



COMPUTER SYSTEMS SECURITY

LECTURE 2: HASHING AND PASSWORD CRACKING

EDLIRA DUSHKU
ASSISTANT PROFESSOR
EDU@ES.AAU.DK

[copyrighted material – for use in AAU only]



AALBORG UNIVERSITY
DENMARK



Communication, Media and Information technologies

Hashing functions

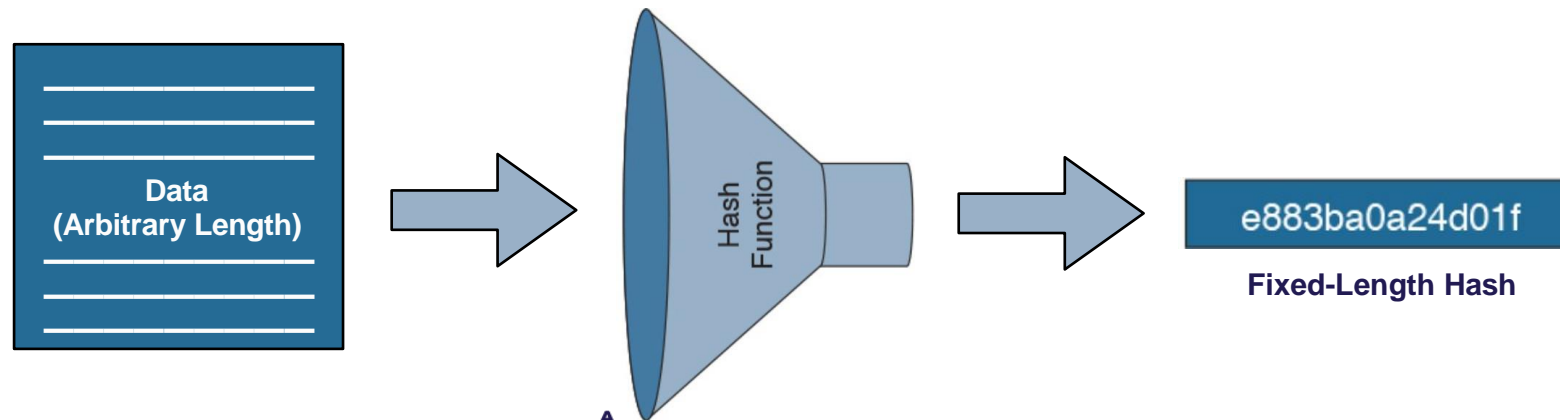
Solution for integrity protection



Hash Functions (1975): Easy to compute but hard to invert

- The **input** can be any string of **any size**
- It produces a **fixed-size output**
- It is **efficiently computable**

Examples: SHA-256, SHA-512, SHA-3



Hash Functions

- A **hash function** H maps arbitrary strings of data (message) to fixed-length output (hash value or message digest).



- Hash functions do not have a secret key.
- The hash function H is public, anyone can evaluate the function.
- The function is deterministic and public, but the mapping should look “random”.

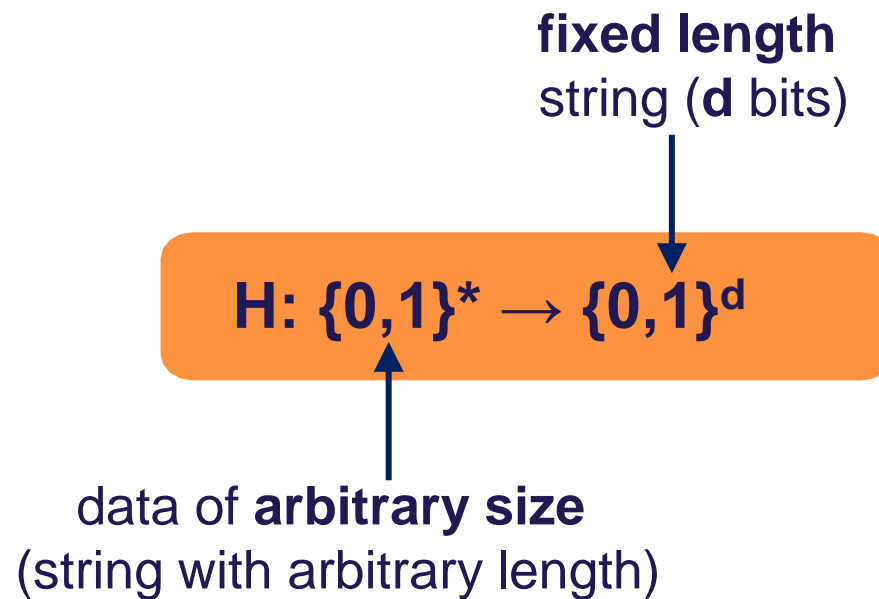


Hash Functions

- A **collision** is a pair of plaintexts M_1 and M_2 that map to the same hash value, $H(M_1) = H(M_2)$
- Collisions are unavoidable (mapping an infinite domain to a finite one), but unlikely (if H designed properly)
- **Hash speed:** For efficiency, the computation of the hash function should take time proportional to the length of the input plaintext



Hash Functions



Example:

MD5: d is **128** bits

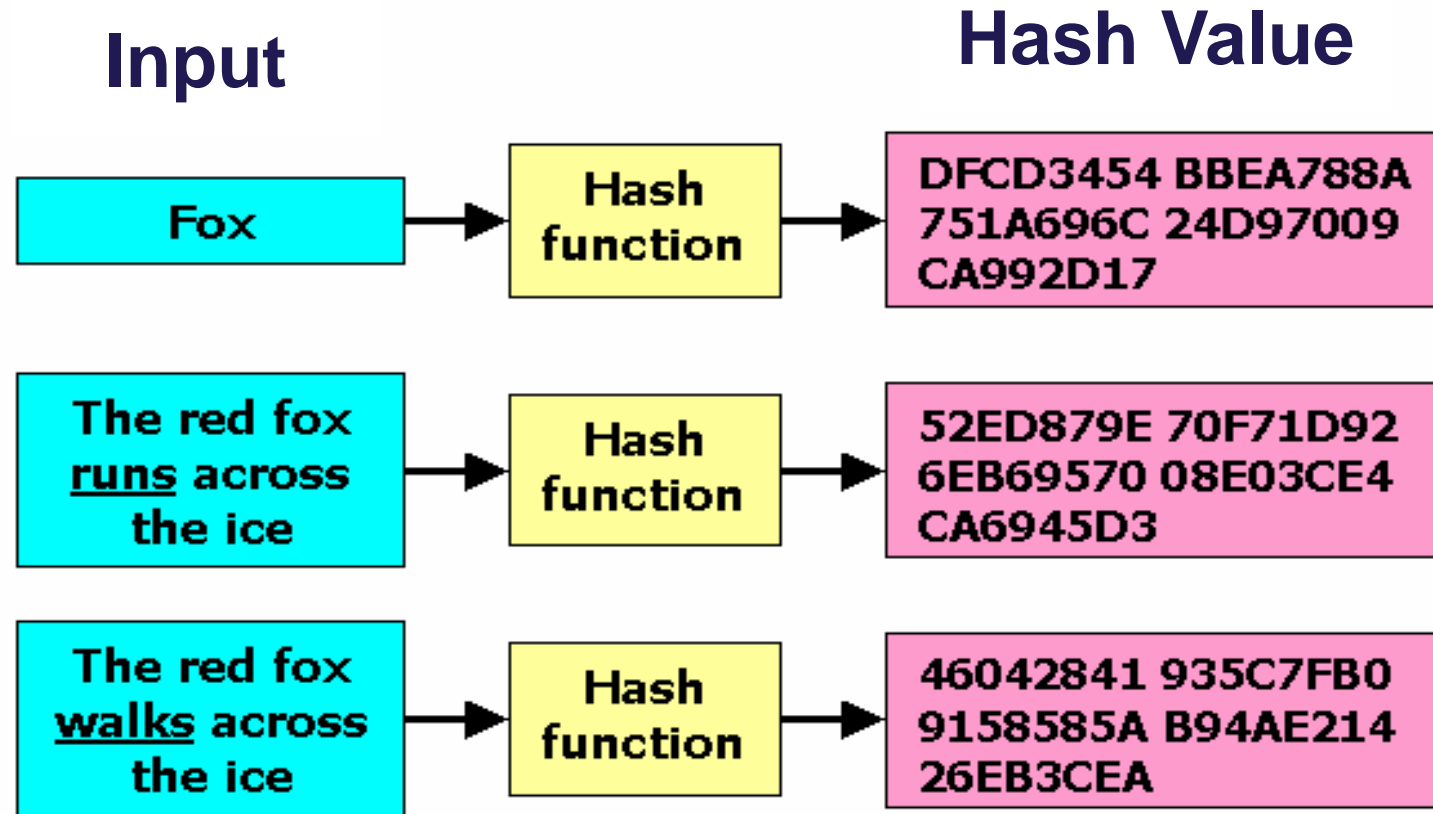
SHA1: d is **160** bits

SHA2: d is **256** bits



Examples of Hash Functions

Hash produces results of the same length



Hashing ensures Data Integrity

Original Email



Hi, Casey!

