UNIVERSITÀ
DEGLI STUDI
DI PADOVA

J. Pazzini
PADOVA UNIVERSITY, INFN

3 - RELIABILITY AND SECURITY

Management and Analysis of Physics Datasets - Module B

Physics of Data

A.A. 2023/2024

STORAGE RELIABILITY

Preserving stored data is extremely important, and we must keep it safe from:

- Data loss
- Data corruption
- (Errors)

STORAGE RELIABILITY

Preserving stored data is extremely important, and we must keep it safe from:

- Data loss
- Data corruption
- (Errors)

Data loss due to storage failure

→ disk HW failure (electrical/mechanical/…)

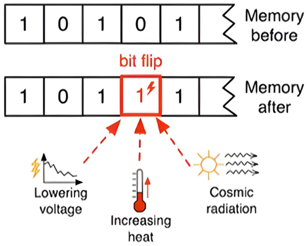
⇒ all data from the disk is usually lost

Data corruption

→ single-bit errors (1 ↔ 0) over a number of read-write IO

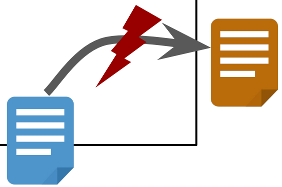
(for HDDs, ~1b error in 10^{14} bits read/written, assuming ~1GB files)

⇒ 1 file corrupted per 10,000 files written



Software / data-transfer errors

→ e.g. not performing consistency checks when copying or transferring data

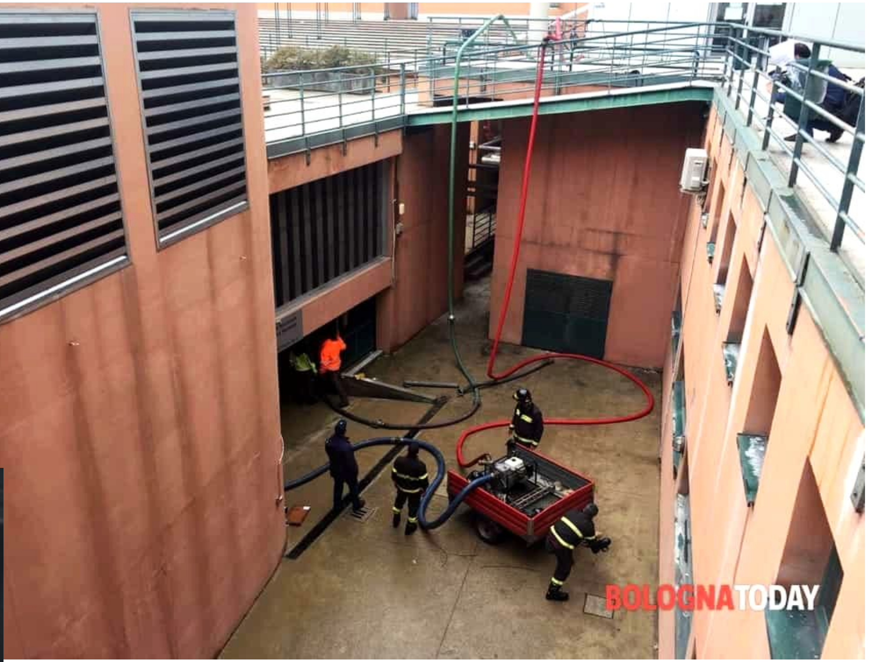


RELIABILITY & LONG-TERM PRESERVATION

Bologna CNAF computing center

- Hosting site for CERN experiments' data
- ~40PB disks + 90PB tape
- Both data-storage & data-processing site
- Temperature / humidity / fire control
- Tons of control systems

RELIABILITY & LONG-TERM PRESERVATION



s*#t may still happen, no matter what…

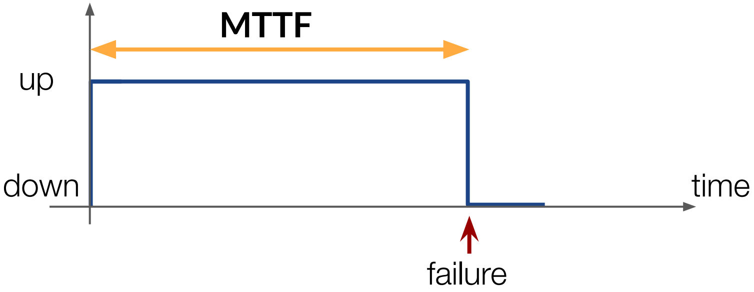
STORAGE RELIABILITY

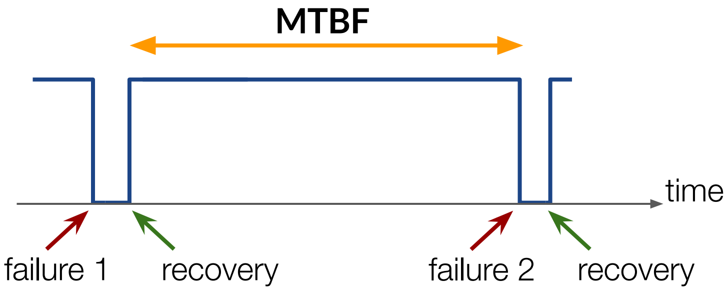
Preserving stored data is extremely important, and we must keep it safe from:

- Data loss
- Data corruption
- Errors

Reliability is defined as the probability that a system will continue to perform correctly after a given time / number of operations

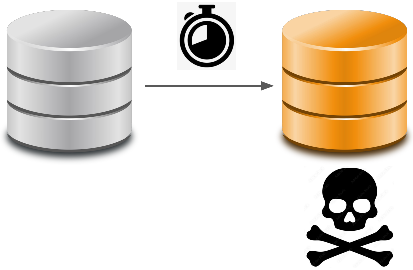
This can be measured by the **Mean Time To Failure (MTTF)** or **Mean Time Between Failure (MTBF)**



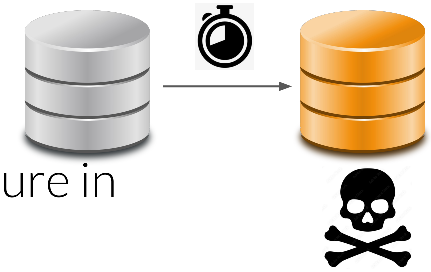


STORAGE RELIABILITY

A realistic MTTF of a single HDD is ~ 10 years
(although manufacturers claim much more, but tons of caveats do apply)



STORAGE RELIABILITY



A realistic MTTF of a single HDD is ~ 10 years
(although manufacturers claim much more, but tons of caveats do apply)

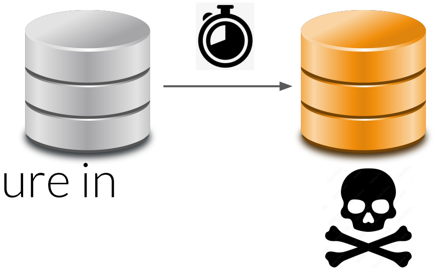
Assuming a flat prior on the probability of a single HDD to incur in a failure in any given day, we expect:

$$p = 1 / (10 \text{ years} * 365) = 0.00027$$

What could this translate if we had N=5,000 HDDs instead of 1?
How many HDDs would we expect to fail per day?

STORAGE RELIABILITY

A realistic MTTF of a single HDD is ~ 10 years
(although manufacturers claim much more, but tons of caveats do apply)



Assuming a flat prior on the probability of a single HDD to incur in a failure in any given day, we expect:

$$p = 1 / (10 \text{ years} * 365) = 0.00027$$

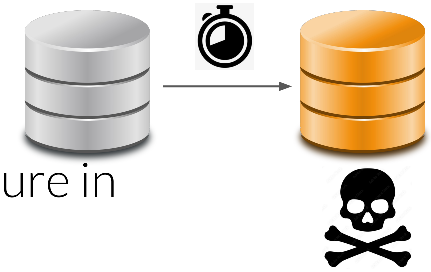
What could this translate if we had N=5,000 HDDs instead of 1?
How many HDDs would we expect to fail per day?

