IN5290 Ethical hacking

Ethical hacking cryptography

Laszlo Erdödi

laszloe@ifi.uio.no

Lecture Overview

- Where to use crypto in hacking?
- How passwords are stored
- What is a hash, what are the characteristics
- Hash cracking methods
- John the ripper/Hashcat usage
- Secure file storage
- Secure messaging

Where to use crypto in ethical hacking?

- Ethical hackers usually have non confidential agreement, all data used/revealed during the project should be well protected
- All communication to client or inside the ethical hacking team must be secured

What happens if the ethical hacker reveals (accidentally) the vulnerabilities / data found during the penetration testing?

- Ethical hackers might find encrypted data during the penetration testing that should be tested (can it be cracked or not)
  - Password hashes
  - Encrypted files
  - Encrypted databases

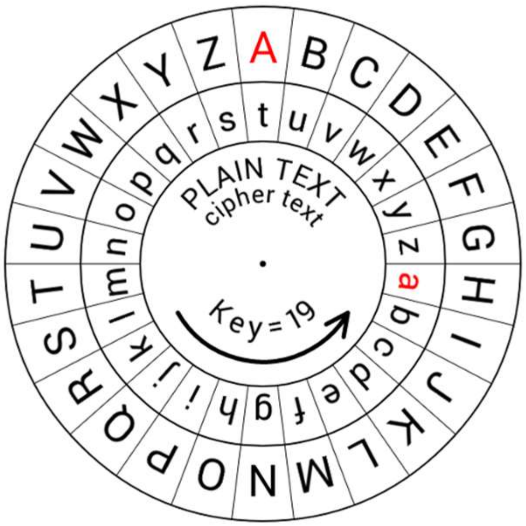
Type of encryptions

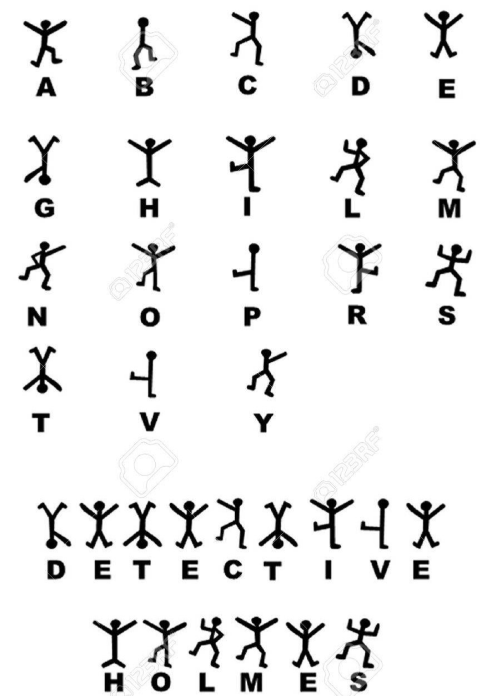
What are the easiest encryptions?

- Ceaser’s cipher? (Julius Ceaser used in military communication)
- Other letter substitution?

    Sherlock Holmes dancing man code?

What would be the strongest encryption?





Online crypto sites

https://dcode.fr

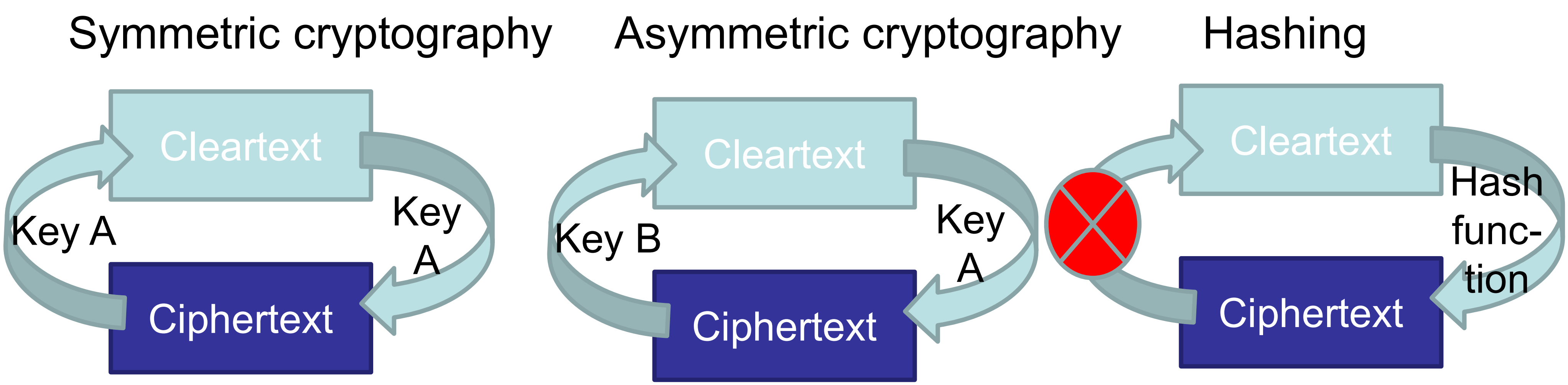
- Hash identifier/ cipher identifier
- Many different encryption/decryption algorithm e.g. ceasar cipher/ vigenere cipher/ Morse / Brainf*ck/ many others