Here's the link to that great article on TLS encryption that I told you about last week:

website.com/tls-encryption-rocks...

da39a3ee5e6b4b0d3255bfef95601890afd80709

Altered Email



Hi, Casey!

Here's the link to that great article on TLS encryption that I told you about last week:

differentwebsite.com/tls-encryption-link...

ef378a3ee5e6b4b0d1235bfef95601890afd450345



Hashing algorithms: MD and SHA series

- **The Message Digest (MD) series**
 - Developed by Ron Rivest
 - MD2, MD4, MD5 and MD6:
- **The Secure Hash Algorithm (SHA) series**
 - Published by the National Institute of Standards and Technology (NIST)
 - SHA-0, SHA-1, SHA-2, SHA-3



Let's try hashing

- Compute the hash of a message *“This is a message!”* in MD5 and SHA-256

```
$ echo -n This is a message! | sha256sum  
8a08a470467c3c8ccccdd31e9aa90f0c5ae3e1b633286f1bd60dc5cdad1f2b98 -
```

```
$ echo -n This is a message! | md5sum  
395b22232b33b983c54856c8781351c0 -
```

```
$ sha256sum test1.txt  
8a08a470467c3c8ccccdd31e9aa90f0c5ae3e1b633286f1bd60dc5cdad1f2b98 test1.txt
```

→ Writing the same message inside a file

- What is the size of the output?
- Why the output of SHA-256 does not have 256 characters?
- Try to hash the message *“This is a message!!!”*. What happens to the output?



Cryptographic Hash Functions

A hash function is **cryptographically secure** if it satisfies the following **3 security properties**:

- 1. Collision resistance**
- 2. Preimage resistant**
- 3. Second preimage resistant**

Cryptographic Hash Functions

A **cryptographic hash function** satisfies additional properties:

Collision resistance (a.k.a. **strong collision resistance**):
It is hard (computationally infeasible) to find a pair of plaintexts M_1 and M_2 such that $H(M_1) = H(M_2)$

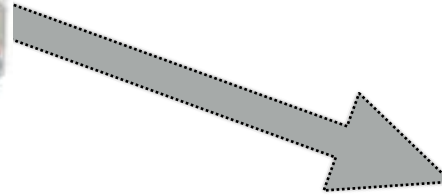
Preimage resistance (a.k.a. **one-way**):
Given a hash value P , it is hard to find a plaintext M such that $H(M) = P$

Second preimage resistance (a.k.a. **weak collision resistance**):
Given a plaintext M_1 , it is hard to find a plaintext M_2 such that $H(M_1) = H(M_2)$

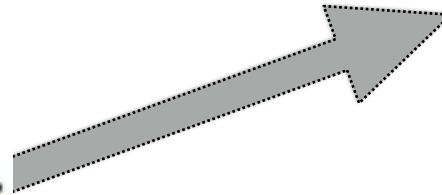
Hash values of at least 256 bits recommended to defend against brute-force attacks.



