716 hashes per second

Fast:
SHA-1:           134,412
SHA-256:         126,323
MD5:             125,741
SHA-512:          76,005
SHA-3 (224-bit): 72,089

Medium speed:
LDAP (SHA1):     13,718
MS DCC:           9,582
NT Hash:          7,782
MySQL:            7,724
Postgres (MD5):   7,284

Slow:
PBKDF2 (SHA-256): 5,026
Cisco PIX:        4,402
MS SQL 2000:      4,225
LDAP (MD5):       4,180
Cisco Type 7:     3,775
PBKDF2 (SHA1):    2,348

Ulta-slow:
LM Hash:            733
APR1:               234
Bcrypt:             103
DES:                 88
Oracle 10:           48
```

To slow the hash, we can use rounds, where we continually hash for a number of rounds. In the following we use 5,000 rounds:

===Splunk hashed password===
User: Fred
Password: qwerty123
Salt: Uk8SVGLsBuSmD75R
===Hashed password **Fred:**
6Uk8SVGLsBuSmD75R$Lhp5yjwRUAM.LbH5IIthZ1u0bAUdJwBvvccBshAvpFPiRn62EYeiKOaP8xh97aV4UaNfVykRZhUy/3ZOZd1oc.

root@kali:~# hashcat -b -m 1800
Initializing hashcat v0.49 with 1 threads and 32mb segment-size…
Device……..: Intel(R) Core(TM) i7–8850H CPU 2.60GHz
Instruction set..: x86_64
Number of threads: 1
Hash type: sha512crypt, SHA512(Unix)
Speed/sec**: 454 words**

root@kali:~# hashcat -b -m 0
Initializing hashcat v0.49 with 1 threads and 32mb segment-size…
Device: Intel(R) Core(TM) i7–8850H CPU 2.60GHz
Instruction set..: x86_64
Number of threads: 1
Hash type: MD5
Speed/sec: **17.33M words**

SHA512-crypt

Non-cryptographic hashing

Hash function	MiB/sec	cycl./hash	cycl./map	Quality problems
o1hash	12439661.09	16.77	166.13	insecure, no seed, zeros, fails all tests
crc32_hw1	23208.73	46.74	179.70	insecure, 100% bias, collisions, distrib, BIC, machine-specific (x86 SSE4.2)
t1ha0_aes_noavx	22785.26	38.71	180.61	LongNeighbors, machine-specific (x86 AES-NI)
t1ha0_aes_avx1	22714.85	48.12	226.52	LongNeighbors, machine-specific (x64 AVX.txt)
t1ha0_aes_avx2	22345.33	44.38	556.47	LongNeighbors, machine-specific (x64 AVX2)
falkhash	20202.42	173.63	321.52	Sparse, LongNeighbors, machine-specific (x64 AES-NI)
MeowHash64low	17378.06	85.48	237.60	Sparse, invertible, machine-specific (x64 AES-NI)
MeowHash32low	17374.64	85.48	258.53	Sparse, invertible, machine-specific (x64 AES-NI)
MeowHash	17371.91	85.48	247.96	Sparse, invertible, machine-specific (x64 AES-NI)
FarmHash32	17112.05	47.7	214.71	machine-specific (x64 SSE4/AVX)
xxh3	16538.52	32.81	184.86	DiffDist bit 7 w. 36 bits, BIC
xxh3low	16462.36	32.77	199.79	