https://gchq.github.io/CyberChef/

- Many different encryptions/ public key tools e.g. PGP decrypt / many others

Type of encryptions

- In symmetric cryptography we use one key to create the cipher text and the same key to get back the key text
- In asymmetric cryptography we have a key pair, one is used for encryption and the other is for decryption
- In case of hashing we produce a hash that cannot be reverted to cleartext



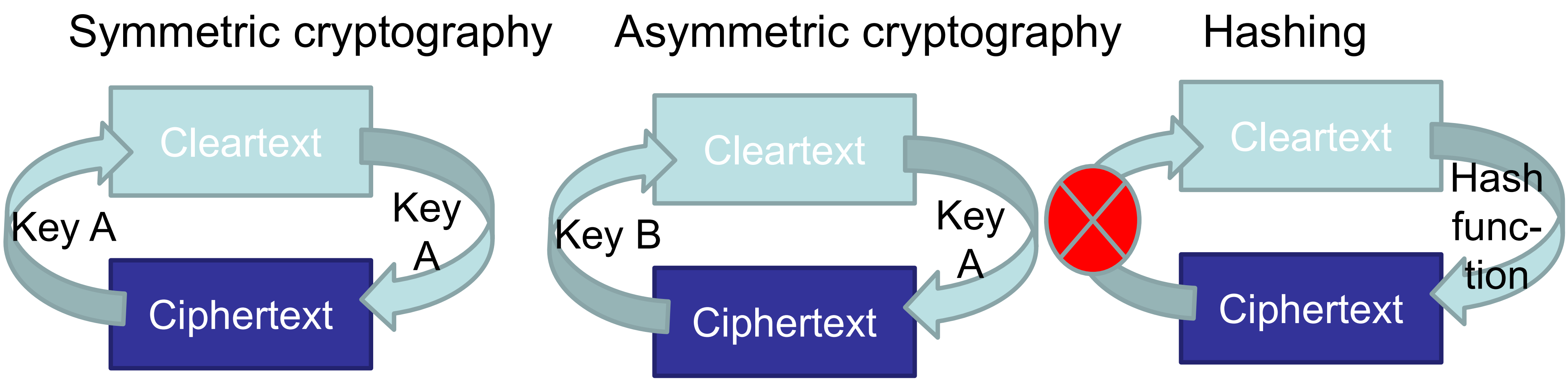
Password storage

Having login function for websites requires to store the usernames and passwords:

- When the user register an account, a new dataset is created in the database with the username and the provided password
- When the user logs in, the provided password is compared with the one that is stored for the user, if they match the user gets appropriate session
- The easiest (but very unsecure) way of storing the password is to store the username and the password as “cleartext”
- Storing the password as a cleartext has the danger that anyone who has access to the database (even if an attacker dumps the database e.g. with SQL injection) has all usernames and passwords, therefore the passwords have to be stored in a much more secure way

What is a hash?

A hash function is any function that can be used to map data of arbitrary size to fixed-size values. It is a one-way function, which is practically infeasible to invert or reverse the computation. Hashing is also deterministic, the same input always provides the same output, the hash.