Assuming all disks independent, the expected number of disk failures per day, according to the binomial distribution

$$E[x] \approx N\, p = 5000 \times 0.00027 = 1.35$$

STORAGE RELIABILITY

A realistic MTTF of a single HDD is ~ 10 years
(although manufacturers claim much more, but tons of caveats do apply)



Assuming a flat prior on the probability of a single HDD to incur in a failure in any given day, we expect:

$$p = 1 \, / \, (10 \text{ years} * 365) = 0.00027$$

What could this translate if we had N=5,000 HDDs instead of 1?
How many HDDs would we expect to fail per day?

Assuming all disks independent, the expected number of disk failures per day, according to the binomial distribution

$$\mathrm{E}[x] \approx N\, p = 5000 \times 0.00027 = 1.35$$

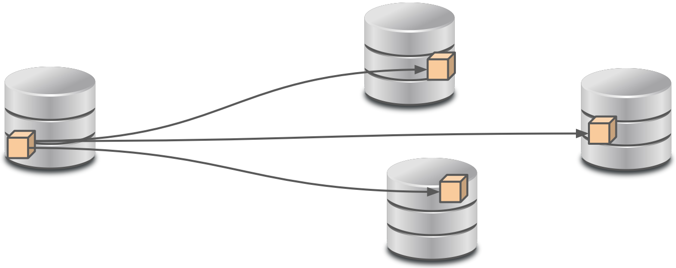
At CERN, with N~100,000 HDDs this would translate to 27 expected HDD failing every single day!

RELIABILITY STRATEGIES

Dataset storage reliability can be improved with a number of strategies

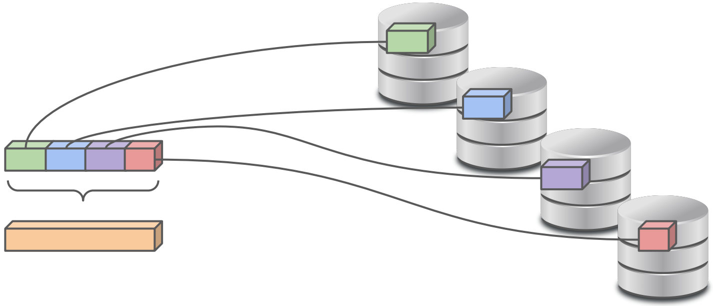
Mirroring

Replicate data across multiple storage elements (and when possible even on different sites!)



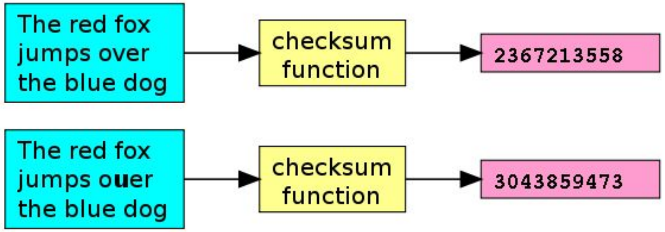
Striping

Subdivide data in “pieces” and scatter it across multiple storage elements



Checksum / Parity checks

Ensure data consistency by detecting (and possibly correcting) errors



The red fox jumps over the blue dog

checksum function

2367213558

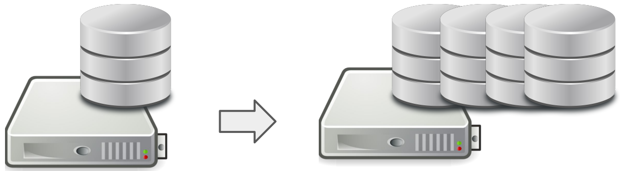
The red fox jumps ouer the blue dog

checksum function

3043859473

RAID

Improving storage reliability with **disk redundancy**



RAID → Redundant Array of Independent Disks

(originally was *Inexpensive*, nowadays not so much…)

Multiple *physical* devices are combined into a single *logical* one to improve reliability and/or performance

Can be implemented in SW (e.g. using `mdadm` in linux), but mostly using a HW controller board to daisy-chain multiple Disks

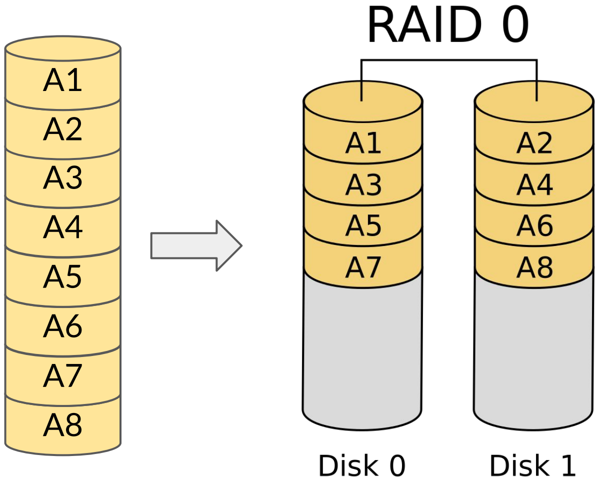
A number of RAID schemas (RAID levels) can be implemented:

- RAID 0
- RAID 1
- RAID 1+0
- RAID 5
- RAID 6
- many others…

RAID LEVELS

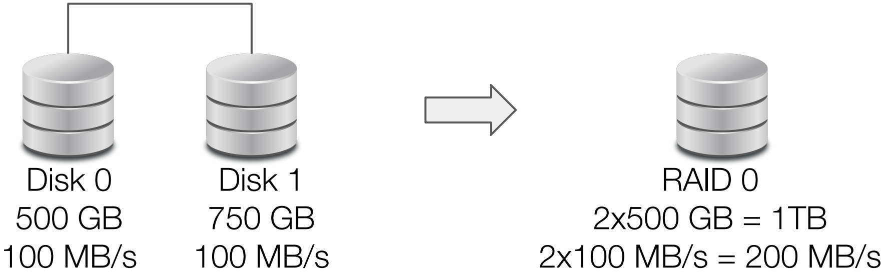
RAID 0 → Disk striping, no mirroring

- Use 2+ disks to store the data
- Data is striped across all the disks
- No data mirroring (i.e. no data replication)



When combining 2 disks, the overall storage capacity will be 2x the capacity of the smallest one

Used mostly for performance rather than for reliability
→ Can read/write at the same time from all disks

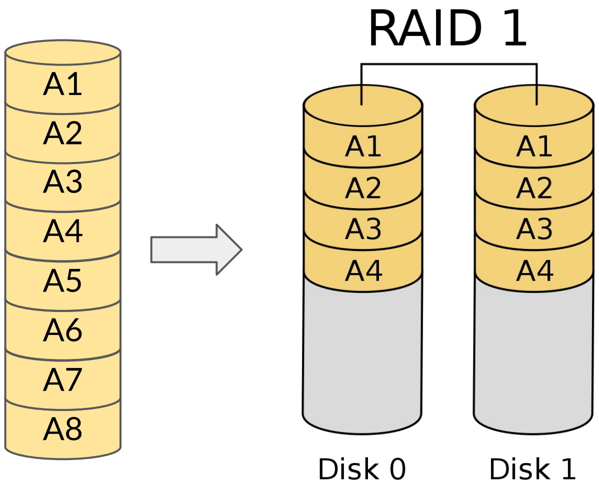


Losing 1 disk will mean losing ALL data!