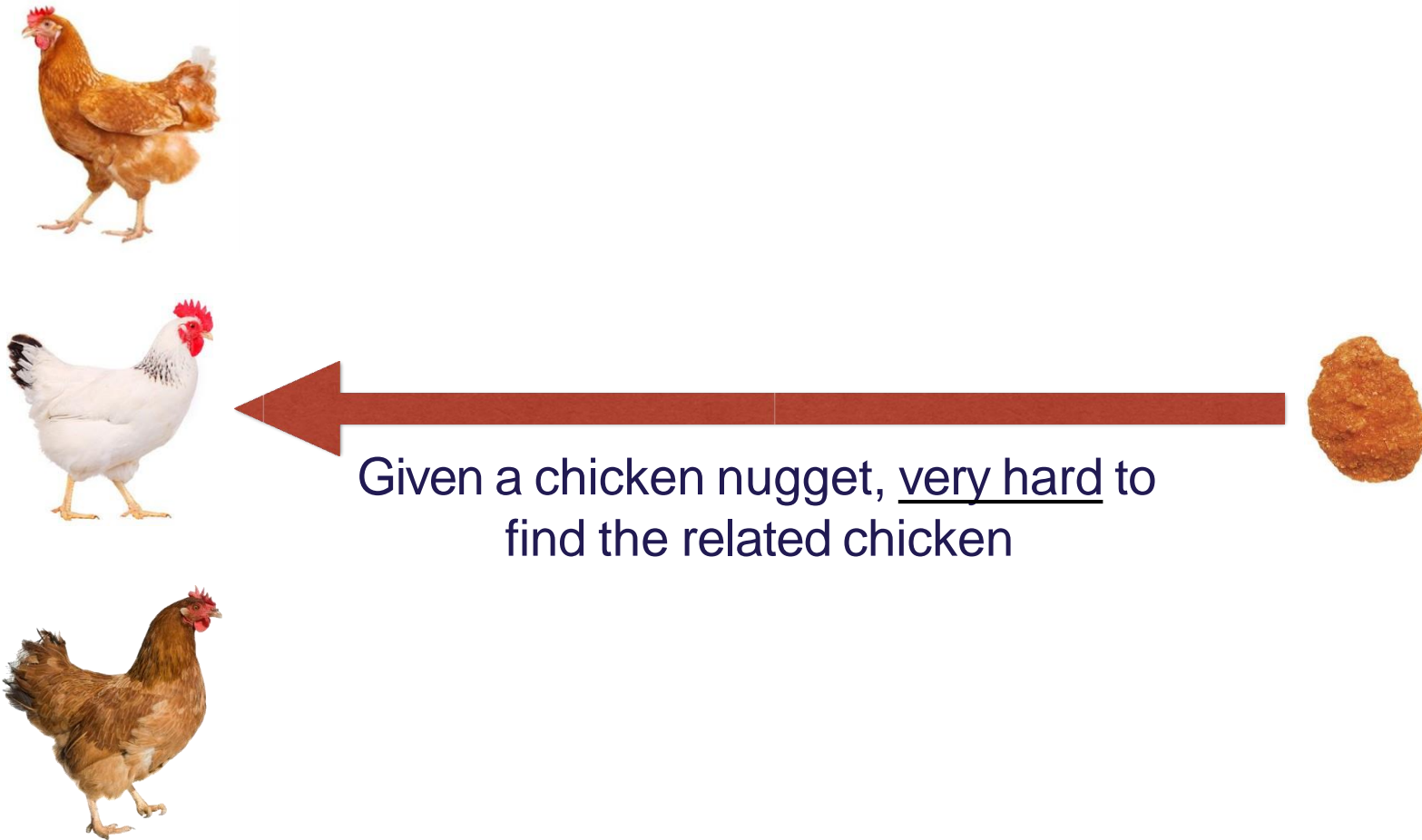
Property 1: Collision Resistance



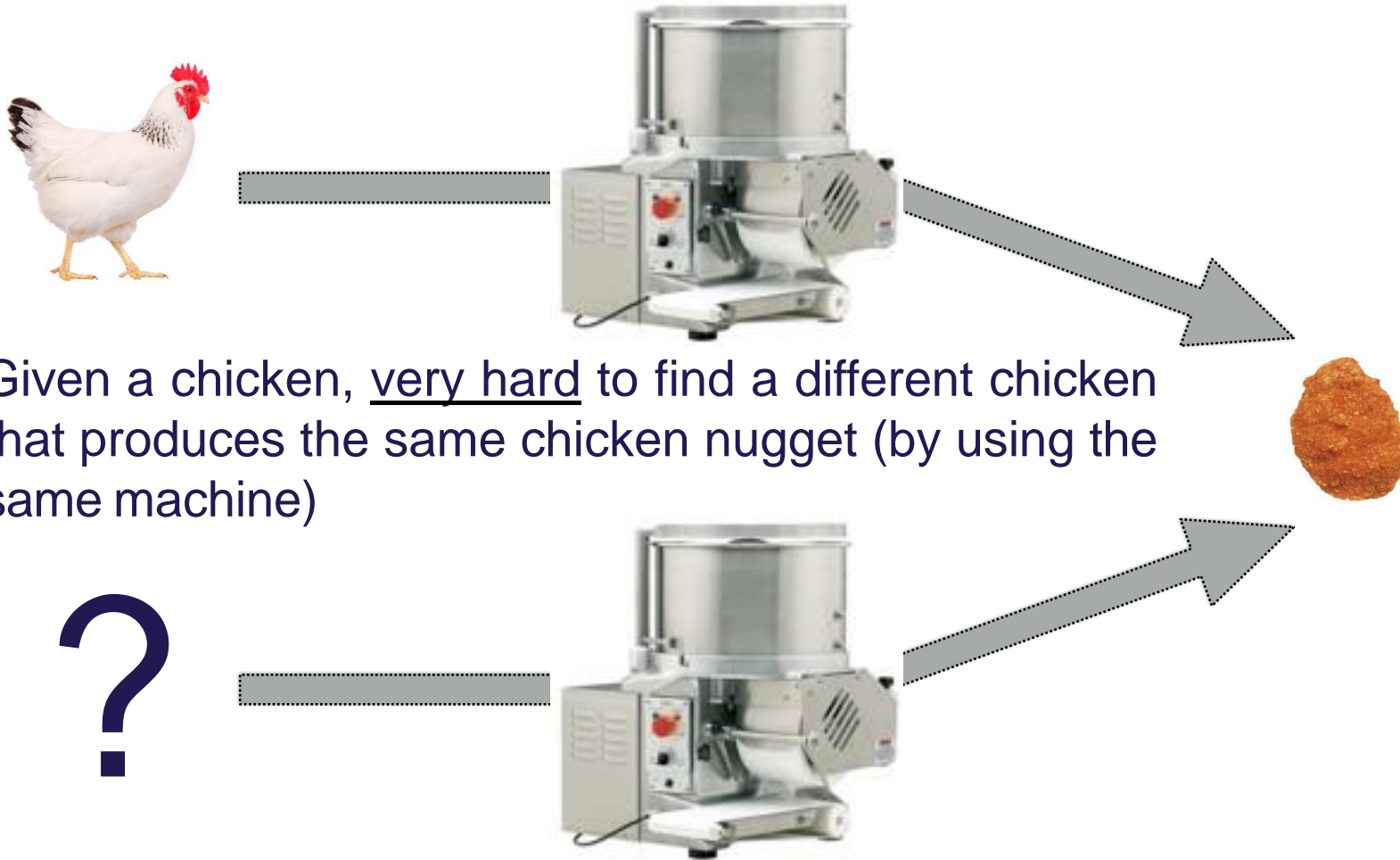
Very hard to find two different chickens that produces the same chicken nugget (by using the same machine)