Hashing algorithms

Comparison of SHA functions

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Rounds	Operations	Security bits (Info)	Capacity against length extension attacks	Performance on Skylake (median cpb)[1]		First Published
										long messages	8 bytes	
MD5 (as reference)		128	128 (4 × 32)	512	Unlimited[2]	64	And, Xor, Rot, Add (mod 2^{32}), Or	<64 (collisions found)	0	4.99	55.00	1992
SHA-0		160	160 (5 × 32)	512	$2^{64} - 1$	80	And, Xor, Rot, Add (mod 2^{32}), Or	<34 (collisions found)	0	≈ SHA-1	≈ SHA-1	1993
SHA-1								<63 (collisions found[3])		3.47	52.00	1995
SHA-2	*SHA-224* *SHA-256*	224 256	256 (8 × 32)	512	$2^{64} - 1$	64	And, Xor, Rot, Add (mod 2^{32}), Or, Shr	112 128	32 0	7.62 7.63	84.50 85.25	2004 2001
	SHA-384 *SHA-512*	384 512	512 (8 × 64)	1024	$2^{128} - 1$	80	And, Xor, Rot, Add (mod 2^{64}), Or, Shr	192 256	128 (≤ 384) 0	5.12 5.06	135.75 135.50	
	SHA-512/224 *SHA-512/256*	224 256						112 128	288 256	≈ SHA-384	≈ SHA-384	
SHA-3	*SHA3-224* *SHA3-256* *SHA3-384* *SHA3-512*	224 256 384 512	1600 (5 × 5 × 64)	1152 1088 832 576	Unlimited[4]	24[5]	And, Xor, Rot, Not	112 128 192 256	448 512 768 1024	8.12 8.59 11.06 15.88	154.25 155.50 164.00 164.00	2015
	SHAKE128 *SHAKE256*	*d* (arbitrary) *d* (arbitrary)		1344 1088				min(d/2, 128) min(d/2, 256)	256 512	7.08 8.59	155.25 155.50	

Hash cracking

- **Brute-force based:** The attacker tries out all combinations, time consuming
- **Dictionary based:** The attacker has a list of possible clear texts, only those hashes are cracked that were in the list
- **Hybrid:** It combines dictionary attacks with brute-forcing. Not only the dictionary words but slight modifications of it are tried. *Trondheim -> Tr0ndhe1m, miehdnorT, TRONDHEIM, etc.*
- **Rainbow tables:** It uses precalculated hashes that are ordered in chains and very effective to store and search

Brute-force password cracking

The attacker tries all combinations:

- Calculate the hash of the first cleartext
- Compare the result with the hash that has to be cracked
- If it is identical then the cleartext was found
- If it is not identical then the next clear text has to be checked
- The number of combinations depends on 2 parameter:
  - The alphabet (which characters can be used)
  - The length of the password

Brute-force password cracking combination

Number of combination examples:

- English lower case alphabets, password length is 8: 26^8= 208.827.064.576
- English lower and upper case alphabets, password length is 8: 52^8= 53.459.728.531.456
- English lower and upper case alphabets, numbers, special characters, password length is 8: 78^8= 1.370.114.370.683.136
- English lower and upper case alphabets, numbers, special characters, password length is 10: 78^8= 8.335.775.831.236.199.424

How many MD5 passwords can we calculate in a second?

Forcing stronger passwords

Please provide your new password:

potato

I’m sorry the password must contain at least 8 characters:

mashedpotato

I’m sorry the password must contain at least one digit:

50mashedpotato

I’m sorry the password must contain at least one special character:

50mashed-potato

I’m sorry the password must contain at least one capital letter:

50G@DDAMNmashed-potato!!!!!

I’m sorry you cannot use your old password again:

Dictionary based password cracking

The attacker tries all cleartexts in the dictionary file:

- Calculate the hash of the first cleartext
- Compare the result with the hash that has to be cracked
- If it is identical then the cleartext was found
- If it is not identical then the next clear text has to be taken from the dictionary
- The number of combinations depends on cleartexts in the dictionary file:
  - Normal words
  - Sleng
  - Geography names
  - Famous people
  - Etc.