farmhash32_c	16299.81	47.79	219.19	machine-specific (x64 SSE4/AVX)
xxh128low	15174.85	33.79	187.05	
xxh128	15174.14	40.46	195.65	
farsh32	14053.09	74.29	245.33	insecure: AppendedZeroes, collisions+bias, MomentChi2, LongNeighbors
metrohash64crc_2	14034.84	48.94	162.54	UB, Cyclic 8/8 byte, DiffDist, BIC, machine-specific (SSE4.2/NEON)
metrohash64crc_1	14000.5	49.08	150.54	UB, Cyclic 8/8 byte, DiffDist, BIC, MomentChi2, machine-specific (SSE4.2/NEON)
metrohash128crc_1	13948.67	65.2	168.08	UB, machine-specific (SSE4.2/NEON)
metrohash128crc_2	13920.19	65.12	176.70	UB, machine-specific (SSE4.2/NEON)
halftime_hash128	13478.23	97.79	252.14	
wyhash32low	12911.09	29.59	205.43	2 bad seeds
wyhash	12879	30.35	196.77	2^33 bad seeds
CityCrc128	12343.43	74.5	209.75	
fletcher2	12011.15	25.29	298.60	bad seed 0, UB, fails all tests
fletcher4	11928.55	25.27	293.49	bad seed 0, UB, fails all tests
halftime_hash256	11620.28	98.44	252.60	
fibonacci	11339.87	26.33	705.64	UB, zeros, fails all tests
ahash64	9862.62	27.32	181.68	rust
Spooky128	9751.14	63.84	192.47	UB
Spooky64	9747.47	62.2	191.71	UB

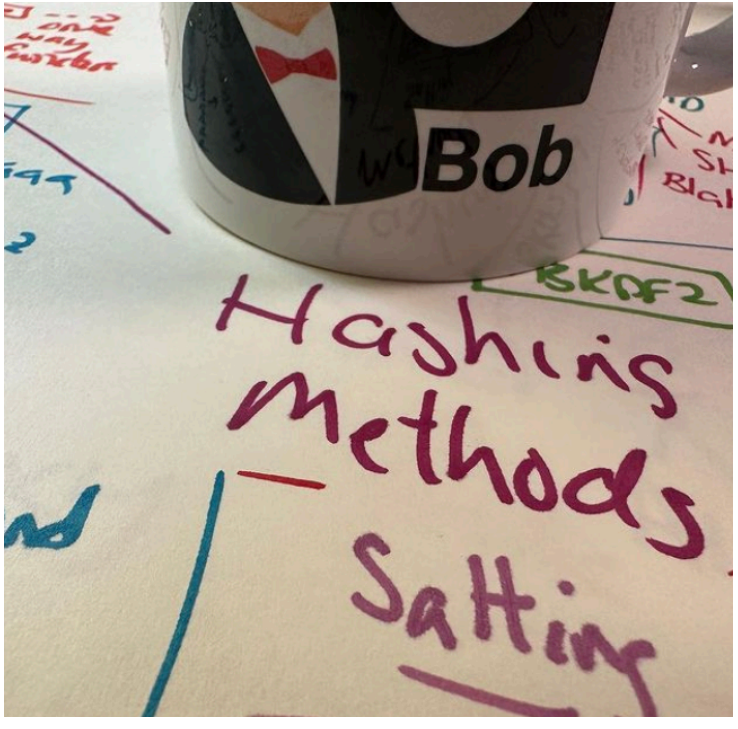
Used for hash tables. Google recommend the following for 64-bit hashes without quality problems:

- xxh3low
- wyhash
- ahash64
- t1ha2_atonce
- FarmHash
- halftime_hash128
- Spooky32
- pengyhash
- nmhash32
- mx3
- MUM/mir
- fasthash32

Non-crypto

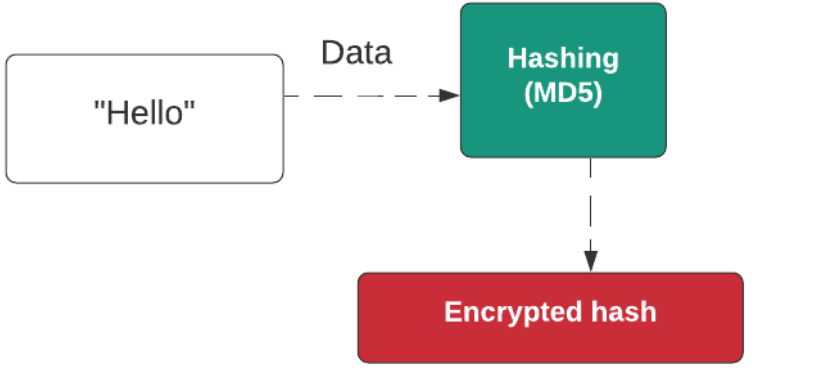
Unit 3: Hashing

Hashing Types.
Hashing Methods.
Salting.
Collisions.
LM and NTLM Hashes (Windows).
Hash Benchmarks.
Message Authentication Codes (MACs).
Key Derivation Function (KDF)
OTP/HOTP.
Hashcat.



Prof Bill Buchanan OBE
https://asecuritysite.com/hash

Hashing



AF5597C29467A96523A70787C319F4DB

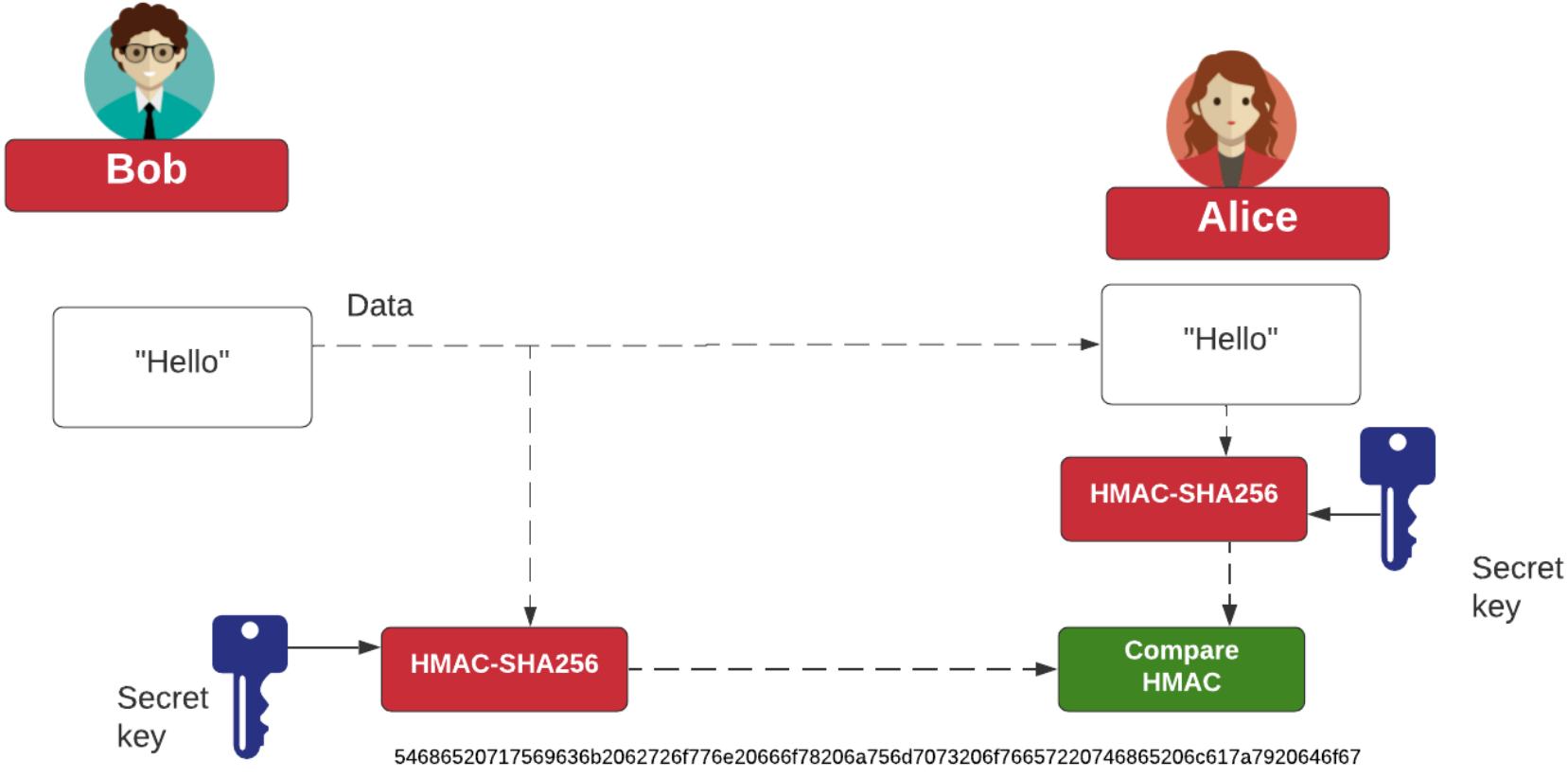
Hashed password

Bob: AF5597C29467A96523A70787C319F4DB

Hashed password with salt (hellozj8n)

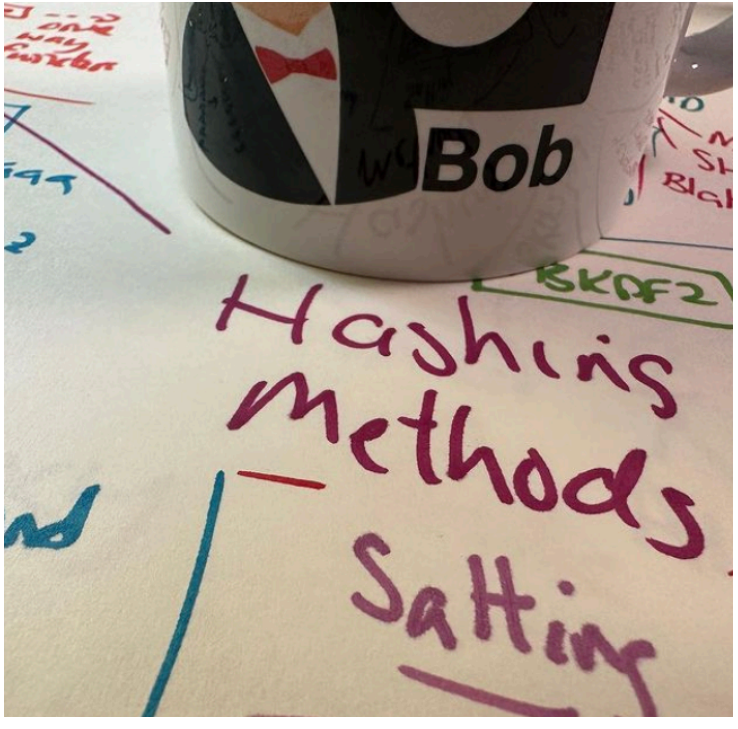
Bob: zj8n:51A7C663A3BDCD06D6CE21E2BCB2AD5A

HMAC


Bob
Alice
Data
"Hello"
"Hello"
HMAC-SHA256
Secret key
HMAC-SHA256
Compare HMAC
Secret key
54686520717569636b2062726f776e20666f78206a756d7073206f766572207468652065206c617a792046f67

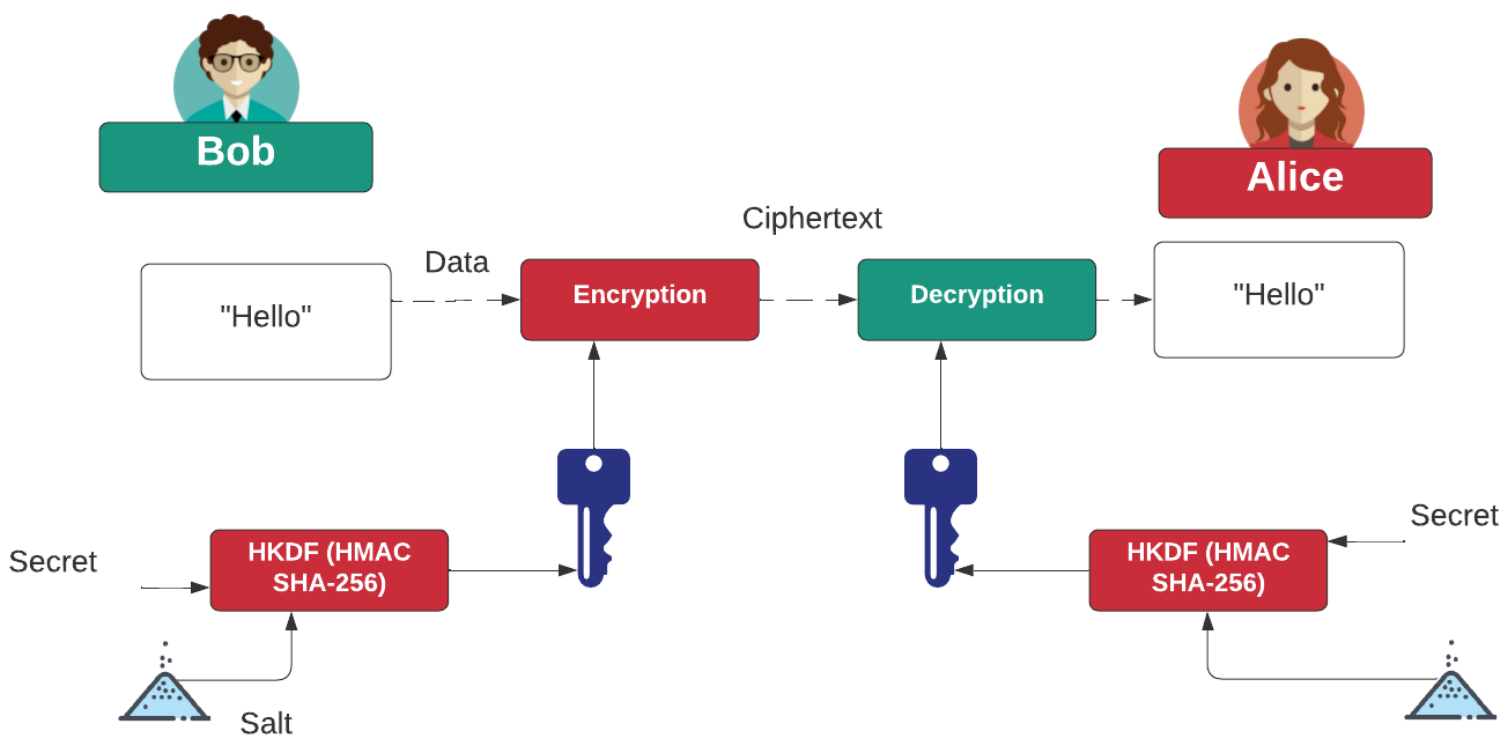
Unit 3: Hashing

Hashing Types.
Hashing Methods.
Salting.
Collisions.
LM and NTLM Hashes (Windows).
Hash Benchmarks.
Message Authentication Codes (MACs).
Key Derivation Functions (KDF)
OTP/HOTP.
Hashcat.



Prof Bill Buchanan OBE
https://asecuritysite.com/hash

HKDF

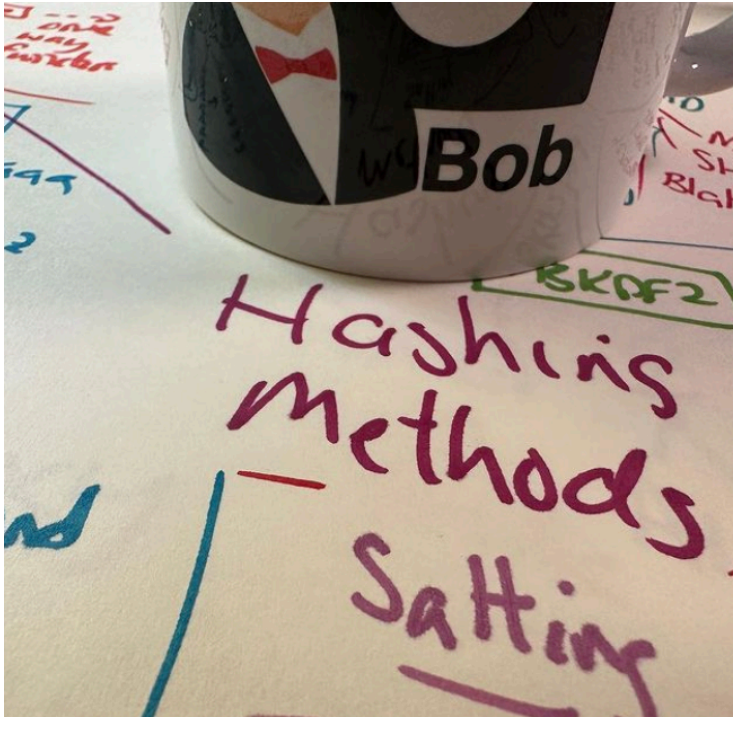