RAID LEVELS

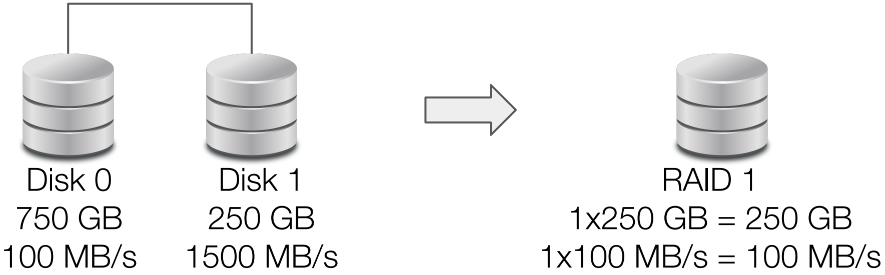
RAID 1 → Disk mirroring, no striping

- Use 2+ disks to store the data
- Data is mirrored across all the disks
- No data striping (i.e. no data subdivision)



When combining 2 disks, the overall storage capacity will be 1x the capacity of the smallest one

Used for reliability, but usually bad for performance
→ Write at the slowest disk throughput

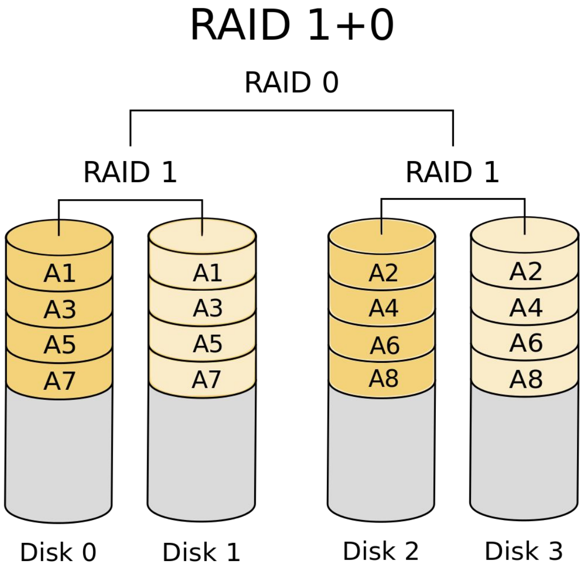


Data is preserved as long as at least 1 disk is functional

RAID LEVELS

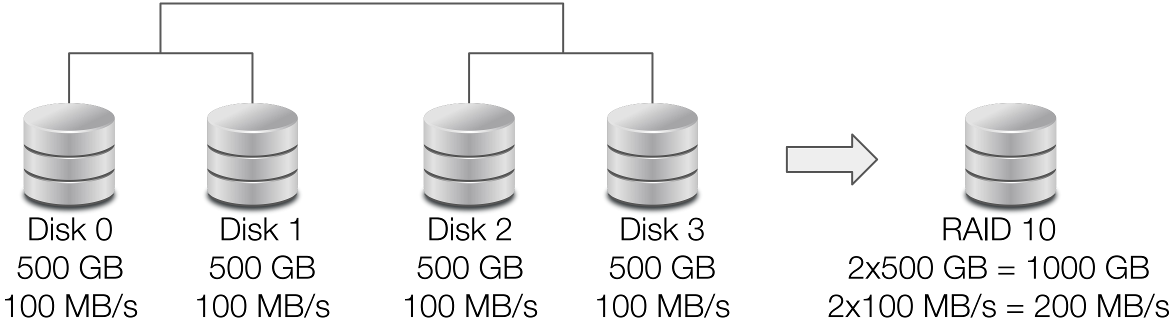
RAID 1+0 (or RAID 10) → Disk mirroring plus striping

- Use 4+ disks to store the data
- Data is mirrored (RAID 1) across 2+ disks
- And striped (RAID 0) across 2+ RAID 1 disks



Good overall performance (close to RAID 0) but improved reliability (from RAID 1 mirroring):

- Read is fast due to striping
- Write is slower due to mirroring



Data is preserved as long as at least 1 disk is functional in each mirrored pair

RAID WITH PARITY

A **parity** information can be added to data blocks as an error protection scheme
→ usually based on **XOR logic**

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Disk 1	Disk 2	Disk 3	D1 ⊕ D2 ⊕ D3	Parity
0	0	1		
0	1	1		
1	1	1		
0	1	0		
1	0	0		
0	0	0		
0	1	1		
1	1	1		

RAID WITH PARITY

A **parity** information can be added to data blocks as an error protection scheme
→ usually based on **XOR logic**

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Disk 1	Disk 2	Disk 3	D1 ⊕ D2 ⊕ D3	Parity
0	0	1	1	1
0	1	1	0	0
1	1	1	1	1
0	1	0	1	1
1	0	0	1	1
0	0	0	0	0
0	1	1	0	0
1	1	1	1	1

RAID WITH PARITY

A **parity** information can be added to data blocks as an error protection scheme
→ usually based on **XOR logic**

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Disk 1	Disk 2	Disk 3	D1 ⊕ D2 ⊕ D~~3~~	Parity
0	0		0	1
0	1		1	0
1	1		0	1
0	1		1	1
1	0		1	1
0	0		0	0
0	1		1	0
1	1		0	1

In case of a SINGLE disk failure, data can be recovered from a single parity bit information

RAID WITH PARITY

A **parity** information can be added to data blocks as an error protection scheme
→ usually based on **XOR logic**

Disk 1	Disk 2	Disk 3	D1 ⊕ D2 ⊕ D3	Parity
0	0	1	1	1
0	1	1	0	0
1	1	1	1	1
0	1	0	1	1
1	0	0	1	1
0	0	0	0	0
0	1	1	0	0
1	1	1	1	1

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

In case of a SINGLE disk failure, data can be recovered from a single parity bit information

RAID WITH PARITY

A **parity** information can be added to data blocks as an error protection scheme → usually based on **XOR logic**

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Disk 1	Disk 2	Disk 3	D1 ⊕ D2 ⊕ D3	Parity
		1	1	1
		1	1	0
		1	1	1
		0	0	1
		0	0	1
		0	0	0
		1	1	0
		1	1	1

In case of a DOUBLE(+) disk failure, data CAN NOT BE RECOVERED from a single parity bit information

RAID WITH PARITY

A **parity** information can be added to data blocks as an error protection scheme → usually based on **XOR logic**

1	1	or	0	0	?

Disk 1	Disk 2	Disk 3	D1 ⊕ D2 ⊕ D3	Parity
		1	1	1
		1	1	0
		1	1	1
		0	0	1
		0	0	1
		0	0	0
		1	1	0
		1	1	1

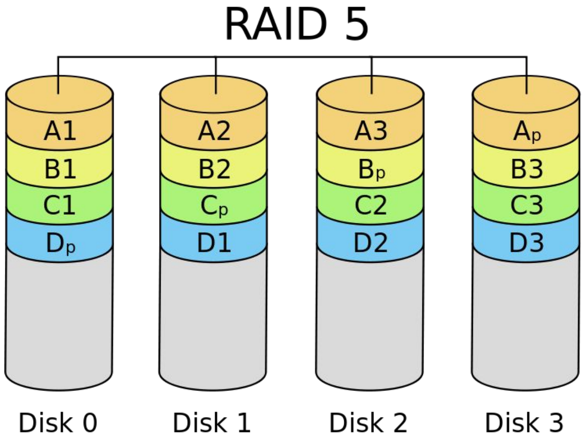
A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

In case of a DOUBLE(+) disk failure, data CAN NOT BE RECOVERED from a single parity bit information

RAID LEVELS WITH PARITY

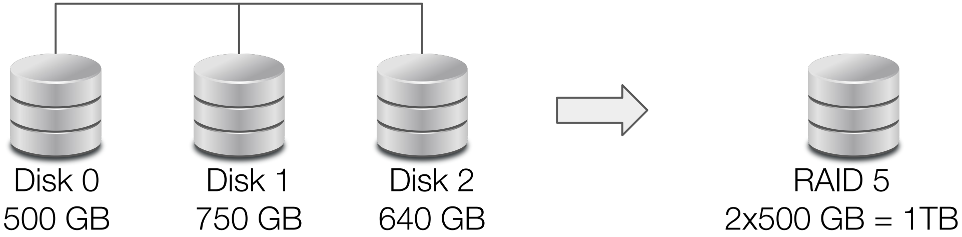
RAID 5 → Disk striping, no mirroring, with distributed parity

- Use 2+ disks to store the data + 1 to store parity
- Data is striped across all the disks
- No data mirroring (i.e. no data replication)
- Block parity is distributed across all disks so all disks can participate in satisfying read requests



Similar to RAID 0 (for *n-1* disks), with good overall performances (throughput calculation not really trivial due to write/read parity information)

Single disk loss / single data error is now recoverable

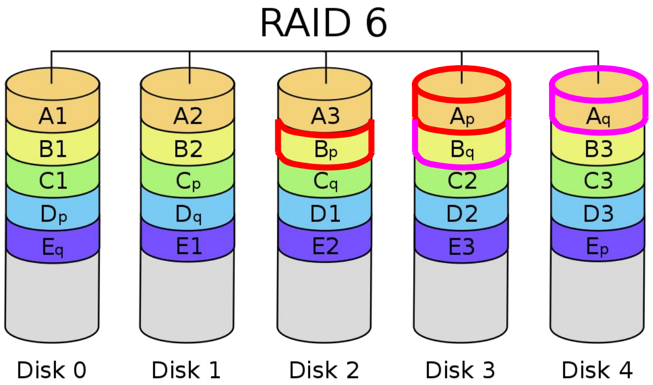


Up to 1 disk loss
Up to 1 block error recovery

RAID LEVELS WITH PARITY

RAID 6 → Disk striping, no mirroring, with distributed *double* parity

- Use 2+ disks to store the data + 2 to store parity
- Data is striped across all the disks
- No data mirroring (i.e. no data replication)
- Two parity infos are distributed across all disks