Dictionary based password cracking

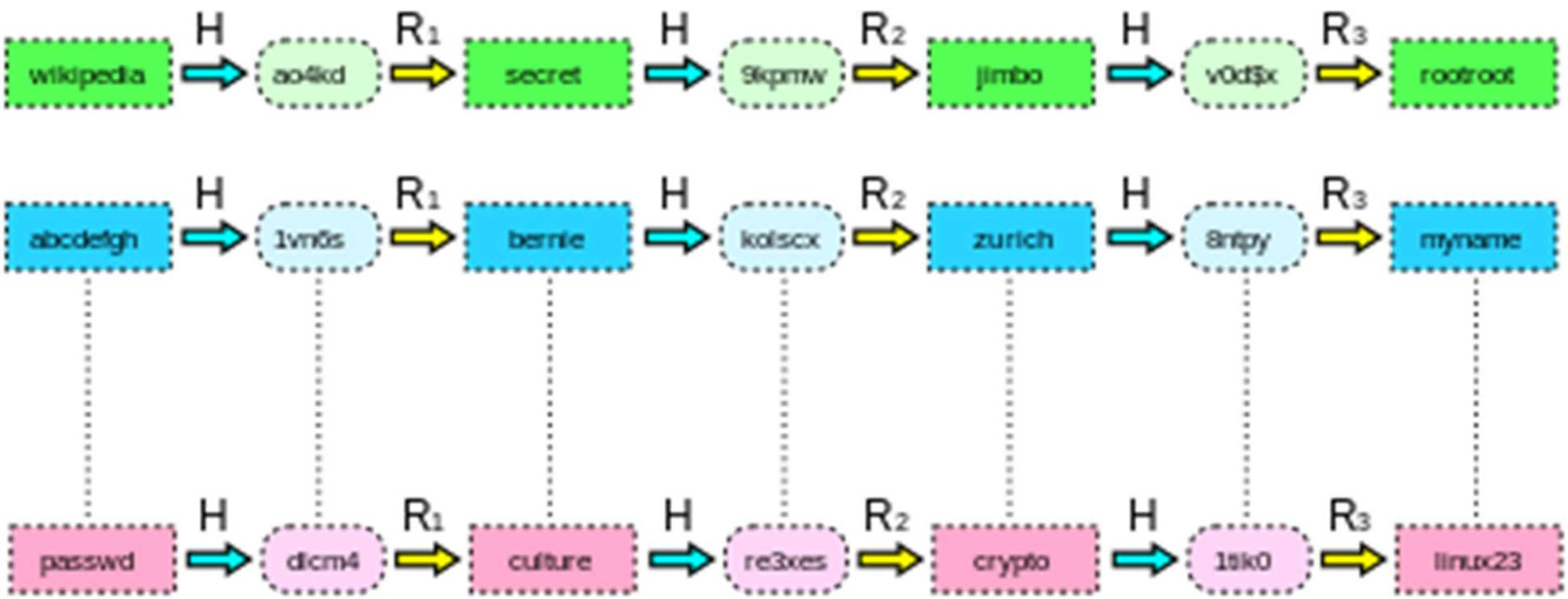
Hybrid password cracking

The attacker tries all cleartexts in the dictionary file an its permutations:

- Calculate the hash of the first cleartext
- Compare the result with the hash that has to be cracked
- If it is identical then the cleartext was found
- If it is not identical then the next version of the current clear text has to be considered
- If there is no more hybrid version then the next version has to be taken
- Hybrid words:
  - Double (TrondheimTrondheim)
  - Reverse (miehdnorT)
  - Substitute (Tr0ndh41m)
  - Numbers added (Trondheim0 – Trondheim99)