```
Hashing type:   SHA-256
Message:         hello
  Hex:           68656c6c6f
Salt:            8e94ef805b93e683ff18
Info:
====================
PRK: 1e133888e9fed8f9ceb210f88af26fa8f62f4190dd230f6317bf9f61ee07a690
OKM: 13485067e21af17c0900f70d885f0259

Key (Hex):       13485067e21af17c0900f70d885f0259
Key (Base-64):   E0hQZ+Ia8XwJAPcNiF8CWQ==
```

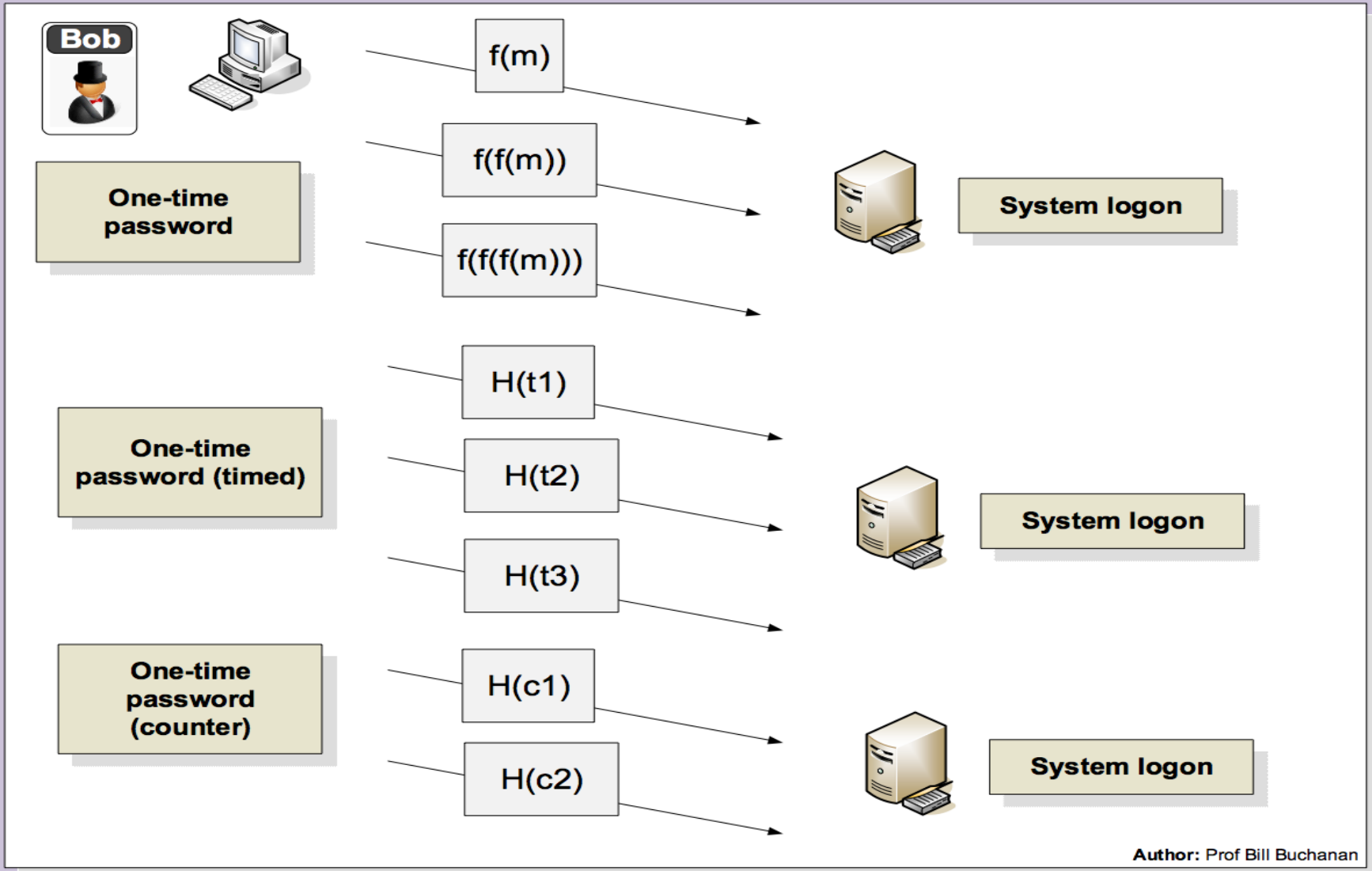
HKDF

Unit 3: Hashing

Hashing Types.
Hashing Methods.
Salting.
Collisions.
LM and NTLM Hashes (Windows).
Hash Benchmarks.
Message Authentication Codes (MACs).
OTP/HOTP.
Hashcat.

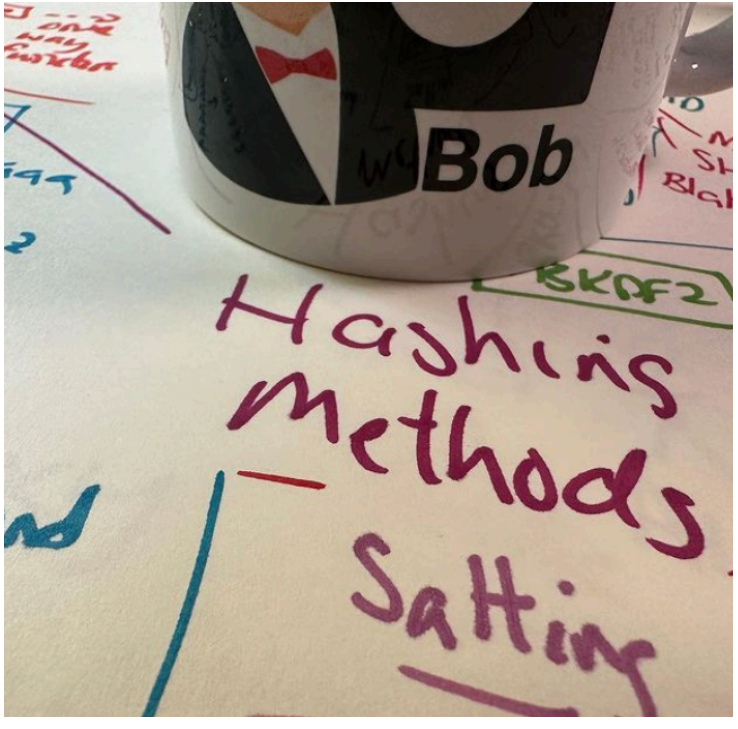


Prof Bill Buchanan OBE
https://asecuritysite.com/hash


Bob
One-time password
f(m)
f(f(m))
f(f(f(m)))
System logon
One-time password (timed)
H(t1)
H(t2)
H(t3)
System logon
One-time password (counter)
H(c1)
H(c2)
System logon
Author: Prof Bill Buchanan

Unit 3: Hashing

Hashing Types.
Hashing Methods.
Salting.
Collisions.
LM and NTLM Hashes (Windows).
Hash Benchmarks.
Message Authentication Codes (MACs).
Key Derivation Function (KDF)
OTP/HOTP.
Hashcat



Prof Bill Buchanan OBE
https://asecuritysite.com/hash

Hashing

Hashcat

Prof Bill Buchanan OBE

https://asecuritysite.com/hash

Hashcat

- **Rule-based Attacks.** Define a rule of the password.
- **Dictionary Attacks.** Search through a dictionary of common passwords.
- **Brute-Force Attacks.** Search through

Attack modes: Brute-force, Combinator, Dictionary, Mask, Rule-based, Toggle-case and Hybrid.

Hashcat

- hashcat [options] hashfile [mask|wordfiles|directories]

- hashcat -m 0 myhash myfile.txt -o output.txt