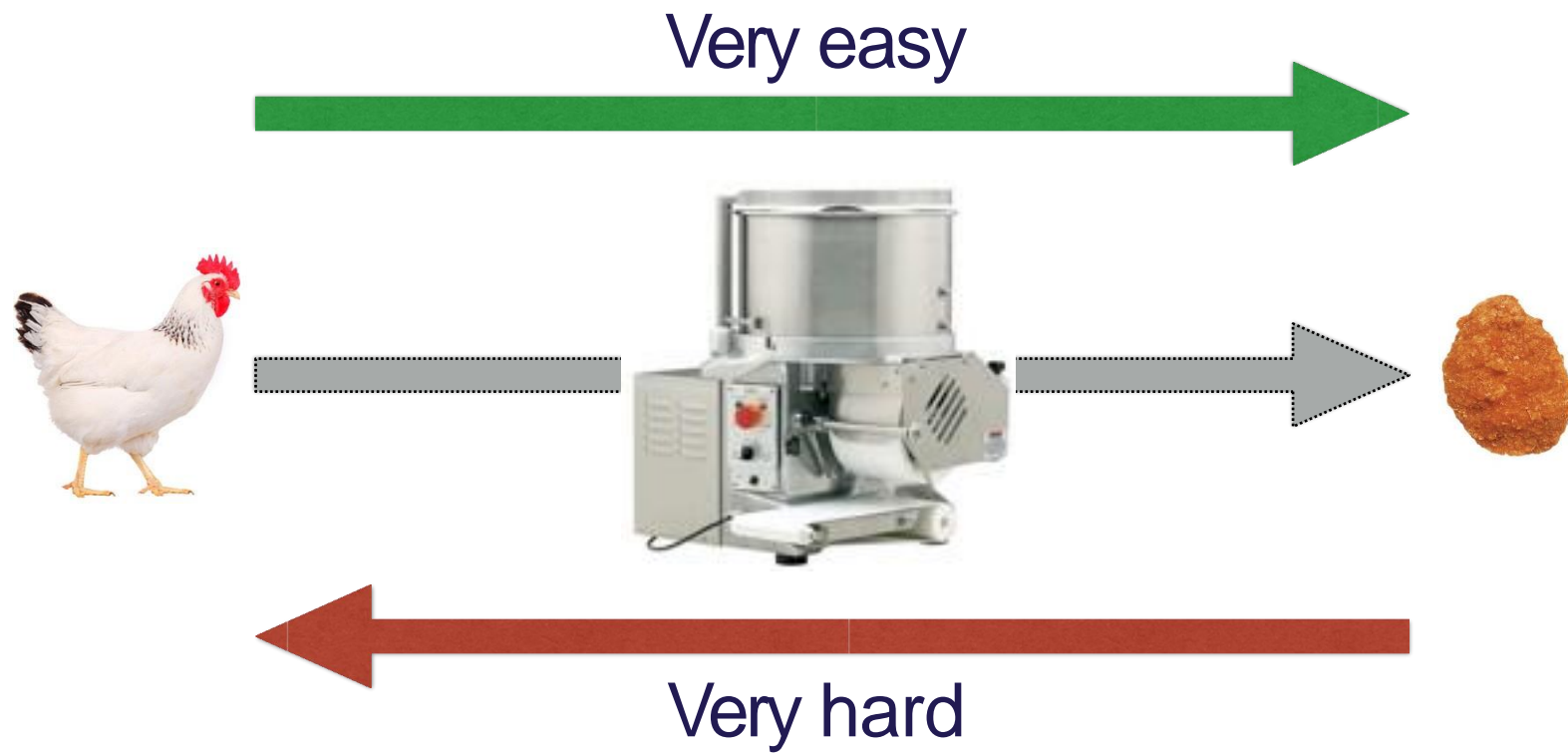
Property 2: Preimage Resistant



Property 3: Second Preimage Resistant



Cryptographic Hash Functions



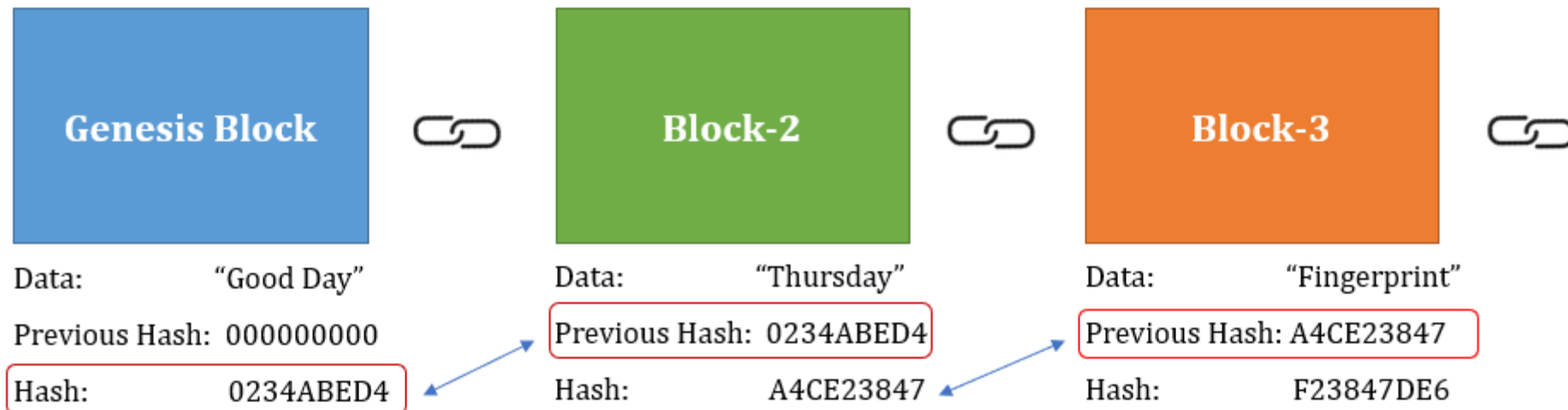
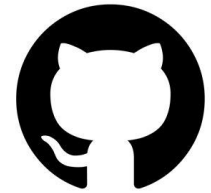
Properties of Hashing

- It is deterministic: the same message always results in the same hash. A hash algorithm always gives an output of identical size regardless of the size of the input.
- It is one-way: It is quick to compute the hash value for any given message but on the other hand it is infeasible to generate a message from its hash value except by trying all possible messages
- It is collision-free: It is infeasible to find two different messages with the same hash value
- It has avalanche effect: A small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value



Applications of Hashing

- **Storing Passwords in Databases:** store the hash of the password in the database table.
- **Digital Signatures and Authentication:** message digests (hashes) are widely used to provide some assurance that a transferred file has **arrived intact**. E.g. **Github**
- **Blockchains and Cryptocurrencies**



Hashing vs Encryption

- **Hashing:** Creates a unique "fingerprint" of the data. Changes to the data will result in a different hash, indicating tampering.
- **Encryption:** Transforms data into an unreadable format using a key. Decryption requires the correct key.

Feature	Hashing	Encryption
Purpose	Data Integrity Check	Confidentiality
Reversibility	Not Reversible	Reversible with a Key
Key Usage	No Key	Secret Key or Public/Private Key Pair
Output	Fixed-length "fingerprint"	Varies depending on algorithm
Security Focus	Detecting modifications	Preventing unauthorized access
Examples	MD5, SHA-256	AES, RSA
Use Cases	Verifying file integrity, password storage	Secure communication, protecting sensitive data



Hashing algorithms: MD and SHA series

- The Message Digest (MD) series

- Developed by Ron Rivest
- MD2, MD4, MD5 and MD6: → Relies on a different structure



They are all vulnerable.

- The Secure Hash Algorithm (SHA) series

- Published by the National Institute of Standards and Technology (NIST)
- SHA-0, SHA-1, SHA-2, SHA-3 → Relies on a different structure



Vulnerable



It is secure so far. You can find it with different names based on the size of the hash, e.g.,

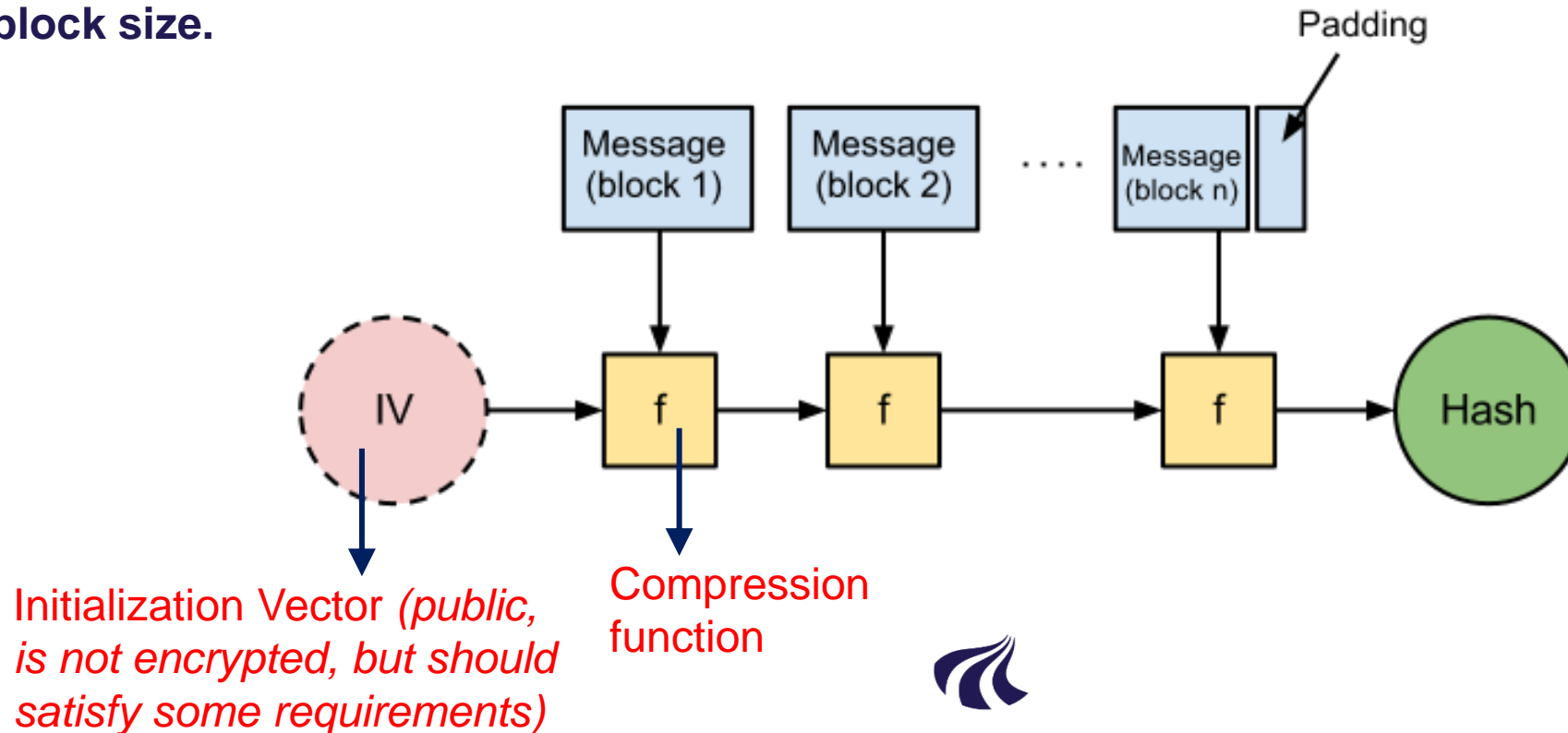
SHA-256

SHA-512



The Merkle–Damgård construction

- It is a **method of building collision-resistant cryptographic hash functions** from collision-resistant one-way compression functions.
- This construction was used in the design of many popular hash algorithms such as MD5, SHA-1 and SHA-2.
- Even though the structure is the same, SHA2 and MD5 have **different compression function and block size**.



Which one is easier to break?