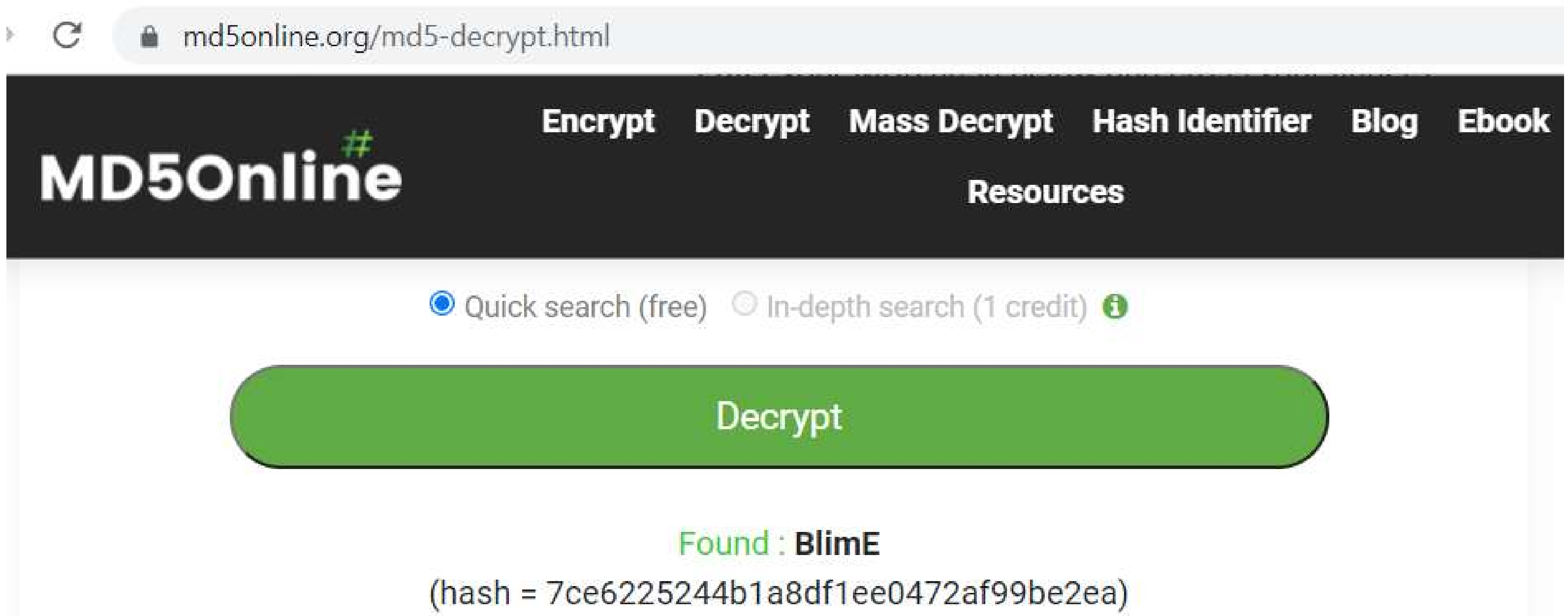
Rainbow tables

- The problem with brute-forcing that it is very slow
- The problem with pre-calculation that there is not enough space to store the hashes
- One mixing idea is the rainbow table:



Precomputed hash databases

- https://www.md5online.org



- https://md5decrypt.net/
- and many others

John the ripper

```
Copyright (c) 1996-2019 by Solar Designer and others
Homepage: http://www.openwall.com/john/

Usage: john [OPTIONS] [PASSWORD-FILES]
--single[=SECTION[,..]]    "single crack" mode, using default or named rules
--single=:rule[,..]        same, using "immediate" rule(s)
--wordlist[=FILE] --stdin  wordlist mode, read words from FILE or stdin
                  --pipe   like --stdin, but bulk reads, and allows rules
--loopback[=FILE]          like --wordlist, but extract words from a .pot file
--dupe-suppression         suppress all dupes in wordlist (and force preload)
--prince[=FILE]            PRINCE mode, read words from FILE
--encoding=NAME            input encoding (eg. UTF-8, ISO-8859-1). See also
                           doc/ENCODINGS and --list=hidden-options.
--rules[=SECTION[,..]]     enable word mangling rules (for wordlist or PRINCE
                           modes), using default or named rules
--rules=:rule[;..]]        same, using "immediate" rule(s)
--rules-stack=SECTION[,..] stacked rules, applied after regular rules or to
                           modes that otherwise don't support rules
--rules-stack=:rule[;..]   same, using "immediate" rule(s)
--incremental[=MODE]       "incremental" mode [using section MODE]
--mask[=MASK]              mask mode using MASK (or default from john.conf)
--markov[=OPTIONS]         "Markov" mode (see doc/MARKOV)
--external=MODE            external mode or word filter
--subsets[=CHARSET]        "subsets" mode (see doc/SUBSETS)
--stdout[=LENGTH]          just output candidate passwords [cut at LENGTH]
--restore[=NAME]           restore an interrupted session [called NAME]
--session=NAME             give a new session the NAME
--status[=NAME]            print status of a session [called NAME]
--make-charset=FILE        make a charset file. It will be overwritten
--show[=left]              show cracked passwords [if =left, then uncracked]
--test[=TIME]              run tests and benchmarks for TIME seconds each
--users=[-]LOGIN|UID[,..]  [do not] load this (these) user(s) only
--groups=[-]GID[,..]       load users [not] of this (these) group(s) only
--shells=[-]SHELL[,..]     load users with[out] this (these) shell(s) only
--salts=[-]COUNT[:MAX]     load salts with[out] COUNT [to MAX] hashes
--costs=[-]C[:M][,...]     load salts with[out] cost value Cn [to Mn]. For
                           tunable cost parameters, see doc/OPTIONS
```