```
* General:
 -m,  --hash-type=NUM            Hash-type, see references below
 -a,  --attack-mode=NUM          Attack-mode, see references below
* Benchmark:
 -b,  --benchmark                Run benchmark
* Files:
 -o,  --outfile=FILE             Define outfile for recovered hash
      --outfile-format=NUM       Define outfile-format for recovered hash, see references below
--show                           Show cracked passwords only (see --username)
* Resources:
 -c,  --segment-size=NUM         Size in MB to cache from the wordfile
 -n,  --threads=NUM              Number of threads
* Rules:
 -r,  --rules-file=FILE          Rules-file use: -r 1.rule
 -g,  --generate-rules=NUM       Generate NUM random rules
      --generate-rules-func-min=NUM Force NUM functions per random rule min
      --generate-rules-func-max=NUM Force NUM functions per random rule max
* Toggle-Case attack-mode specific:
      --toggle-min=NUM           Number of alphas in dictionary minimum
      --toggle-max=NUM           Number of alphas in dictionary maximum

* Mask-attack attack-mode specific:
      --pw-min=NUM               Password-length minimum
      --pw-max=NUM               Password-length maximum

* Permutation attack-mode specific:
      --perm-min=NUM             Filter words shorter than NUM
      --perm-max=NUM             Filter words larger than NUM
* Table-Lookup attack-mode specific:
 -t,  --table-file=FILE          Table file
      --table-min=NUM            Number of chars in dictionary minimum
      --table-max=NUM            Number of chars in dictionary maximum
```

Hashcat

- hashcat [options] hashfile [mask|wordfiles|directories]
- hashcat -m 0 myhash myfile.txt -o output.txt

```
-a,  --attack-mode=NUM

Attack Modes:
0 = Straight
1 = Combination
2 = Toggle-Case
3 = Brute-force
4 = Permutation
5 = Table-Lookup
6 = Prince
```

Hashcat

- hashcat [options] hashfile [mask|wordfiles|directories]
- hashcat -m 0 myhash myfile.txt -o output.txt

```
-a,  --attack-mode=NUM

-m hash-type
* Hash types:

    0 = MD5
   10 = md5($pass.$salt)
   20 = md5($salt.$pass)
   30 = md5(unicode($pass).$salt)
   40 = md5($salt.unicode($pass))
   50 = HMAC-MD5 (key = $pass)
   60 = HMAC-MD5 (key = $salt)
  100 = SHA1
  110 = sha1($pass.$salt)
  120 = sha1($salt.$pass)
  130 = sha1(unicode($pass).$salt)
  140 = sha1($salt.unicode($pass))
  150 = HMAC-SHA1 (key = $pass)
  160 = HMAC-SHA1 (key = $salt)
  200 = MySQL323
  300 = MySQL4.1/MySQL5
  400 = phpass, MD5(Wordpress), MD5(phpBB3), MD5(Joomla)
  500 = md5crypt, MD5(Unix), FreeBSD MD5, Cisco-IOS MD5
  900 = MD4
 1000 = NTLM
1400 = SHA256
1410 = sha256($pass.$salt)
5500 = NetNTLMv1-VANILLA / NetNTLMv1-ESS
 5600 = NetNTLMv2
 5700 = Cisco-IOS SHA256
8900 = scrypt
 9200 = Cisco $8$
10000 = Django (PBKDF2-SHA256)
```

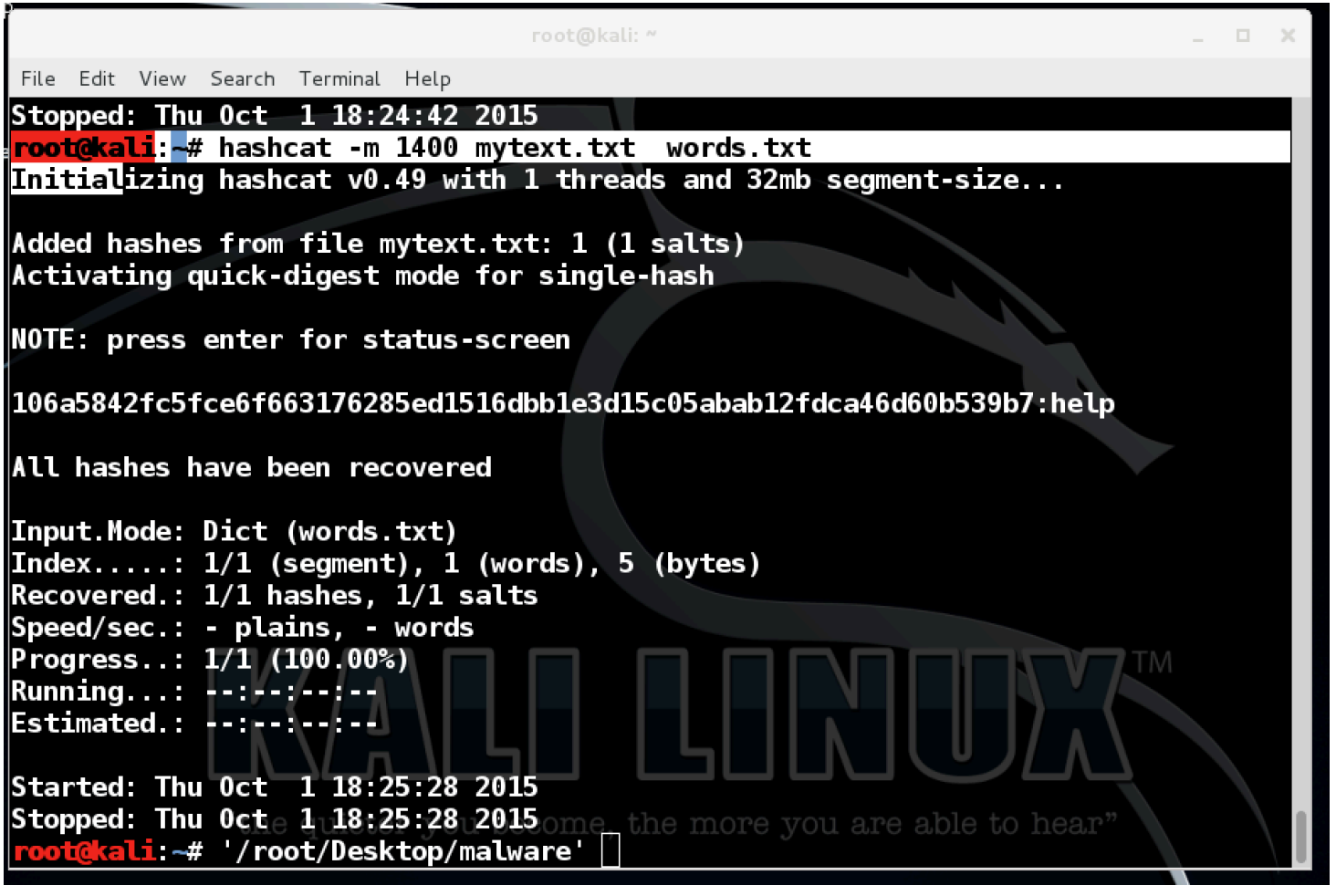
Hashcat - Outputs

- hashcat [options] hashfile [mask|wordfiles|directories]
- hashcat -m 0 myhash myfile.txt -o output.txt

Output format

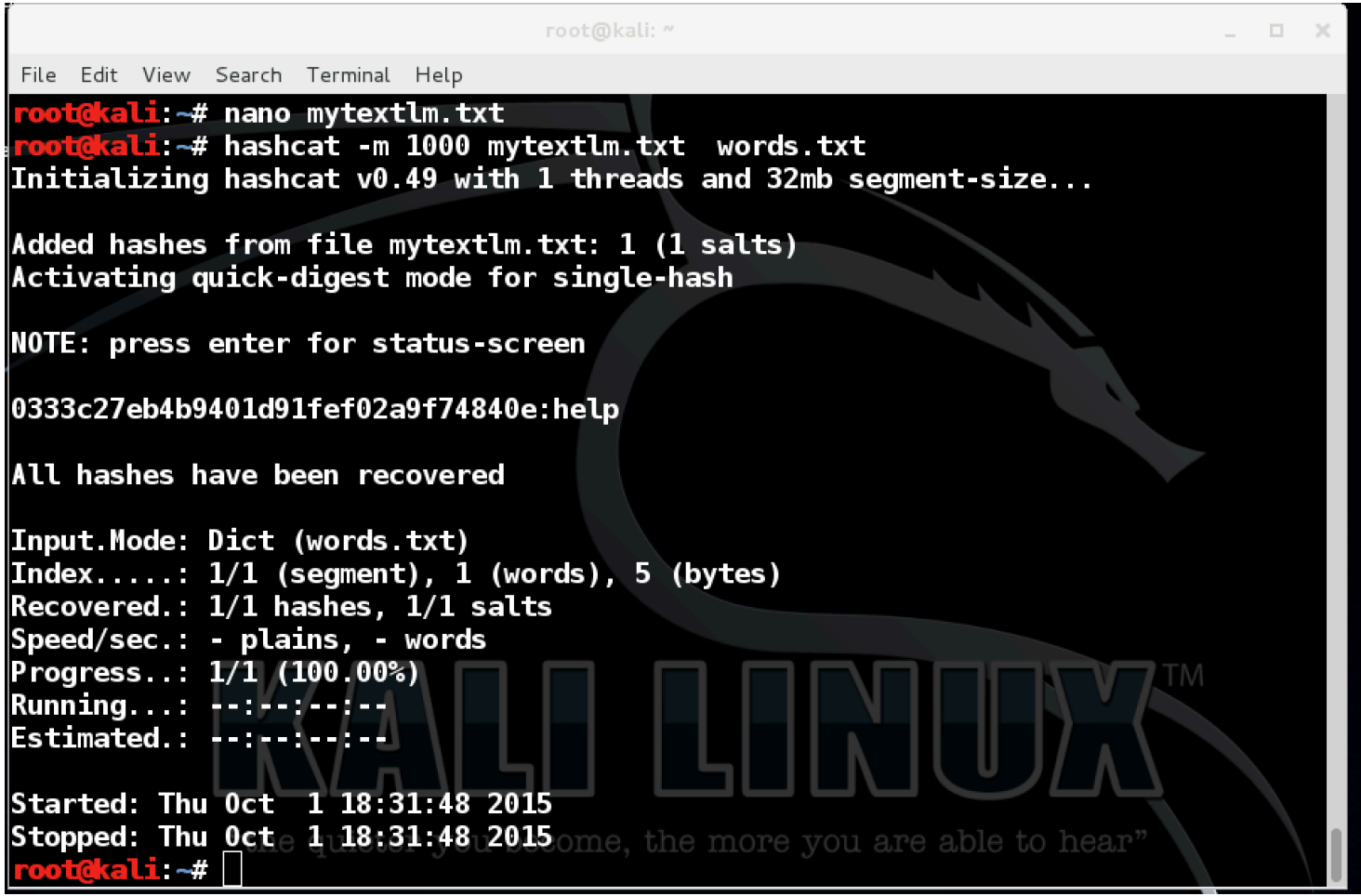
1 = hash[:salt]
2 = plain
3 = hash[:salt]:plain
4 = hex_plain
5 = hash[:salt]:hex_plain
6 = plain:hex_plain
7 = hash[:salt]:plain:hex_plain
8 = crackpos
9 = hash[:salt]:crackpos
10 = plain:crackpos
11 = hash[:salt]:plain:crackpos
12 = hex_plain:crackpos
13 = hash[:salt]:hex_plain:crackpos
14 = plain:hex_plain:crackpos
15 = hash[:salt]:plain:hex_plain:crackpos

Hashcat



hashcat -m 1400 mytext.txt words.txt

Hashcat



hashcat -m 1000 mytextnt.txt words.txt

Combinational Attack (-a 1)