Collision resistance (a.k.a. strong collision resistance):
It is hard to find a pair of plaintexts M_1 and M_2 such that $H(M_1) = H(M_2)$

Second preimage resistance (a.k.a. weak collision resistance):
Given a plaintext M_1 , it is hard to find a plaintext M_2 such that $H(M_1) = H(M_2)$

Birthday paradox



Birthday Paradox

Let's assume there are **N** people in a given room. What is the probability that **nobody** shares a birthday?

$$1 \times \frac{364}{365} \times \frac{363}{365} \times \dots \times \frac{365 - (N-1)}{365}$$

$$\frac{1}{365^N} \prod_{i=0}^{N-1} (365 - i)$$

This is the probability that we do not have any matches in a group of **N** people



Birthday Paradox

Let's assume there are **N people** in a given room. What is the probability that **at least 2 people** share the same birthday?

$$1 - \left[1 \times \frac{364}{365} \times \frac{363}{365} \times \dots \times \frac{365 - (N-1)}{365} \right]$$

$$1 - \left[\frac{1}{365^N} \prod_{i=0}^{N-1} (365 - i) \right]$$

This is the probability that there is **at least 2 people** with the same birthday in a group of **N people**



Birthday Paradox

Let's assume there are **N people** in a given room. What is the probability that **at least 2 people** share the same birthday **with 50% probability**?

$$0.5 = 1 - \left[\frac{1}{365^N} \prod_{i=0}^{N-1} (365 - i) \right]$$

It turns out to be that **N=23**.

This means that even with a small group of people, it is quite common for **at least 2 people to share the same birthday**



Collision Attack Against MD5

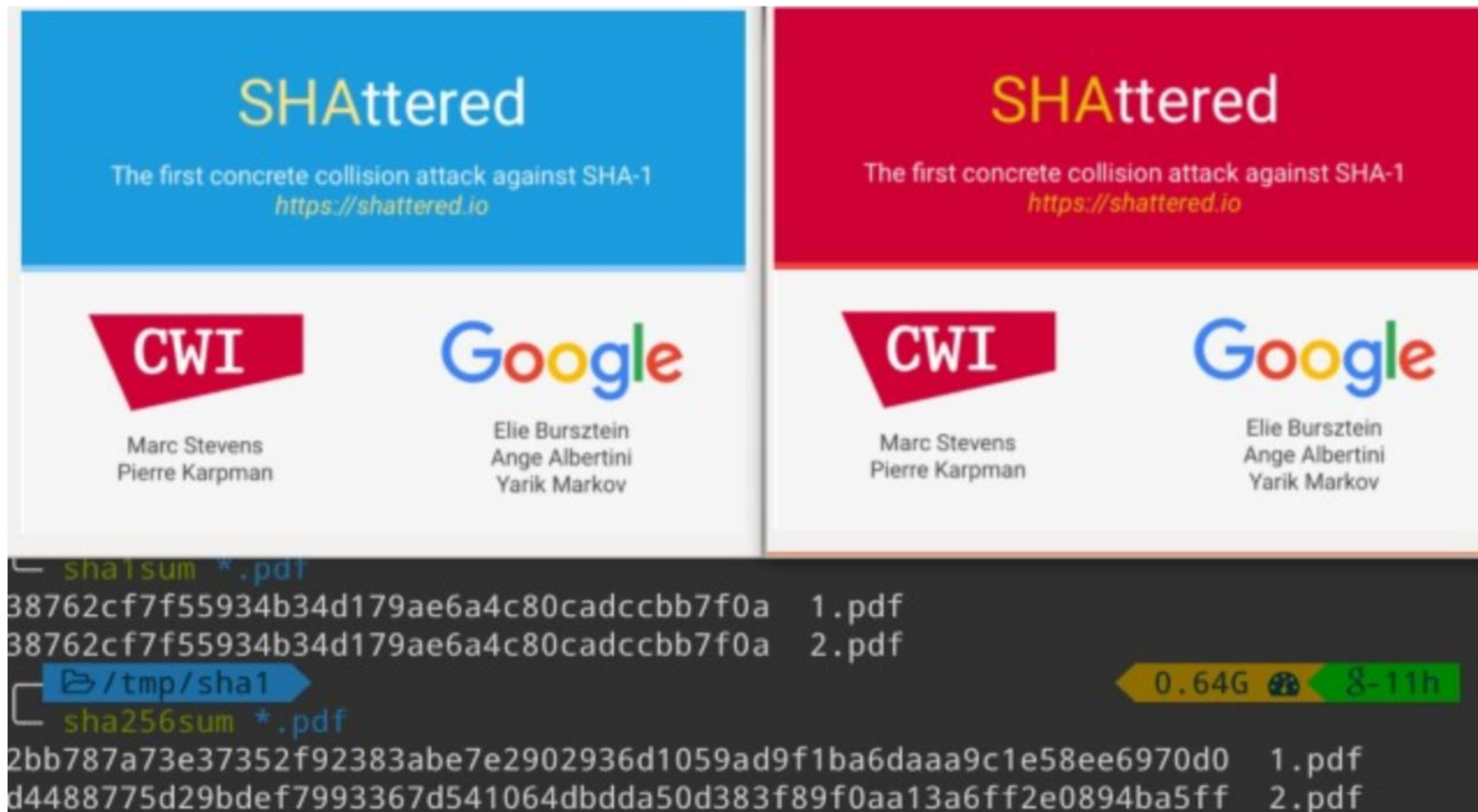
- 2004: MD5's collision-resistance property was broken
- By Xiaoyun Wang

Sequence #1															
d1	31	dd	02	c5	e6	ee	c4	69	3d	9a	06	98	af	f9	5c
2f	ca	b5	87	12	46	7e	ab	40	04	58	3e	b8	fb	7f	89
55	ad	34	06	09	f4	b3	02	83	e4	88	83	25	71	41	5a
08	51	25	e8	f7	cd	c9	9f	d9	1d	bd	f2	80	37	3c	5b
d8	82	3e	31	56	34	8f	5b	ae	6d	ac	d4	36	c9	19	c6
dd	53	e2	b4	87	da	03	fd	02	39	63	06	d2	48	cd	a0
e9	9f	33	42	0f	57	7e	e8	ce	54	b6	70	80	a8	0d	1e
c6	98	21	bc	b6	a8	83	93	96	f9	65	2b	6f	f7	2a	70
Sequence #2															
d1	31	dd	02	c5	e6	ee	c4	69	3d	9a	06	98	af	f9	5c
2f	ca	b5	07	12	46	7e	ab	40	04	58	3e	b8	fb	7f	89
55	ad	34	06	09	f4	b3	02	83	e4	88	83	25	f1	41	5a
08	51	25	e8	f7	cd	c9	9f	d9	1d	bd	72	80	37	3c	5b
d8	82	3e	31	56	34	8f	5b	ae	6d	ac	d4	36	c9	19	c6
dd	53	e2	34	87	da	03	fd	02	39	63	06	d2	48	cd	a0
e9	9f	33	42	0f	57	7e	e8	ce	54	b6	70	80	28	0d	1e
c6	98	21	bc	b6	a8	83	93	96	f9	65	ab	6f	f7	2a	70
Both produce MD5 digest 79054025255fb1a26e4bc422aef54eb4															

These two messages are different, but they generate the same MD5 output

Collision Attack Against SHA-1

- 2017: SHA-1's collision-resistance property was broken
- CWI Amsterdam and Google



They managed to generate **two different PDF files**, one in blue and the other in red color, which produce the same hash output.



Collision Attack Against SHA-1

- 2017: SHA-1's collision-resistance property was broken
- CWI Amsterdam and Google

SHattered
The first concrete collision attack against SHA-1
<https://shattered.io>

CWI
Marc Stevens
Pierre Karpman


Google
Elie Bursztein
Ange Albertini
Yarik Markov

```
shasum *.pdf
38762cf7f55934b34d179ae6a4c80cadccbb7f0a 1.pdf
38762cf7f55934b34d179ae6a4c80cadccbb7f0a 2.pdf
/tmp/sha1
sha256sum *.pdf
2bb787a73e37352f92383abe7e2902936d1059ad9f1ba6daaa9c1e58ee6970d0 1.pdf
d4488775d29bdef7993367d541064dbdda50d383f89f0aa13a6ff2e0894ba5ff 2.pdf
```

Compared to other collision attacks:

 **MD5**
1 smartphone
30 sec

 **SHA-1 Shattered**
110 GPU
1 year

 **SHA-1 Bruteforce**
12,000,000 GPU
1 year



MD5 Collision Attack

- **Generating Two Different Files with the Same MD5 Hash**

We will generate two different files with **the same MD5 hash values**.