Hashcat hash-modes

Hash-Mode	Hash-Name	Example
0	MD5	8743b52063cd84097a65d1633f5c74f5
10	md5($pass.$salt)	01dfae6e5d4d90d9892622325959afbe:7050461
20	md5($salt.$pass)	f0fda58630310a6dd91a7d8f0a4ceda2:4225637426
30	md5(utf16le($pass).$salt)	b31d032cfdcf47a399990a71e43c5d2a:144816
40	md5($salt.utf16le($pass))	d63d0e21fdc05f618d55ef306c54af82:13288442151473
50	HMAC-MD5 (key = $pass)	fc741db0a2968c39d9c2a5cc75b05370:1234
60	HMAC-MD5 (key = $salt)	bfd280436f45fa38eaacac3b00518f29:1234
70	md5(utf16le($pass))	2303b15bfa48c74a74758135a0df1201
100	SHA1	b89eaac7e61417341b710b727768294d0e6a277b
110	sha1($pass.$salt)	2fc5a684737ce1bf7b3b239df432416e0dd07357:2014
120	sha1($salt.$pass)	cac35ec206d868b7d7cb0b55f31d9425b075082b:5363620024
130	sha1(utf16le($pass).$salt)	c57f6ac1b71f45a07dbd91a59fa47c23abcd87c2:631225
140	sha1($salt.utf16le($pass))	5db61e4cd8776c7969cfd62456da639a4c87683a:8763434884872
150	HMAC-SHA1 (key = $pass)	c898896f3f70f61bc3fb19bef222aa860e5ea717:1234
160	HMAC-SHA1 (key = $salt)	d89c92b4400b15c39e462a8caa939ab40c3aeeea:1234
170	sha1(utf16le($pass))	b9798556b741befdbddcbf640d1dd59d19b1e193
200	MySQL323	7196759210defdc0

Why to use salts when hashing?

Salting is simply the addition of a unique, random string of characters known only to the site to each password before it is hashed, typically this “salt” is placed in front of each password.

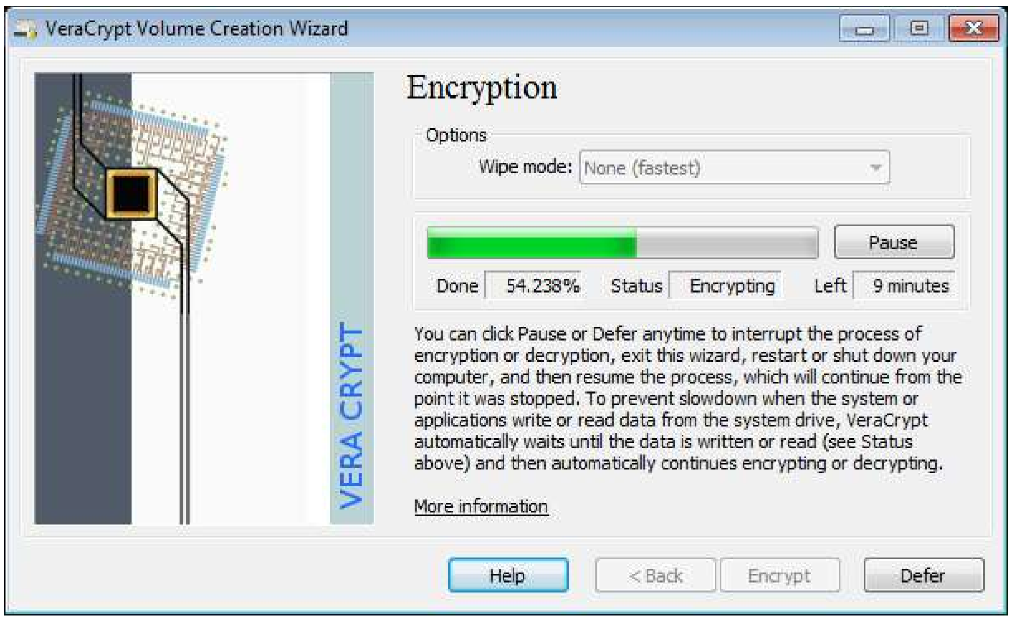
Userid	username	password	salt
…			
768	Laszlo	ABCDEF123456789…	sunglasses
769	Lennon	1234567812345678…	strawberry
770	McCartney	9876543219876543…	camembert

Laszlo’s stored hash = MD5(‘my password’+’sunglasses’) = ABCDEF123456789…

Secure file storage

How to protect the files/ data that we use during the penetration testing?