```
Hashcat -m 0 -a 1 hash.txt dict1.txt dict2.txt
```

dict1.txt

```
jam
jar
jimmy
jamie
```

dict2.txt

```
car
cat
call
```

```
jamcar
jarcar
Jimmycar
Jamiecar
jamcat
jarcat
jimmycat
jamiecat
```

Hashing

Brute Force (-a 3)

Prof Bill Buchanan OBE

https://asecuritysite.com/hash

Hashcat – Brute Force

- hashcat [options] hashfile [mask|wordfiles|directories]
- hashcat -m 0 myhash myfile.txt -o output.txt

```
Custom charsets:
-1,  --custom-charset1=CS       User-defined charsets
-2,  --custom-charset2=CS       Example:
-3,  --custom-charset3=CS       --custom-charset1=?dabcdef : sets
charset ?1 to 0123456789abcdef
-4,  --custom-charset4=CS       -2 mycharset.hcchr : sets charset ?2
to chars contained in file

Character sets

 ?l = abcdefghijklmnopqrstuvwxyz
  ?u = ABCDEFGHIJKLMNOPQRSTUVWXYZ
  ?d = 0123456789
  ?s =  !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
  ?a = ?l?u?d?s
  ?b = 0x00 - 0xff
```

Hashcat – Brute Force

?l = abcdefghijklmnopqrstuvwxyz
?u = ABCDEFGHIJKLMNOPQRSTUVWXYZ
?d = 0123456789
?s = !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
?a = ?l?u?d?s
?b = 0x00 - 0xff

- -1 abcdefghijklmnopqrstuvwxyz0123456789
- -1 abcdefghijklmnopqrstuvwxyz?d
- -1 ?l0123456789-1 ?l?d

Hashcat – Brute Force

hashcat -a 3 -m 1000 mytextlm.txt ?l?l?l?l?l?l?l?l?l?l

[keyspace: aaaaaaaa – zzzzzzzz]

```
Input.Mode: Mask (?l) [1]
Index.....: 0/1 (segment), 26 (words), 0 (bytes)
Recovered.: 0/1 hashes, 0/1 salts
Speed/sec.: - plains, - words
Progress..: 26/26 (100.00%)
Running...: --:--:--:--
Estimated.: --:--:--:--
Input.Mode: Mask (?l?l) [2]
Index.....: 0/1 (segment), 676 (words), 0 (bytes)
Recovered.: 0/1 hashes, 0/1 salts
Speed/sec.: - plains, - words
Progress..: 676/676 (100.00%)
Running...: --:--:--:--
Estimated.: --:--:--:--
Input.Mode: Mask (?l?l?l) [3]
Index.....: 0/1 (segment), 17576 (words), 0 (bytes)
Recovered.: 0/1 hashes, 0/1 salts
Speed/sec.: - plains, - words
Progress..: 17576/17576 (100.00%)
Running...: --:--:--:--
Estimated.: --:--:--:--
0333c27eb4b9401d91fef02a9f74840e:help
All hashes have been recovered
Input.Mode: Mask (?l?l?l?l) [4]
Index.....: 0/1 (segment), 456976 (words), 0 (bytes)
Recovered.: 1/1 hashes, 1/1 salts
Speed/sec.: - plains, - words
Progress..: 271188/456976 (59.34%)
Running...: --:--:--:--
Estimated.: --:--:--:--
```

Hashcat –
Brute Force

- **hashcat -a 3 -m 1000 mytextlm.txt hel?l** [keyspace: hela - help]

```
Input.Mode: Mask (h) [1]
Index.....: 0/1 (segment), 1 (words), 0 (bytes)
Recovered.: 0/1 hashes, 0/1 salts
Speed/sec.: - plains, - words
Progress..: 1/1 (100.00%)
Running...: --:--:--:--
Estimated.: --:--:--:--
Input.Mode: Mask (he) [2]
Index.....: 0/1 (segment), 1 (words), 0 (bytes)
Recovered.: 0/1 hashes, 0/1 salts
Speed/sec.: - plains, - words
Progress..: 1/1 (100.00%)
Running...: --:--:--:--
Estimated.: --:--:--:--
Input.Mode: Mask (hel) [3]
Index.....: 0/1 (segment), 1 (words), 0 (bytes)
Recovered.: 0/1 hashes, 0/1 salts
Speed/sec.: - plains, - words
Progress..: 1/1 (100.00%)
Running...: --:--:--:--
Estimated.: --:--:--:--
0333c27eb4b9401d91fef02a9f74840e:help
All hashes have been recovered
Input.Mode: Mask (hel?l) [4]
Index.....: 0/1 (segment), 26 (words), 0 (bytes)
Recovered.: 1/1 hashes, 1/1 salts
Speed/sec.: - plains, - words
Progress..: 16/26 (61.54%)
Running...: --:--:--:--
Estimated.: --:--:--:--
```

Hashcat – Brute Force

```
...

b1b735762130797915df96acf5bf09b8:hel9

All hashes have been recovered

Input.Mode: Mask (hel?d) [4]
Index.....: 0/1 (segment), 10 (words), 0 (bytes)
Recovered.: 1/1 hashes, 1/1 salts
Speed/sec.: - plains, - words
Progress..: 10/10 (100.00%)
Running...: --:--:--:--
Estimated.: --:--:--:--
```

hashcat -a 3 -m 1000 mytextlm.txt hel?d

[keyspace: hela - help]

Unit 3: Hashing

Rules and Lists

Prof Bill Buchanan OBE

http://asecuritysite.com/crypto02
http://asecuritysite.com/encryption

Hashcat – Password List

```
root@kali:~# wc -l /usr/share/wordlists/rockyou.txt
14344392 /usr/share/wordlists/rockyou.txt
```

123456	abc123
12345	nicole
123456789	daniel
password	babygirl
iloveyou	monkey
princess	lovely
1234567	jessica
rockyou	anthony
12345678	friends
	butterfly

```
root@kali:~# hashcat -m 0 mytextmd.txt /usr/share/wordlists/rockyou.txt

Initializing hashcat v0.49 with 1 threads and 32mb segment-size...