Extend over the RAID 5 to enable data recovery even in case of double error or 2 disk loss

The 2 independent parity information are typically evaluated by error-correcting codes such as the Reed–Solomon codes

$$\mathbf{p = D1 \oplus D2 \oplus D3}$$
$$\mathbf{q = D1 \oplus (D2 \gg shift) \oplus (D3 \gg shift^2)}$$

Up to 2 disks loss
Up to 2 block errors recovery

SECURITY IN DATA MANAGEMENT

Security is a key (often underrated) aspect of dataset management

- Certify users' identity
- Define what users can/cannot access
- Ensure data integrity over transactions
- …

Cryptography is at the basis of all modern security techniques

cryp·tog·ra·phy | \ krip-ˈtä-grə-fē

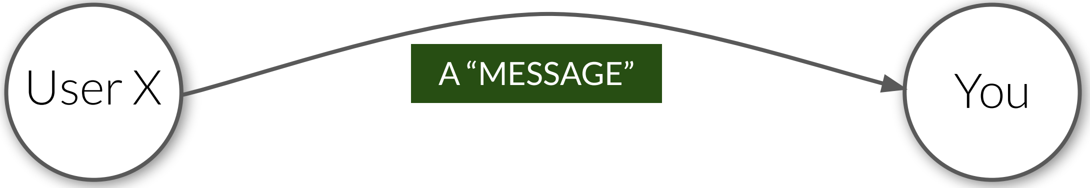
The discipline that embodies the principles, means, and methods for the transformation of data in order to hide their semantic content, prevent their unauthorized use, or prevent their undetected modification.

VISUALIZING SECURITY

Security applies to almost every aspect of computing:

- protecting data stored on a device
- establishing secure connections to remote services
- exchanging information across two peers
- preventing malicious users from tampering data
- …

For the sake of simplicity we can visualize the main security concepts with the idea of an information exchange between two endpoints (a “conversation” of sort)

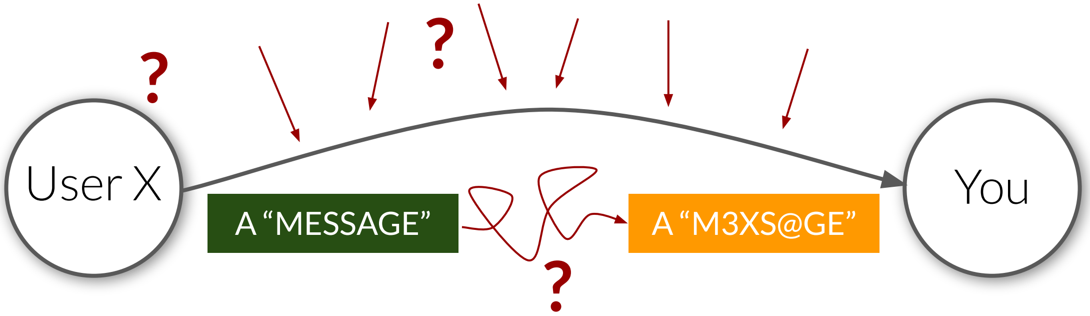


VISUALIZING SECURITY

Security applies to almost every aspect of computing:

- protecting data stored on a device
- establishing secure connections to remote services
- exchanging information across two peers
- preventing malicious users from tampering data
- …

For the sake of simplicity we can visualize the main security concepts with the idea of an information exchange between two endpoints (a "conversation" of sort)



- Am I sure who am I talking to?
- Can anybody else listening to the conversation intercept the data?
- Is the message I'm receiving identical to the one sent?
- Is the other user responsible for the message (s)he's sent me?

CRYPTOGRAPHY IN DATA MANAGEMENT

Confidentiality

⟶ Ensure that nobody can extract knowledge of what you transfer, even if listening the whole conversation

Integrity

⟶ Ensure that message has not been modified during the transmission

Authenticity, Identity, Non-repudiation

⟶ You can verify that you are talking to the entity you think you are talking to

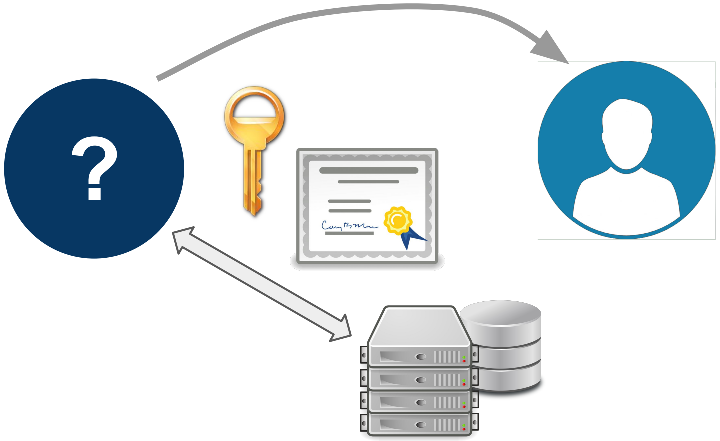
⟶ You can verify who is the specific individual behind that entity

⟶ The individual behind that asset cannot deny being associated with it

AUTHENTICATION & AUTHORIZATION

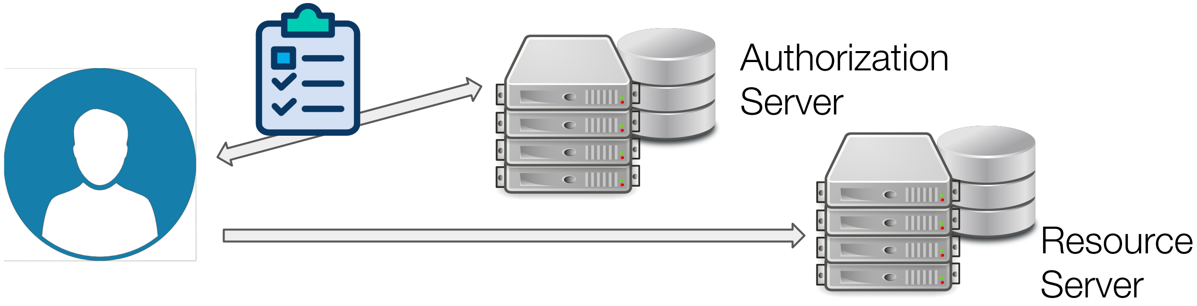
Authentication → identifying users and validating their identity

- Public/private keys
- Certificates
- Single Sign-On (SSO)
- Custom authentication (e.g.: username/password)



Authorization → granting a user the permission to access/use specific resources

- Authorization Authorities via Access Control Lists



ENCODING/DECODING INFORMATION

- A simple example is passing a message ***M*** of length ***L*** between two users
- Both users also exchange a secret key ***K*** of the same length ***L*** to scramble (cypher) and reassemble (decypher) the data
- A simple **XOR** logic function can be used to encode-decode the information
  (once again, using XOR only to simplify the visualization of the topic)