- Full disk encryption
- Encrypted containers
  - E.g. Veracrypt



Secure communication

- Does email secure? ☺
- Does SMS secure? ☺

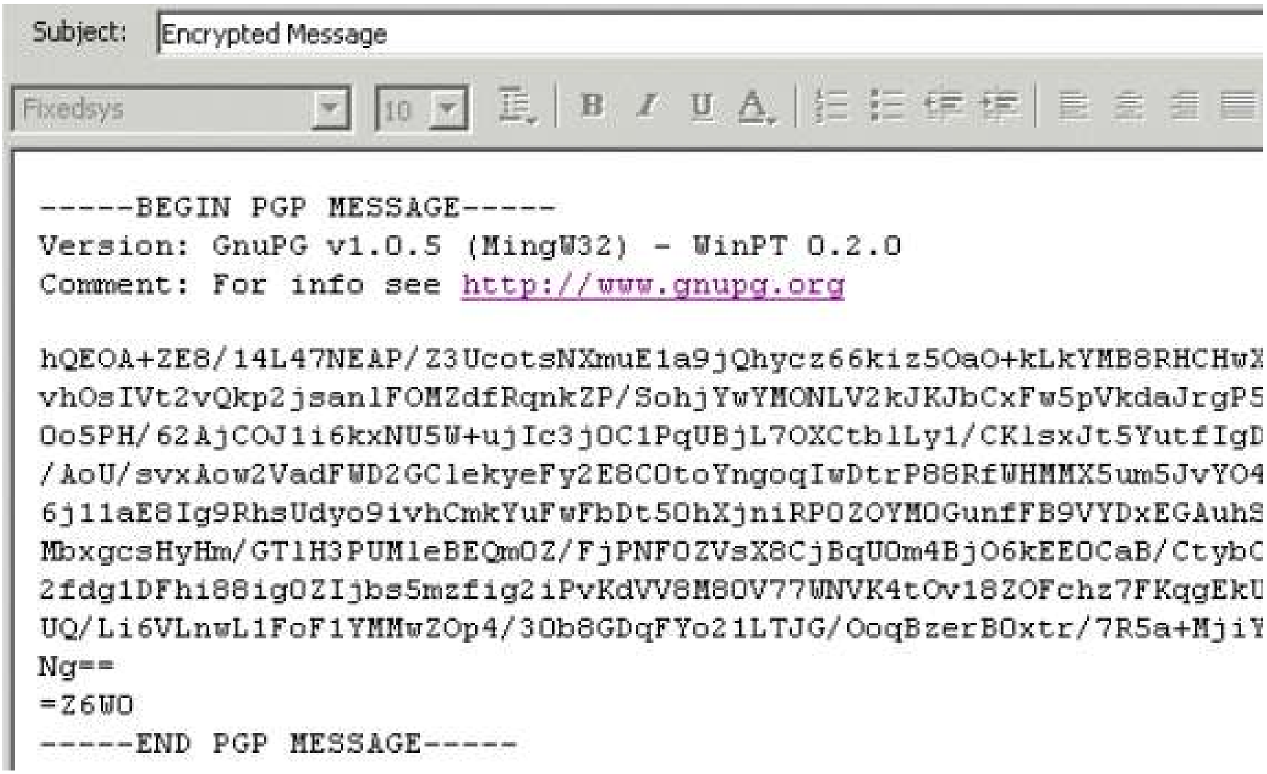
- Use PGP for secure emailing
- Use secure messenger applications (end to end encryption)
- Use multiple channels for communication

Pretty Good Privacy (PGP)

- Everyone has a key pair: public key + private key
- The public key is shared with others
- The private key should be secured

A	text	→	B	Cleartext?
A	Apriv(text)	→	B	Encrypted with A private?
A	Bpub(text)	→	B	Encrypted with B public?
A	Bpub(Apriv(text))	→	B	Encrypted with both?

Pretty Good Privacy (PGP)



End of lecture