Added hashes from file mytextmd.txt: 1 (1 salts)
Activating quick-digest mode for single-hash

NOTE: press enter for status-screen

25f9e794323b453885f5181f1b624d0b:123456789

All hashes have been recovered

Input.Mode: Dict (/usr/share/wordlists/rockyou.txt)
Index.....: 1/5 (segment), 3627099 (words), 33550339 (bytes)
Recovered.: 1/1 hashes, 1/1 salts
Speed/sec.: - plains, - words
Progress..: 4/3627099 (0.00%)
```

Hashcat – Rules

Name	Function	Description	Example Rule	Input Word	Output Word
Nothing	:	Do nothing	:	p@ssW0rd	p@ssW0rd
Lowercase	l	Lowercase	l	p@ssW0rd	p@ssw0rd
Uppercase	u	Uppercase	u	p@ssW0rd	P@SSW0RD
Capitalize	c	Capitalize 1st, lower rest	c	p@ssW0rd	P@ssw0rd
Append	$X	Append character X to end	$1	p@ssW0rd	p@ssW0rd1
Prepend	^X	Prepend character X to front	^1	p@ssW0rd	1p@ssW0rd
Replace	sXY	Replace all instances of X with Y	ss$	p@ssW0rd	p@$$W0rd

```
root@kali:~# ls /usr/share/hashcat/rules
best64.rule                    specific.rule
combinator.rule                T0XlC-insert_00-99_1950-2050_toprules_0_F.rule
d3ad0ne.rule                   T0XlC-insert_space_and_special_0_F.rule
dive.rule                      T0XlC-insert_top_100_passwords_1_G.rule
generated.rule                 T0XlC.rule
Incisive-leetspeak.rule        T0XlCv1.rule
InsidePro-HashManager.rule     toggles1.rule
InsidePro-PasswordsPro.rule    toggles2.rule
leetspeak.rule                 toggles3.rule
Ninja-leetspeak.rule           toggles4.rule
oscommerce.rule                toggles5.rule
rockyou-30000.rule
```

Name	Function	Description	Example Rule	Input Word	Output Word
Nothing	:	Do nothing	:	p@ssW0rd	p@ssW0rd
Lowercase	l	Lowercase	l	p@ssW0rd	p@ssw0rd
Uppercase	u	Uppercase	u	p@ssW0rd	P@SSW0RD
Capitalize	c	Capitalize 1st, lower rest	c	p@ssW0rd	P@ssw0rd
Append	$X	Append character X to end	$1	p@ssW0rd	p@ssW0rd1
Prepend	^X	Prepend character X to front	^1	p@ssW0rd	1p@ssW0rd
Replace	sXY	Replace all instances of X with Y	ss$	p@ssW0rd	p@$$W0rd

edinburgh

Ed1nburgh123

Capitalise the first character.
Add a 1, 2 .. 0 at the end.
Add “123” at the end.
Substitute ‘a’ for ‘@’
Substitute ‘l’ for a ‘1’
Substitute ‘5’ for an ‘s’
Substitute ‘3’ for an ‘e’
Substitute ‘0’ for an ‘o’

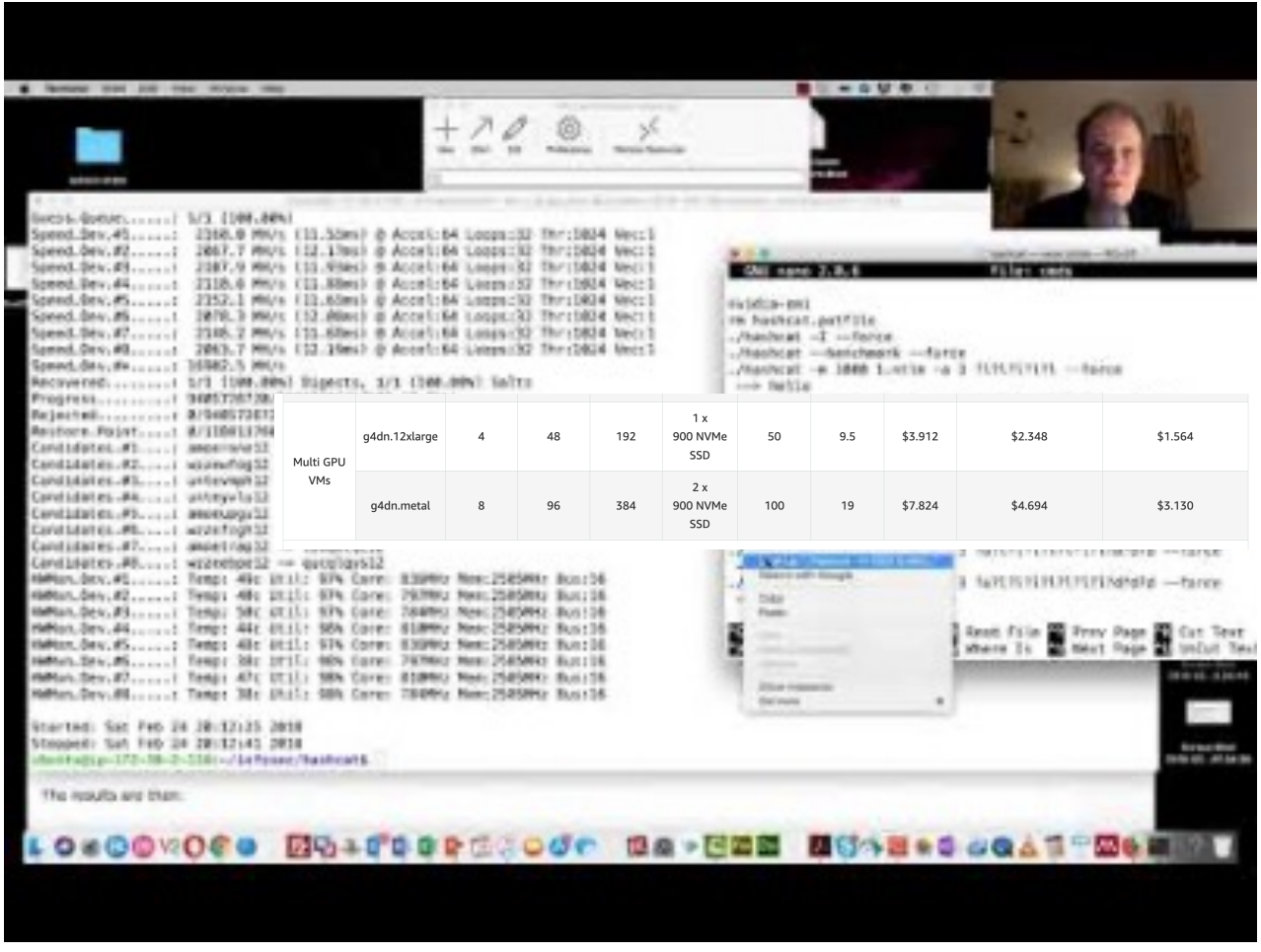
```
:
#Lowercase
l
#Uppercase
u
#Capitalise first character
c
#Add '1' to the end
$1
#Add '2' to the end
$2
#Add '3' to the end
$3
#Add '123' to the end
$1 $2 $3
#Substitute 'a' for '@'
sa@
#Substitute 'a' for '4'
sa4
#substitute 'l' for '1'
sl1
#Substitute 'a' for '@', 'e' for '3', 'l' for '1'
```

Hashcat – With Battlefield Hash

```
root@kali:~/hashes# hashcat -m 0 bfield.hash /usr/share/wordlists/rockyou.txt
Initializing hashcat v0.49 with 1 threads and 32mb segment-size...