USER A

USER B

Original message

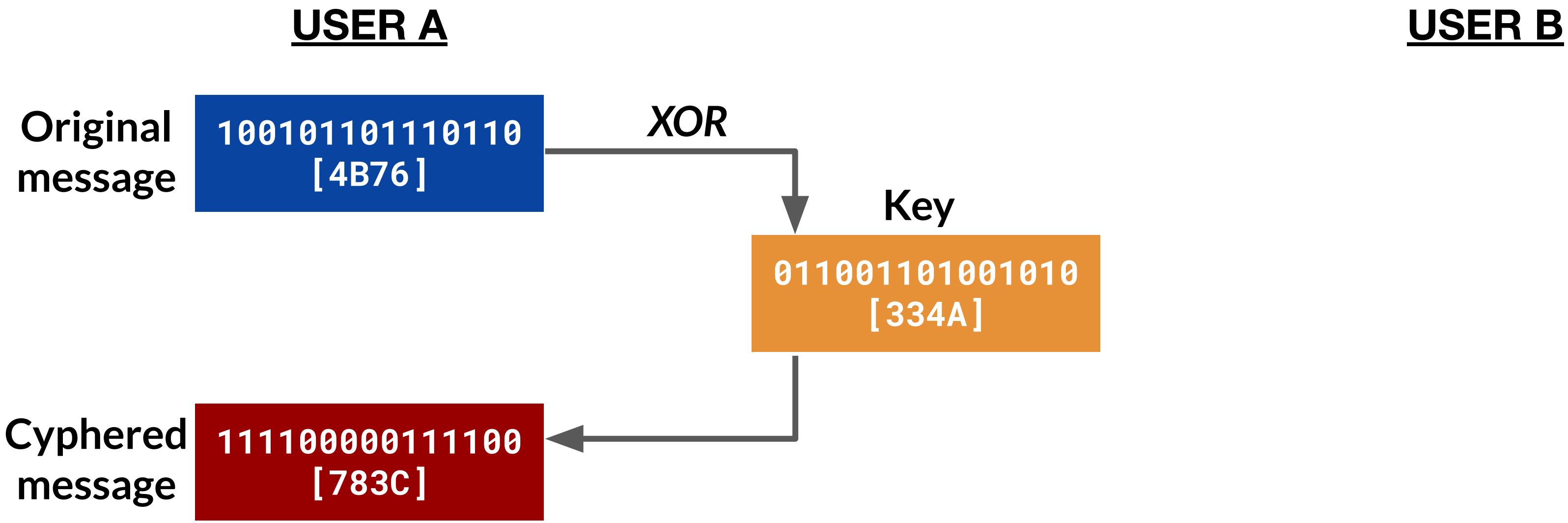
100101101110110
[4B76]

Key

011001101001010
[334A]

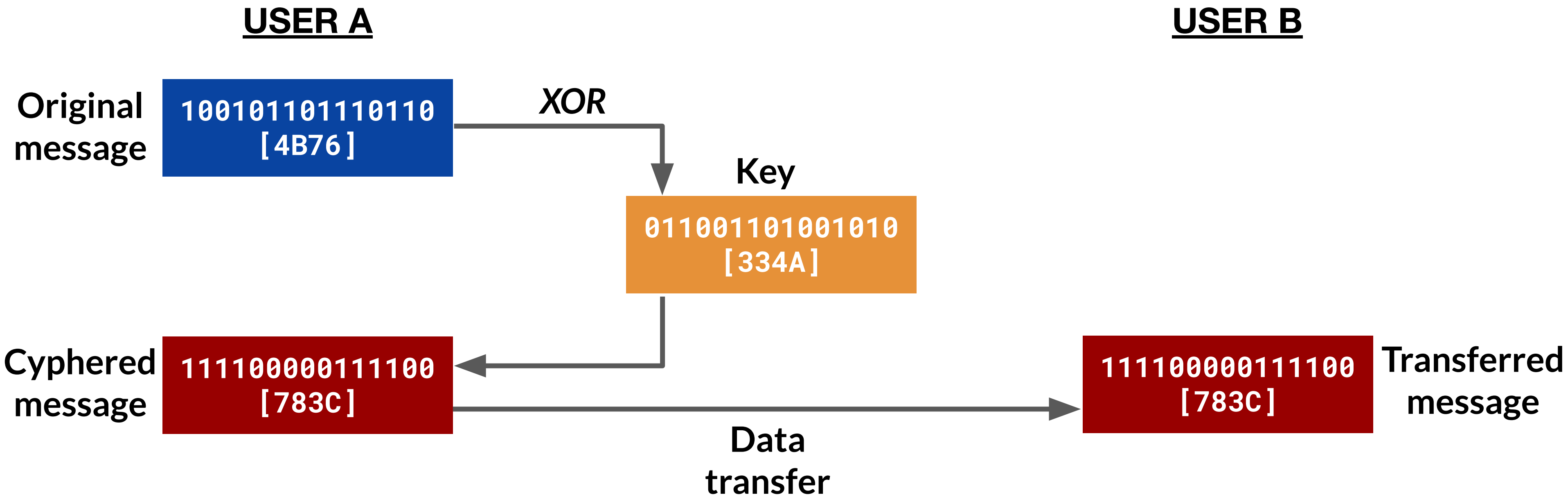
ENCODING/DECODING INFORMATION

- A simple example is passing a message ***M*** of length ***L*** between two users
- Both users also exchange a secret key ***K*** of the same length ***L*** to scramble (cypher) and reassemble (decypher) the data
- A simple **XOR** logic function can be used to encode-decode the information
  (once again, using XOR only to simplify the visualization of the topic)



ENCODING/DECODING INFORMATION

- A simple example is passing a message ***M*** of length ***L*** between two users
- Both users also exchange a secret key ***K*** of the same length ***L*** to scramble (cypher) and reassemble (decypher) the data
- A simple **XOR** logic function can be used to encode-decode the information
  (once again, using XOR only to simplify the visualization of the topic)



USER A

USER B

Original message

1001011101110110
[4B76]

XOR

Key

0110011101001010
[334A]

Cyphered message

1111000000111100
[783C]

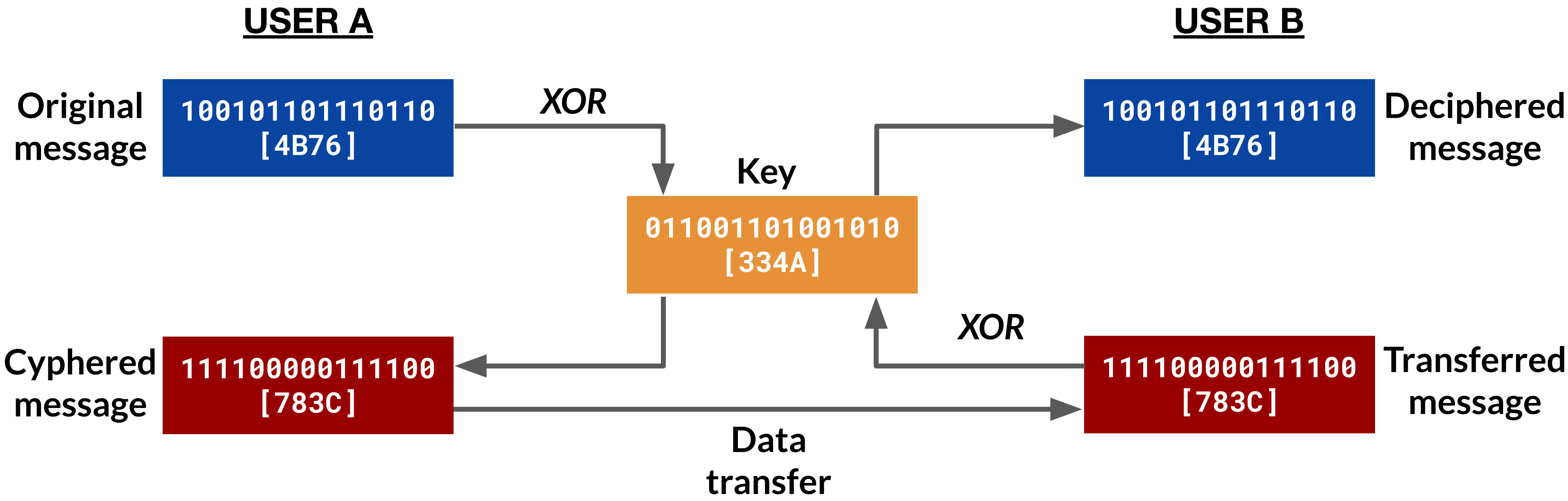
Data transfer

1111000000111100
[783C]