We will use the **md5collgen** program.

It takes as an input a prefix file with any arbitrary content and generates two output files, out1.bin and out2.bin.



The original project: <https://www.win.tue.nl/hashclash/>

MD5 Collision Attack

```
(kali㉿kali)-[/media/sf_SecurityCCT4/md5]
$ echo hello > prefix

(kali㉿kali)-[/media/sf_SecurityCCT4/md5]
$ sudo md5collgen prefix
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'msg1.bin' and 'msg2.bin'
Using prefixfile: 'prefix'
Using initial value: 91e1e88548a4bef54d09a7c23bd03682

Generating first block: .....
Generating second block: S01.
Running time: 3.75114 s

(kali㉿kali)-[/media/sf_SecurityCCT4/md5]
$ md5sum msg1.bin
39845ebb5197394a43f12e029afaa5a2  msg1.bin

(kali㉿kali)-[/media/sf_SecurityCCT4/md5]
$ md5sum msg2.bin
39845ebb5197394a43f12e029afaa5a2  msg2.bin
```



Why collision attack matters?

- **Replace a legitimate file with a malicious one:** The malicious file's hash would match the expected hash, bypassing integrity checks.
- **Bypass authentication:** An attacker could leverage a collision attack to craft a different input (password) that produces the same hash as a legitimate user's password.
- **Sign malicious software with a forged certificate:** By creating a collision with a trusted certificate, they could trick systems into accepting the malware as legitimate.
- **Forge a digital signature:** Create a document with the same hash as a legitimate one, making it appear signed by the same person.
- **Flood systems with colliding data:** Overwhelm systems with data that hashes to the same value, causing denial-of-service.



Cracking Passwords



Cracking Passwords

Why should you choose passwords that are:

- Long
- Contains both numbers and letters
- Not being used in many places

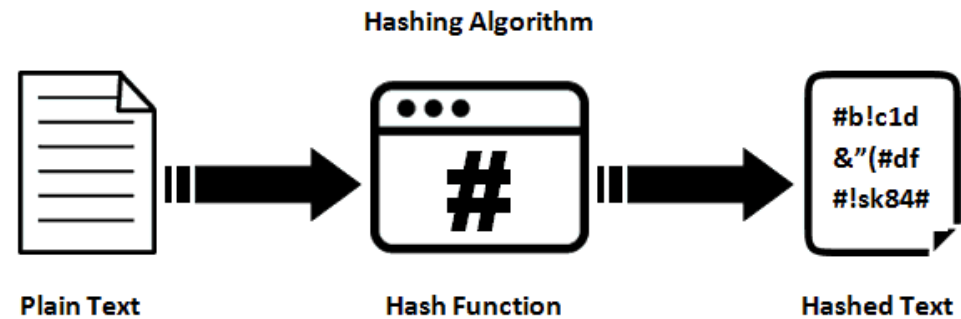
Why should there be a limit on the number of login attempts?

Why should we use 2-factor authentication?

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	1 sec	5 secs
7	Instantly	Instantly	25 secs	1 min	6 mins
8	Instantly	5 secs	22 mins	1 hour	8 hours
9	Instantly	2 mins	19 hours	3 days	3 weeks
10	Instantly	58 mins	1 month	7 months	5 years
11	2 secs	1 day	5 years	41 years	400 years
12	25 secs	3 weeks	300 years	2k years	34k years
13	4 mins	1 year	16k years	100k years	2m years
14	41 mins	51 years	800k years	9m years	200m years
15	6 hours	1k years	43m years	600m years	15bn years
16	2 days	34k years	2bn years	37bn years	1tn years
17	4 weeks	800k years	100bn years	2tn years	93tn years
18	9 months	23m years	61tm years	100tn years	7qd years

<https://www.hivesystems.io/blog/are-your-passwords-in-the-green>

Why doesn't Facebook have my password?

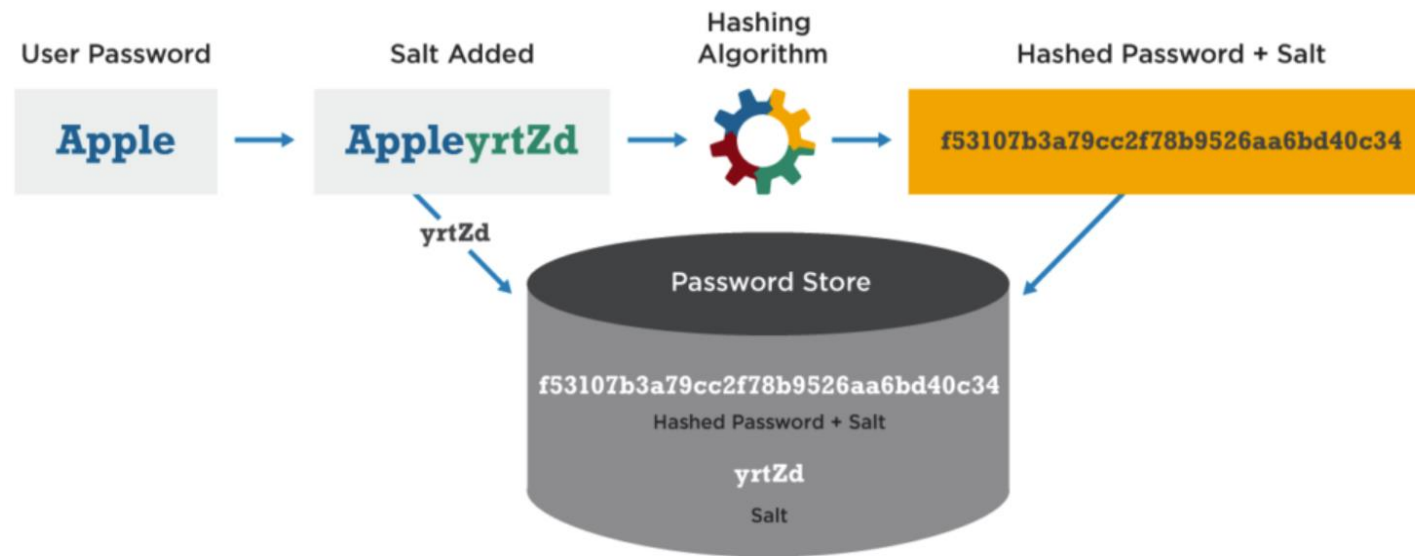


User	Password	User	Password Hash
Stephen	auhsoJ	Stephen	39e717cd3f5c4be78d97090c69f4e655
Lisa	hsifdrowS	Lisa	f567c40623df407ba980bfad6dff5982
James	1010NO1Z	James	711f1f88006a48859616c3a5cbcc0377
Harry	sinocarD tupaC	Harry	fb74376102a049b9a7c5529784763c53
Sarah	auhsoJ	Sarah	39e717cd3f5c4be78d97090c69f4e655



Adding Salt to Hashing

- Have you heard of salting?