Added hashes from file bfield.hash: 548686 (1 salts)

NOTE: press enter for status-screen

e10adc3949ba59abbe56e057f20f883e:123456
25f9e794323b453885f5181f1b624d0b:123456789
5f4dcc3b5aa765d61d8327deb882cf99:password
f25a2fc72690b780b2a14e140ef6a9e0:iloveyou
8afa847f50a716e64932d995c8e7435a:princess
fcea920f7412b5da7be0cf42b8c93759:1234567
25d55ad283aa400af464c76d713c07ad:12345678
```

Hashcat – With 8xGPU

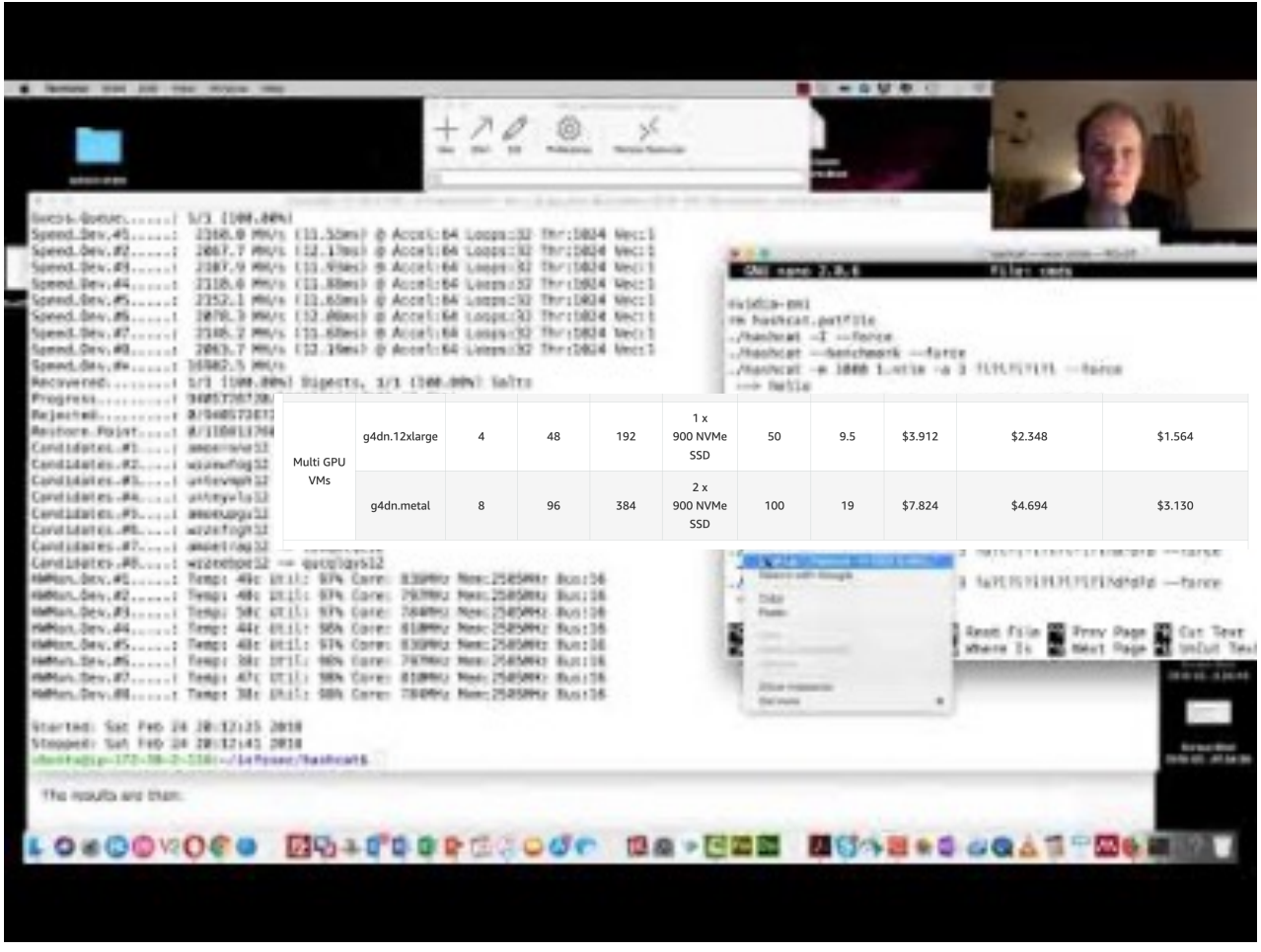
Name	GPUs	vCPU	Memory (GiB)	GPU Memory (GiB)	Price/hr* (Linux)	Price/hr* (Windows)	1-yr Reserved Instance Effective Hourly* (Linux)	3-yr Reserved Instance Effective Hourly* (Linux)
g3s.xlarge	1	4	30.5	8	$0.75	$0.93	$0.525	$0.405
g3.4xlarge	1	16	122	8	$1.14	$1.876	$0.741	$0.538
g3.8xlarge	2	32	244	16	$2.28	$3.752	$1.482	$1.076
g3.16xlarge	4	64	488	32	$4.56	$7.504	$2.964	$2.152



Multi GPU VMs	g4dn.12xlarge	4	48	192	1 x 900 NVMe SSD	50	9.5	$3.912	$2.348	$1.564
	g4dn.metal	8	96	384	2 x 900 NVMe SSD	100	19	$7.824	$4.694	$3.130

Hashcat – With 8xGPU

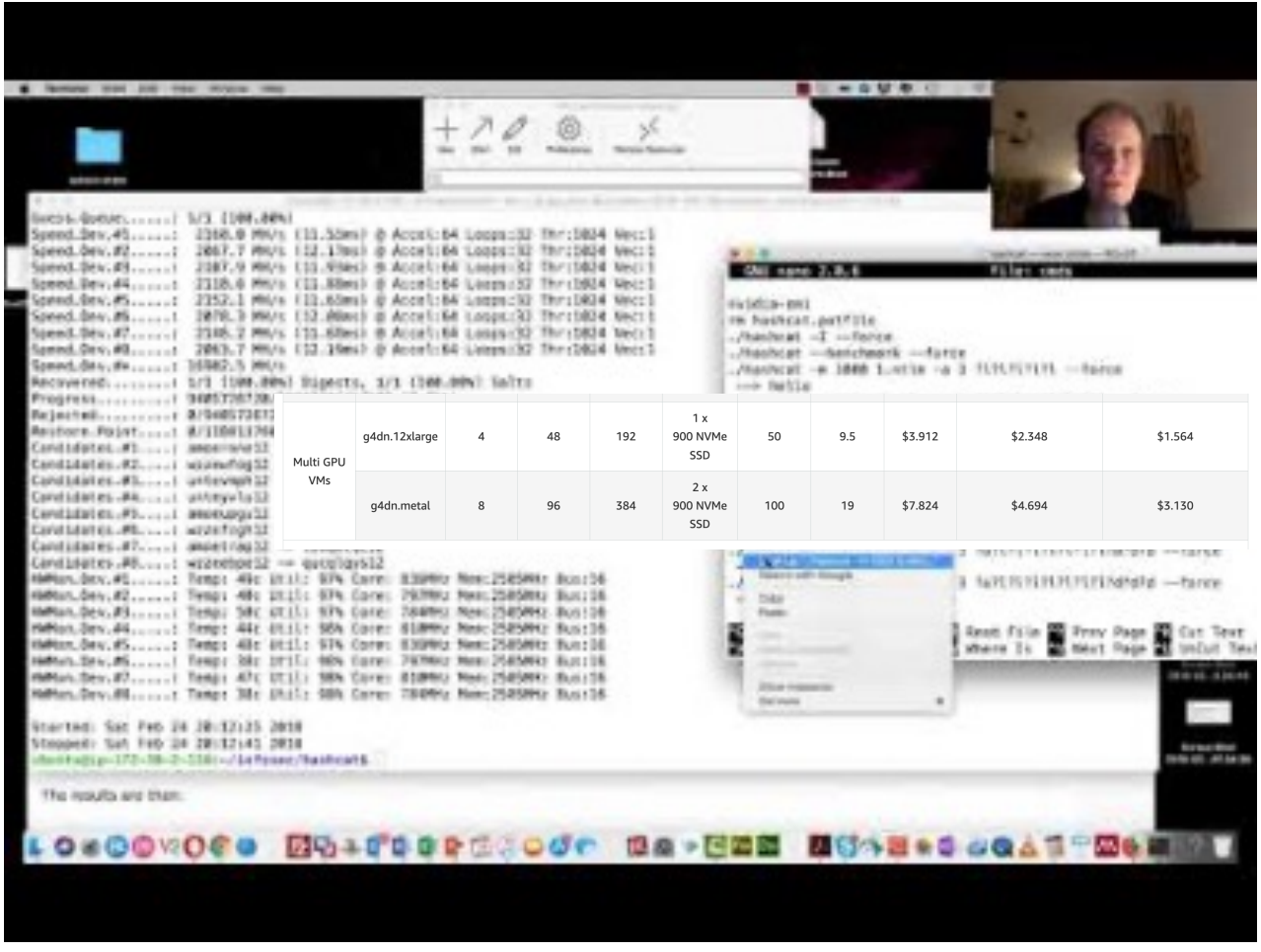
Name	GPUs	vCPU	Memory (GiB)	GPU Memory (GiB)	Price/hr* (Linux)	Price/hr* (Windows)	1-yr Reserved Instance Effective Hourly* (Linux)	3-yr Reserved Instance Effective Hourly* (Linux)
g3s.xlarge	1	4	30.5	8	$0.75	$0.93	$0.525	$0.405
g3.4xlarge	1	16	122	8	$1.14	$1.876	$0.741	$0.538
g3.8xlarge	2	32	244	16	$2.28	$3.752	$1.482	$1.076
g3.16xlarge	4	64	488	32	$4.56	$7.504	$2.964	$2.152



Multi GPU VMs	g4dn.12xlarge	4	48	192	1 x 900 NVMe SSD	50	9.5	$3.912	$2.348	$1.564
	g4dn.metal	8	96	384	2 x 900 NVMe SSD	100	19	$7.824	$4.694	$3.130

Hashcat – With 8xGPU