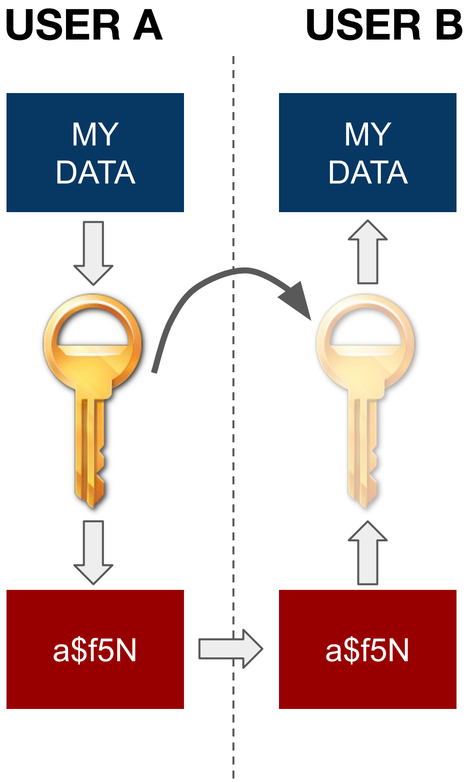
Transferred message

ENCODING/DECODING INFORMATION

- A simple example is passing a message ***M*** of length ***L*** between two users
- Both users also exchange a secret key ***K*** of the same length ***L*** to scramble (cypher) and reassemble (decypher) the data
- A simple **XOR** logic function can be used to encode-decode the information
  (once again, using XOR only to simplify the visualization of the topic)



SYMMETRIC KEY ENCRYPTION



Private-key cryptography

Probably the simplest encryption technique

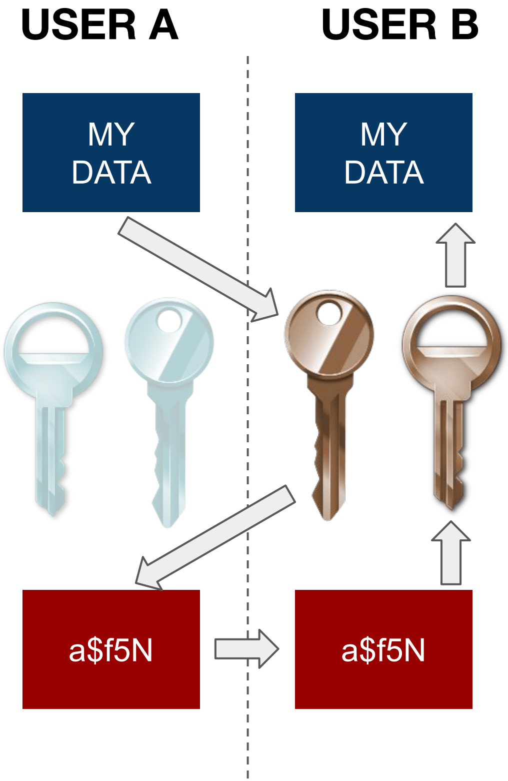
- Symmetric encryption algorithms include AES-128, AES-192, and AES-256
- Simple encryption algorithms
  → very fast and very simple to implement!

However:

- All ends of the data communication have to exchange the same (private) key used to encrypt the data to be able to decrypt it

For this reason it is mostly used as a "2nd layer" of encryption, after a first more robust stage

ASYMMETRIC KEY ENCRYPTION



Public-key cryptography

Each user is assigned a key pair:

- **public** → used to **encrypt** data
- **private** → used to **decrypt** data

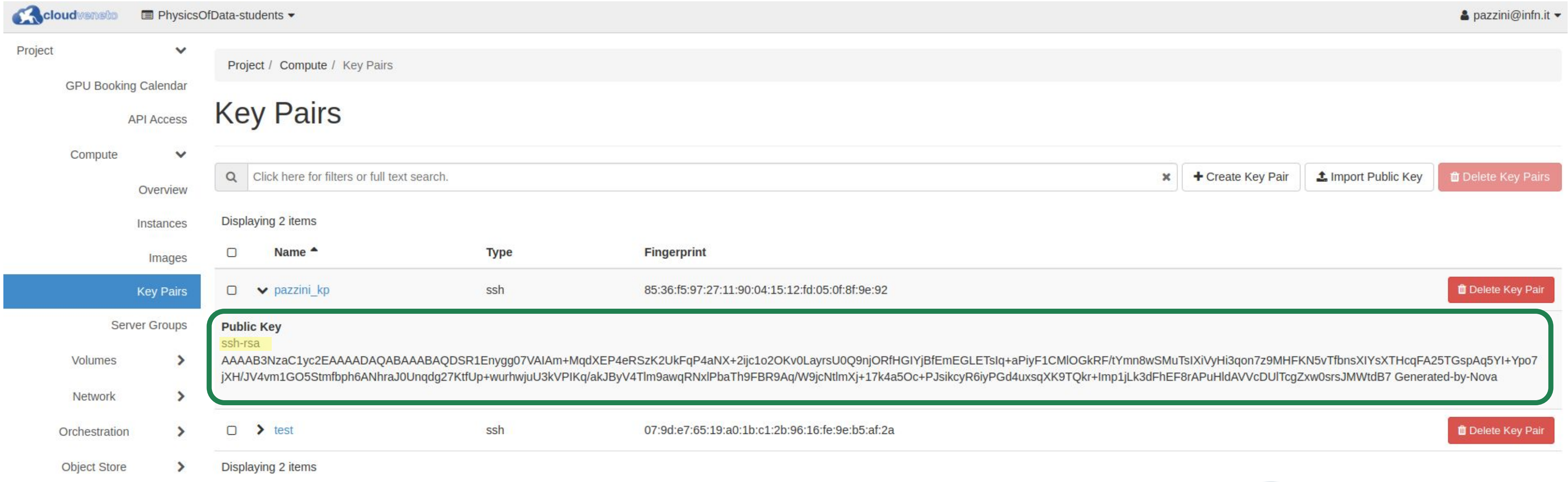
⇒ The relation between the two keys is unknown
⇒ From one key is not possible to infer the other

Asymmetric encryption algorithms include RSA, ECC, …
→ Typically substantially slower than symmetric-key encryption algos

For its security is one of the standards for connecting to remote services (e.g. a VM on Cloud Veneto)

PRIVATE KEY	→	kept a secret
PUBLIC KEY	→	freely available to anyone who might want to send you a message

ASYMMETRIC KEY ENCRYPTION - CLOUDVENETO



Project / Compute / Key Pairs

Key Pairs

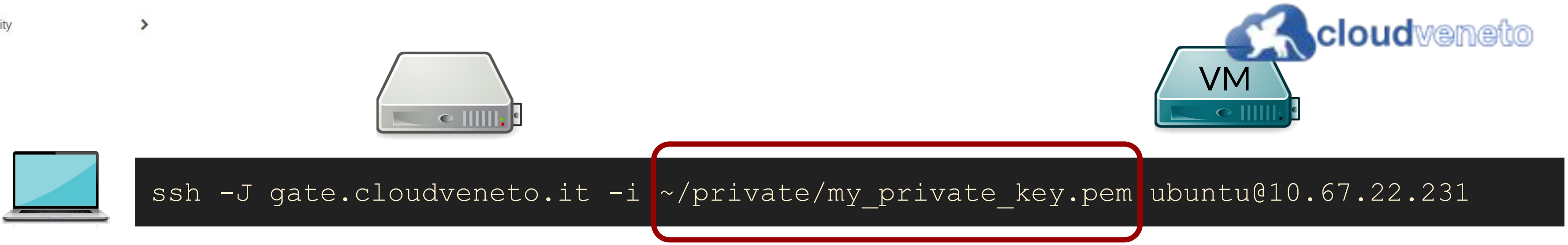
Displaying 2 items

□	Name ▲	Type	Fingerprint	
□	pazzini_kp	ssh	85:36:f5:97:27:11:90:04:15:12:fd:05:0f:8f:9e:92	Delete Key Pair
□	test	ssh	07:9d:e7:65:19:a0:1b:c1:2b:96:16:fe:9e:b5:af:2a	Delete Key Pair

Public Key (pazzini_kp)