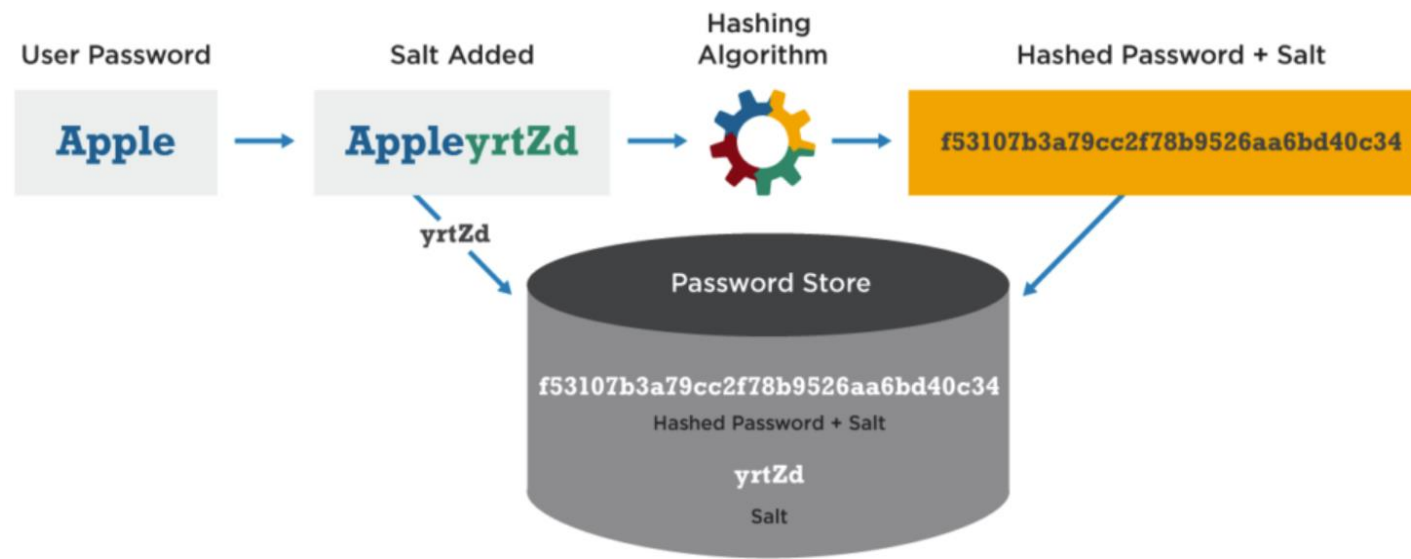


User	Random Salt	Password Hash
Stephen	06917d7ed65c466fa180a6fb62313ab9	b65578786e544b6da70c3a9856cdb750
Lisa	51f2e43105164729bb46e7f20091adf8	2964e639aa7d457c8ec0358756cbffd9
James	fea659115b7541479c1f956a59f7ad2f	dd9e4cd20f134dda87f6ac771c48616f
Harry	30ebf72072134f1bb40faa8949db6e85	204767673a8d4fa9a7542ebc3eceb3a2
Sarah	711f51082ea84d949f6e3efecf29f270	e3afb27d59a34782b6b4baa0c37e2958



Adding Salt to Hashing

- The salt is a randomly generated string for each password, which is not secret.
- It is combined with the password and added to the hashing process to force the uniqueness of the output hash.



Cracking Passwords

1. Brute-force or dictionary attack

- It means using automated software that systematically checks all possible passwords until the correct one is found.
- It's trial-and-error process in which the hacker computes the hash of each word in a dictionary or a word list and then compares the resulting hash to the hash of the password.

2. Rainbow tables

- These are precomputed tables used for reversing cryptographic hash functions, usually used for cracking password hashes.
- The hacker has the hash of the password she wants to crack and searches for that hash in the list of precomputed hashes of the rainbow table.

Space vs Time: A brute force attack takes a lot of time but that doesn't require a lot of storage. A rainbow table requires a lot of space, many tens or hundreds of GB, but it doesn't require a lot of time.



Authentication in linux: /etc/passwd vs /etc/shadow

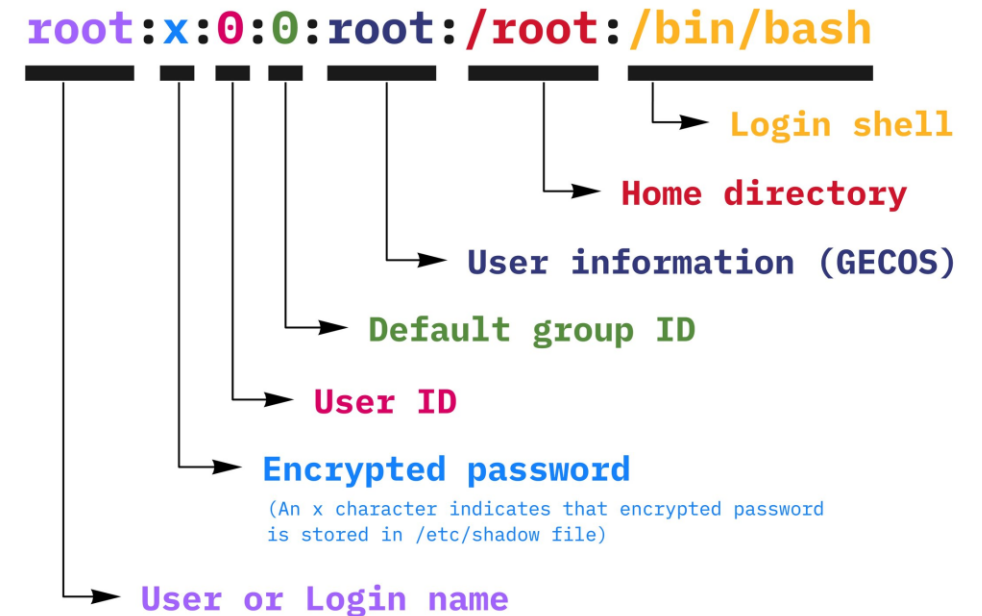
- **/etc/passwd**: all valid user accounts
- **/etc/shadow**: requires **root privileges** to read and write the file

-> Can be used to crack the password.



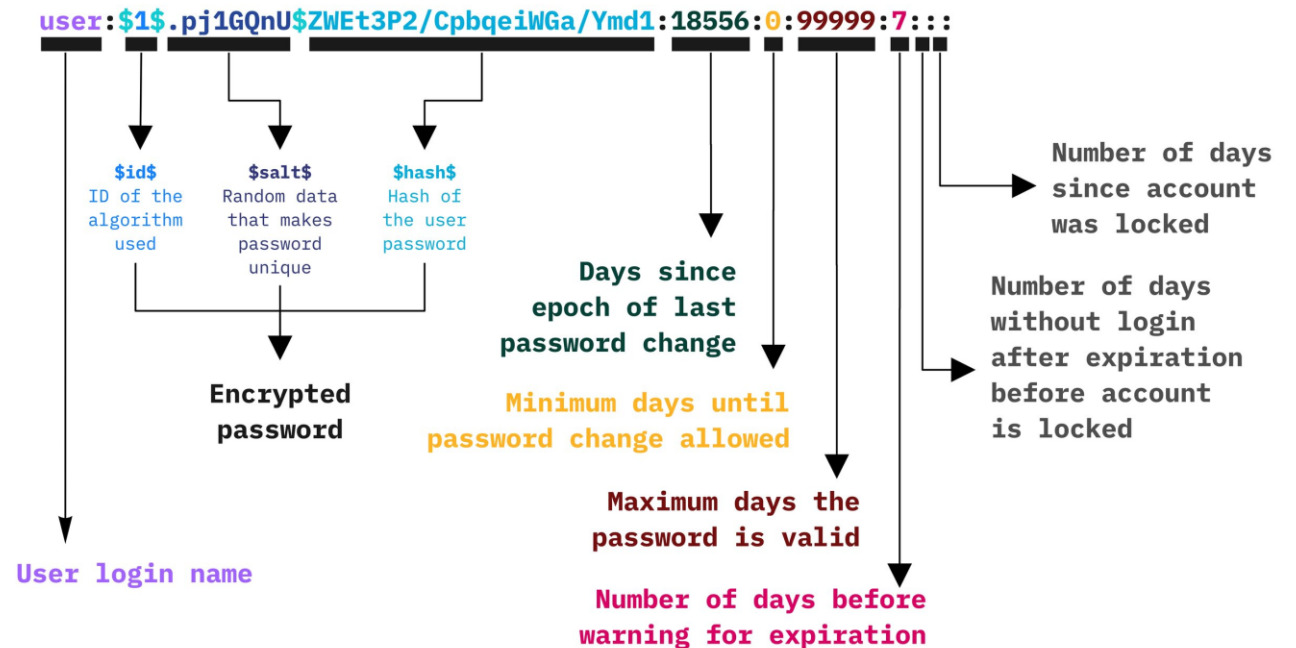
/etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
```



/etc/shadow