Name	GPUs	vCPU	Memory (GiB)	GPU Memory (GiB)	Price/hr* (Linux)	Price/hr* (Windows)	1-yr Reserved Instance Effective Hourly* (Linux)	3-yr Reserved Instance Effective Hourly* (Linux)
g3s.xlarge	1	4	30.5	8	$0.75	$0.93	$0.525	$0.405
g3.4xlarge	1	16	122	8	$1.14	$1.876	$0.741	$0.538
g3.8xlarge	2	32	244	16	$2.28	$3.752	$1.482	$1.076
g3.16xlarge	4	64	488	32	$4.56	$7.504	$2.964	$2.152



Multi GPU VMs	g4dn.12xlarge	4	48	192	1 x 900 NVMe SSD	50	9.5	$3.912	$2.348	$1.564
	g4dn.metal	8	96	384	2 x 900 NVMe SSD	100	19	$7.824	$4.694	$3.130

GPUs in the Cloud

G5: NVIDIA A10G Tensor Core GPUs. Up to to 192 vCPUs. 31.2 teraFLOPS, 24GB RAM.

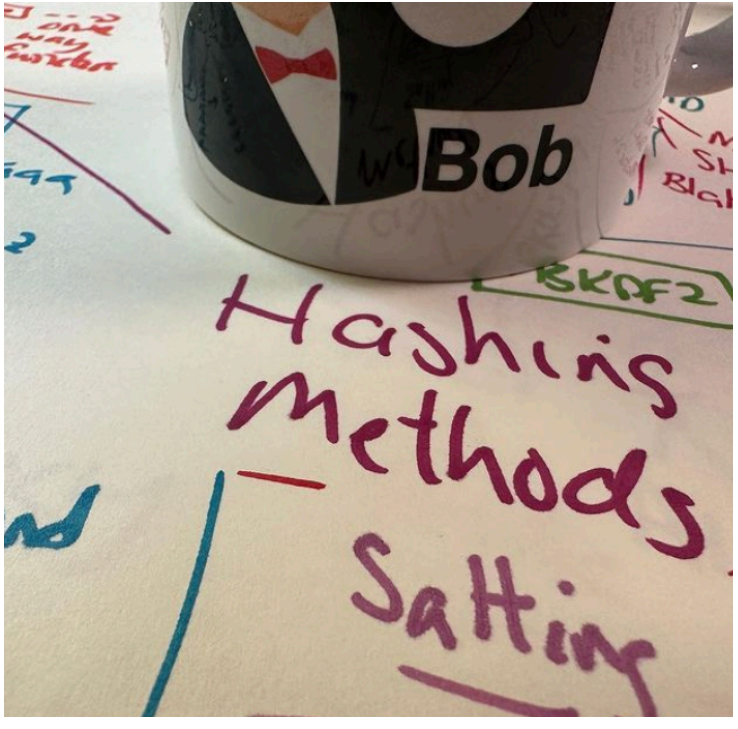
Name	GPUs	vCPU	Memory (GiB)	GPU Memory (GiB)	Price/hr* (Linux)	Price/hr* (Windows)	1-yr Reserved Instance Effective Hourly* (Linux)	3-yr Reserved Instance Effective Hourly* (Linux)
g3s.xlarge	1	4	30.5	8	$0.75	$0.93	$0.525	$0.405
g3.4xlarge	1	16	122	8	$1.14	$1.876	$0.741	$0.538
g3.8xlarge	2	32	244	16	$2.28	$3.752	$1.482	$1.076
g3.16xlarge	4	64	488	32	$4.56	$7.504	$2.964	$2.152

Multi GPU VMs	g4dn.12xlarge	4	48	192	1 x 900 NVMe SSD	50	9.5	$3.912	$2.348	$1.564
	g4dn.metal	8	96	384	2 x 900 NVMe SSD	100	19	$7.824	$4.694	$3.130

Multi GPU VMs	g5.12xlarge	4	96	48	192	1x3800	40	16	$5.672	$3.403	$2.269
	g5.24xlarge	4	96	96	384	1x3800	50	19	$8.144	$4.886	$3.258
	g5.48xlarge	8	192	192	768	2x3800	100	19	$16.288	$9.773	$6.515

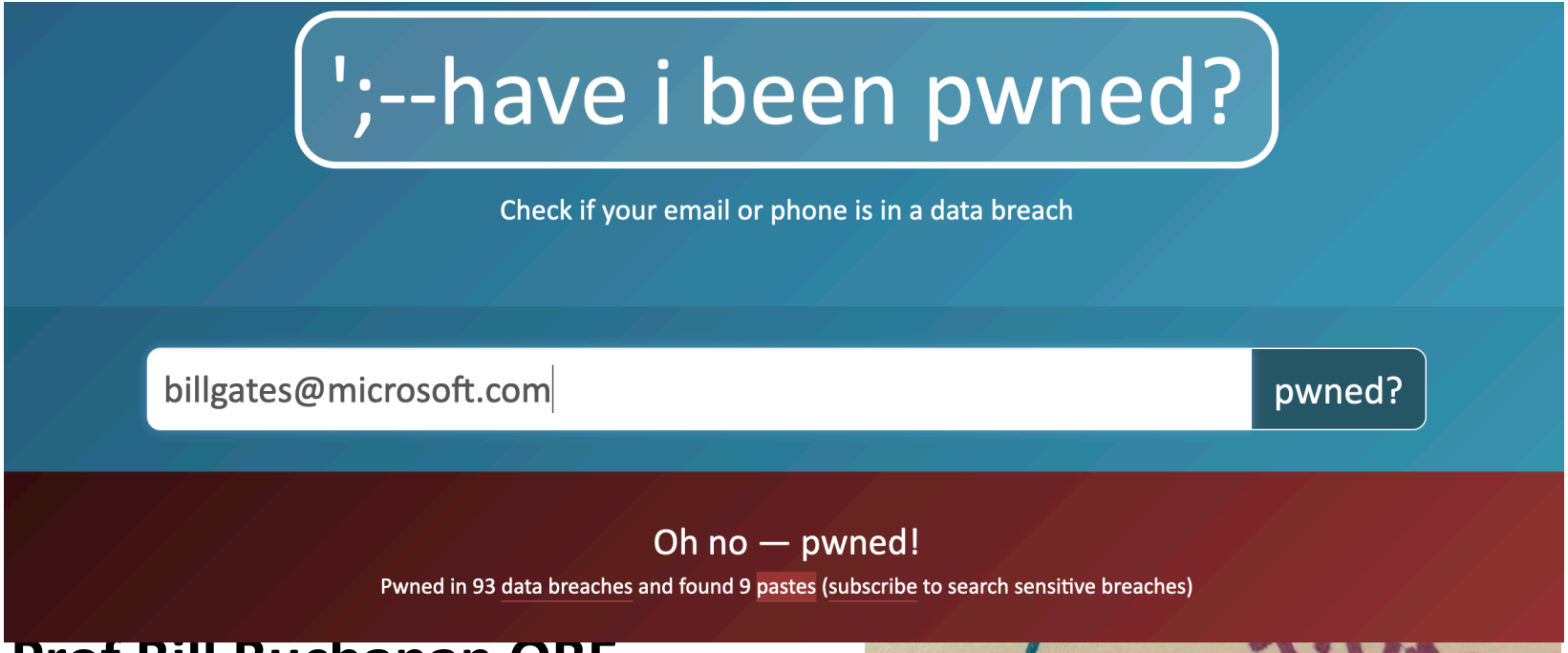
Unit 3: Hashing

Hashing Types.
Hashing Methods.
Salting.
Collisions.
LM and NTLM Hashes (Windows).
Hash Benchmarks.
Message Authentication Codes (MACs).
OTP/HOTP.
Hashcat **(for practical work)**



Prof Bill Buchanan OBE
https://asecuritysite.com/hash

Unit 3: Hashing



Prof Bill Buchanan OBE

https://asecuritysite.com/hash