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDSR1Enygg07VAIAm+MqdXEP4eRSzK2UkFqP4aNX+2ijc1o2OKv0LayrsU0Q9njORfHGIYjBfEmEGLETsIq+aPiyF1CMlOGkRF/tYmn8wSMuTsIXiVyHi3qon7z9MHFKN5vTfbnsXIYsXTHcqFA25TGspAq5YI+Ypo7
jXH/JV4vm1GO5Stmfbph6ANhraJ0Unqdg27KtfUp+wurhwjuU3kVPIKq/akJByV4Tlm9awqRNxlPbaTh9FBR9Aq/W9jcNtlmXj+17k4a5Oc+PJsikcyR6iyPGd4uxsqXK9TQkr+Imp1jLk3dFhEF8rAPuHldAVVcDUlTcgZxw0srsJMWtdB7 Generated-by-Nova
```

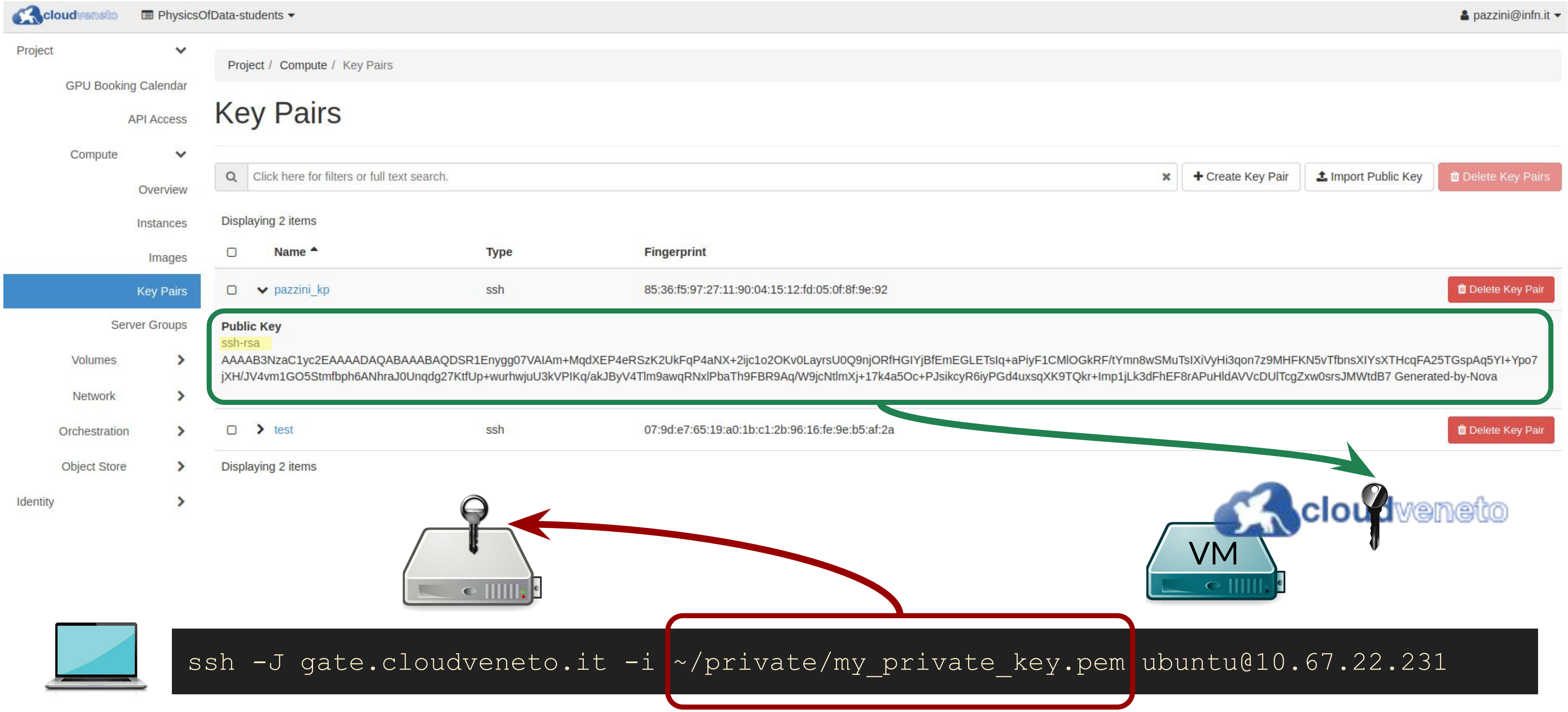
Displaying 2 items



```
ssh -J gate.cloudveneto.it -i ~/private/my_private_key.pem ubuntu@10.67.22.231
```

As always… carefully check the documentation first ⟶ https://userguide.cloudveneto.it/en/latest/GettingStarted.html#creating-a-keypair

ASYMMETRIC KEY ENCRYPTION - CLOUDVENETO



Key Pairs

Displaying 2 items

□	Name	Type	Fingerprint	
□	pazzini_kp	ssh	85:36:f5:97:27:11:90:04:15:12:fd:05:0f:8f:9e:92	Delete Key Pair
□	test	ssh	07:9d:e7:65:19:a0:1b:c1:2b:96:16:fe:9e:b5:af:2a	Delete Key Pair

Public Key

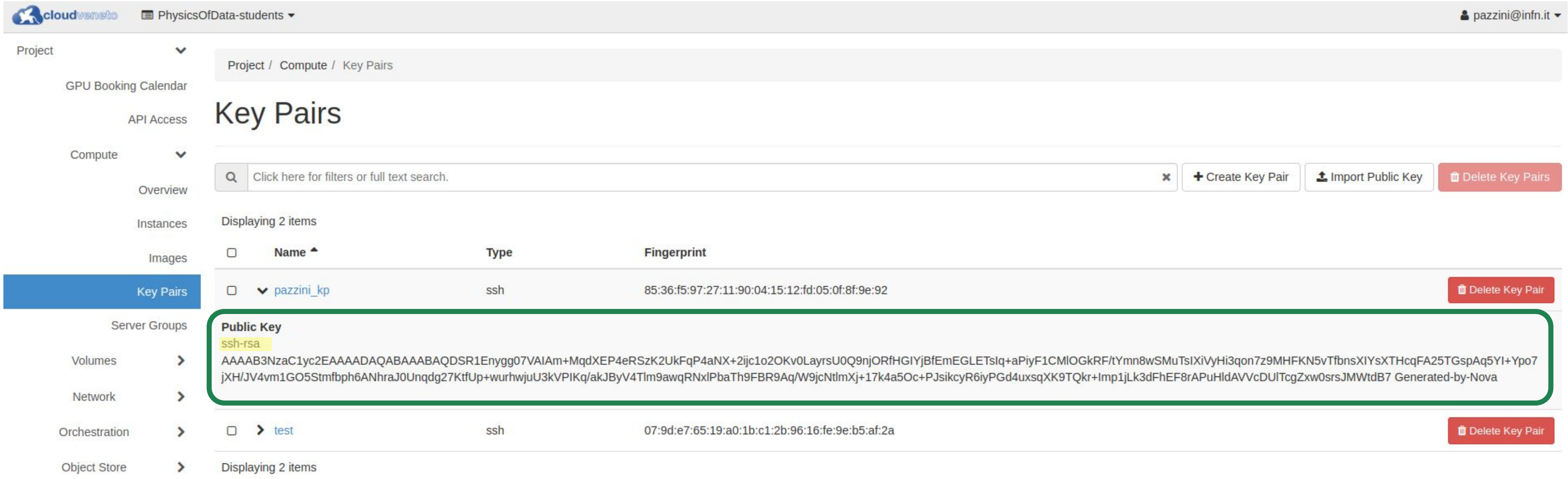
```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDSR1Enygg07VAIAm+MqdXEP4eRSzK2UkFqP4aNX+2ijc1o2OKv0LayrsU0Q9njORfHGIYjBfEmEGLETslq+aPiyF1CMlOGkRF/tYmn8wSMuTsIXiVyHi3qon7z9MHFKN5vTfbnsXIYsXTHcqFA25TGspAq5YI+Ypo7
jXH/JV4vm1GO5Stmfbph6ANhraJ0Unqdg27KtfUp+wurhwjuU3kVPIKq/akJByV4Tlm9awqRNxlPbaTh9FBR9Aq/W9jcNtlmXj+17k4a5Oc+PJsikcyR6iyPGd4uxsqXK9TQkr+Imp1jLk3dFhEF8rAPuHldAVVcDUlTcgZxw0srsJMWtdB7 Generated-by-Nova
```

Displaying 2 items