```
[root@arch01 ~]# cat /etc/shadow | sed 's/michael/test/' | sed 's/mbo/joe/'
root:$6$4GxAA08J$AB7vFkLSCxtVdVMcPav8jZ5u4ZsyG22hy1cqWPdp0oel84Ves7NOYEXSubfwkbt
UeHNxYwjJUGe8U/sjITBhq/:16672::::::
bin:x:14871::::::
daemon:x:14871::::::
mail:x:14871::::::
ftp:x:14871::::::
http:x:14871::::::
uidd:x:14871::::::
dbus:x:14871::::::
nobody:x:14871::::::
systemd-journal-gateway:x:14871::::::
systemd-timesync:x:14871::::::
systemd-network:x:14871::::::
systemd-bus-proxy:x:14871::::::
systemd-resolve:x:14871::::::
systemd-journal-upload:!!:16672::::::
systemd-journal-remote:!!:16672::::::
avahi:!!:16672::::::
polkitd:!:16672:0:99999:7:::
joe:$6$TA4PslzF$ch961z/ppk1VrmVAqSjSEdf75FIahttselx/bsDd
KGvYV1xkohA37aeEvmu8d1:16672:0:99999:7:::
git:!:16683::::::
test:$6$PNkLwU7L$2Hm8YRMGgRoxxt4srAzGBZJFfxU7SnlDbauWb6APg5dyXSiQvQwSxHY1j0i5t2eM
kZ1PwBzY1aHAVZu29wSBpJ0:16735:0:99999:7:::
```



For more info about the shadow file, you can run the command: **man shadow**



John the Ripper (JTR)

1. Single crack mode.

- It uses the login names together with other fields from the passwd file, also with a large set of mangling rules applied. This is the fastest cracking mode and is applicable to very simple passwords.

2. Dictionary Attack

- In this mode you need to supply a dictionary file that contains one word per line and a password file. You can enable word mangling rules which are used to modify or "mangle" words producing other likely passwords.

3. "Incremental" mode.

- This is the most powerful cracking mode because it will try all possible character combinations as passwords. If you supply a random password with length of more than 12-14 chars will never terminate and you'll have to interrupt it manually.



John the Ripper (JtR) – brute force offline

- JtR operates mainly on **hashed passwords** stored in files on your local system.

It is already installed in Kali Linux:

`apt install john`

```
└─$ sudo apt install john
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
john is already the newest version (1.9.0-Jumbo-1+git20211102-0kali7).
The following packages were automatically installed and are no longer required:
  cython3 debtags kali-debtags libjavascriptcoregtk-4.0-18 libperl5.36 libqt5multimedia5
  libqt5multimedia5-plugins libqt5multimediasgsttools5 libqt5multimediawidgets5 librtlsdr0 libucl1
  libwebkit2gtk-4.0-37 libzxing2 perl-modules-5.36 python3-backcall python3-debian python3-future
  python3-pickleshare python3-requests-toolbelt python3-rfc3986 python3-unicodedsv
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 30 not upgraded.
```



John the Ripper (JTR)

- The first step is to **combine** the provided **password and shadow files** into a single file.

`unshadow /etc/passwd /etc/shadow > unshadowed.txt`

```
$ sudo unshadow /etc/passwd /etc/shadow > unshadowed.txt
Created directory: /root/.john
```

- Check the content of the combined file

`cat unshadowed.txt`

- Run John

`john -single --forma=crypt unshadowed.txt`

```
# john -single --forma=crypt unshadowed.txt
Using default input encoding: UTF-8
Loaded 1 password hash (crypt, generic crypt(3) [?/64])
Cost 1 (algorithm [1:descrypt 2:md5crypt 3:sunmd5 4:bcry
hes
Cost 2 (algorithm specific iterations) is 1 for all load
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for s
Warning: Only 94 candidates buffered for the current sal
kali (kali) ←
1g 0:00:00:00 DONE (2024-02-12 19:05) 2.631g/s 247.3p/s
Use the "--show" option to display all of the cracked pa
Session completed.
```

- Run John

`john --show unshadowed.txt`

```
# john --show unshadowed.txt
kali:kali:1000:1000:,,,:/home/kali:/usr/bin/zsh

1 password hash cracked, 0 left
```

John the Ripper (JTR) – dictionary

- John the Ripper comes with its own password list in `/usr/share/john`
- On Kali Linux there are more dictionary files in `/usr/share/metasploit-framework/data/wordlists`
- The first step is to **combine** the provided **password and shadow files** into a single file.
`john --wordlist=/usr/share/john/password.lst --rules --forma=crypt unshadowed.txt`
- Run John
`john -show unshadowed.txt`

Other tools for Brute-force attacks or dictionary attacks

- Automated tools: **Hydra, Medusa**
- Primarily targets **online services** like SSH, FTP, HTTP, and more.
- But we need to know the username (or try to guess it)
- And then we have to choose a password list - and we choose the John The Ripper list.

Example: **Hydra -l admin -P /usr/share/john/password.lst -vV ftplogin.com ftp**

Cracking Passwords Countermeasures

- Use **strong passwords** that consist of at least 12 random characters including both lower and uppercase letters, digits and special characters.
- **Do not use dictionary words** including combinations of these words no matter the language.
- **Do not store passwords unencrypted** like for example in word files. Do not write them down!
- **Do not reuse your passwords!** Use a unique password for each website or service.
- Additionally, **setup 2-way authentication** for important websites like your bank, paypal or even Google or Facebook accounts



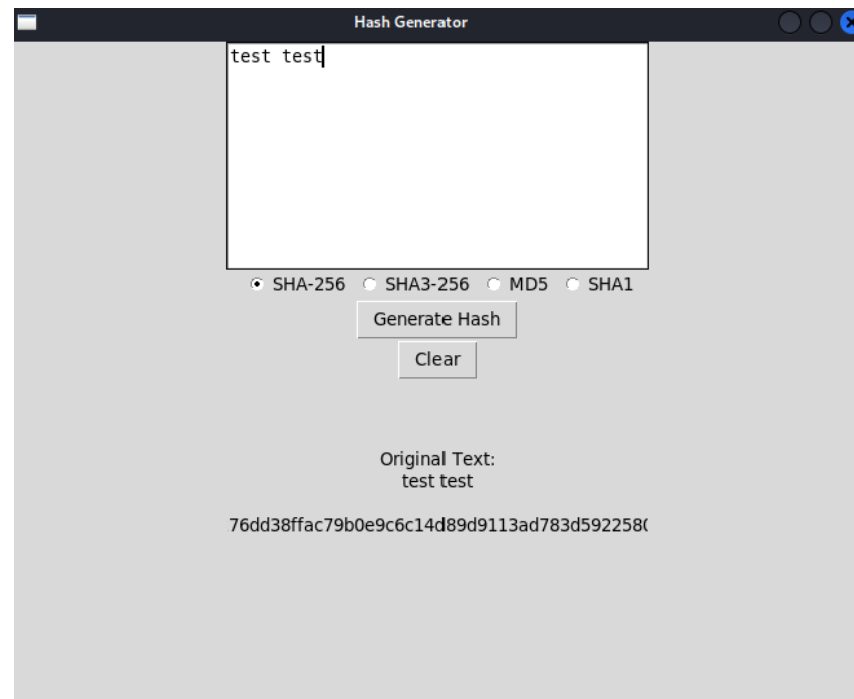
Summary

- Hash functions
- Properties of the cryptographic hash functions
- Collision attack in hashing
- Adding Salt to passwords
- Tools for password cracking



Assignment 1

- In your favorite programming language, create a simple hash generator similar to the one shown in this figure



Assignment 2

- On Kali add a new user called **admin** and set an easy-to-guess password, e.g., **secret**
- Using John the Ripper try to crack the user's password using this wordlist: `/usr/share/metasploit-framework/data/wordlists/unix_passwords.txt`
- Hint: Find the linux command to create a new user, and set its password.



Feedback

<https://www.menti.com/albj43xe1v1r>

Code: 3323 7155



AALBORG UNIVERSITET