```
ssh -J gate.cloudveneto.it -i ~/private/my_private_key.pem ubuntu@10.67.22.231
```

As always… carefully check the documentation first ⟶ https://userguide.cloudveneto.it/en/latest/GettingStarted.html#creating-a-keypair

ASYMMETRIC KEY ENCRYPTION - CLOUDVENETO



cloudveneto PhysicsOfData-students ▾ pazzini@infn.it ▾

Project
GPU Booking Calendar
API Access
Compute
Overview
Instances
Images
Key Pairs
Server Groups
Volumes
Network
Orchestration
Object Store
Identity

Project / Compute / Key Pairs

Key Pairs

Click here for filters or full text search. ✖ + Create Key Pair Import Public Key Delete Key Pairs

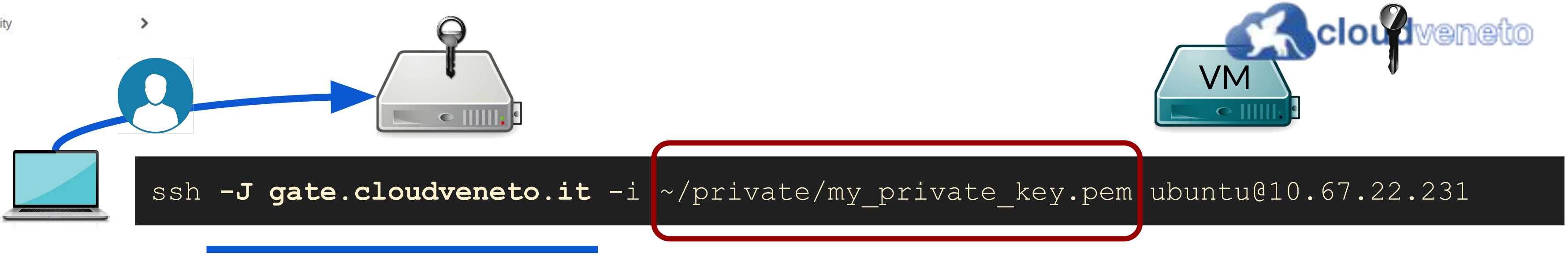
Displaying 2 items

□	Name ▴	Type	Fingerprint	
□	⌄ pazzini_kp	ssh	85:36:f5:97:27:11:90:04:15:12:fd:05:0f:8f:9e:92	Delete Key Pair
□	› test	ssh	07:9d:e7:65:19:a0:1b:c1:2b:96:16:fe:9e:b5:af:2a	Delete Key Pair

Public Key

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDSR1Enygg07VAIAm+MqdXEP4eRSzK2UkFqP4aNX+2ijc1o2OKv0LayrsU0Q9njORfHGIYjBfEmEGLETslq+aPiyF1CMlOGkRF/tYmn8wSMuTsIXiVyHi3qon7z9MHFKN5vTfbnsXIYsXTHcqFA25TGspAq5YI+Ypo7
jXH/JV4vm1GO5Stmfbph6ANhraJ0Unqdg27KtfUp+wurhwjuU3kVPIKq/akJByV4Tlm9awqRNxlPbaTh9FBR9Aq/W9jcNtlmXj+17k4a5Oc+PJsikcyR6iyPGd4uxsqXK9TQkr+Imp1jLk3dFhEF8rAPuHldAVVcDUlTcgZxw0srsJMWtdB7 Generated-by-Nova

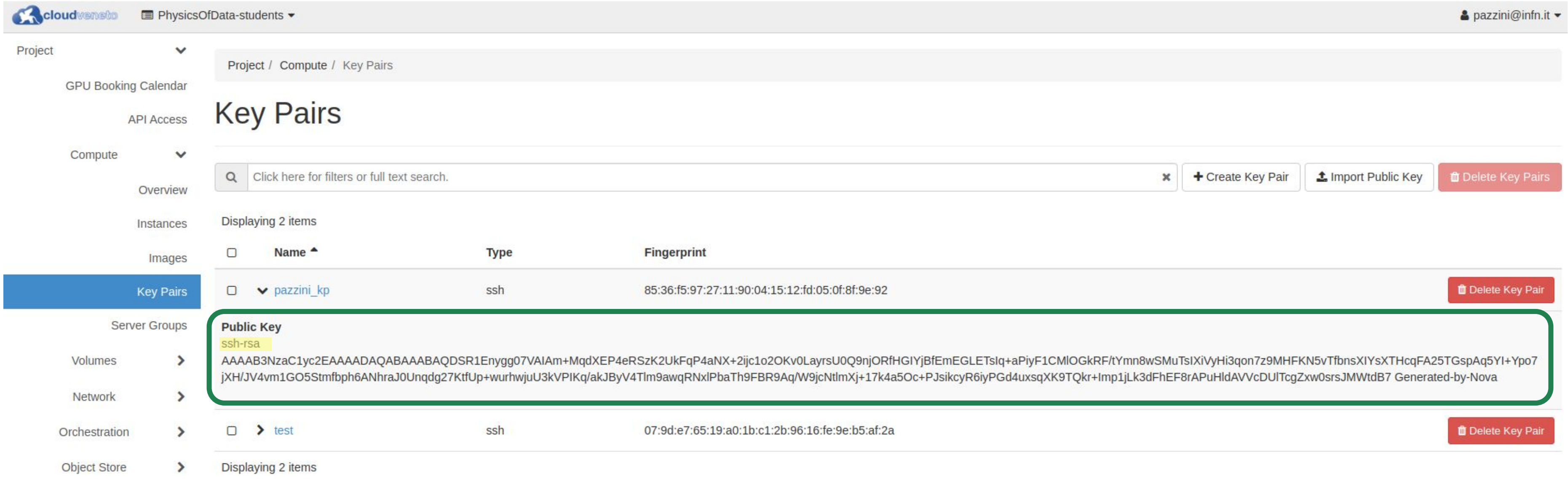
Displaying 2 items



```
ssh -J gate.cloudveneto.it -i ~/private/my_private_key.pem ubuntu@10.67.22.231
```

As always… carefully check the documentation first ⟶ https://userguide.cloudveneto.it/en/latest/GettingStarted.html#creating-a-keypair

ASYMMETRIC KEY ENCRYPTION - CLOUDVENETO



Key Pairs

Displaying 2 items

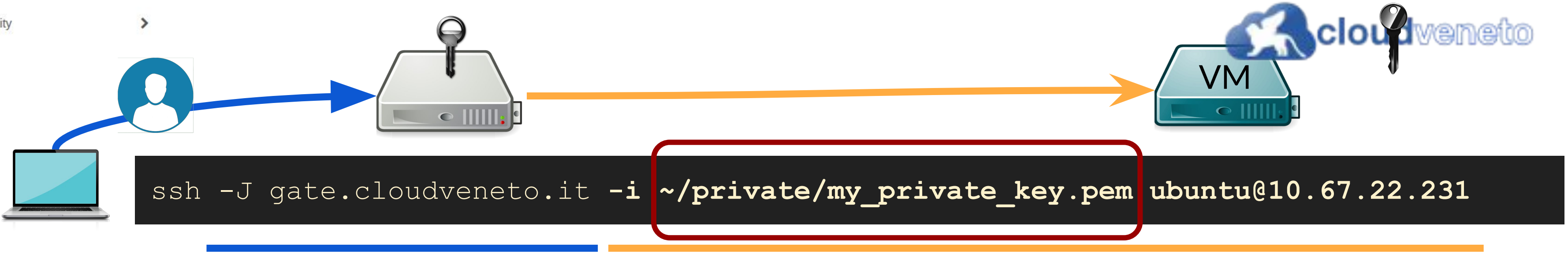
□	Name	Type	Fingerprint	
□	pazzini_kp	ssh	85:36:f5:97:27:11:90:04:15:12:fd:05:0f:8f:9e:92	Delete Key Pair
□	test	ssh	07:9d:e7:65:19:a0:1b:c1:2b:96:16:fe:9e:b5:af:2a	Delete Key Pair

Public Key

ssh-rsa

AAAAB3NzaC1yc2EAAAADAQABAAABAQDSR1Enygg07VAIAm+MqdXEP4eRSzK2UkFqP4aNX+2ijc1o2OKv0LayrsU0Q9njORfHGIYjBfEmEGLETsIq+aPiyF1CMlOGkRF/tYmn8wSMuTsIXiVyHi3qon7z9MHFKN5vTfbnsXIYsXTHcqFA25TGspAq5YI+Ypo7jXH/JV4vm1GO5Stmfbph6ANhraJ0Unqdg27KtfUp+wurhwjuU3kVPIKq/akJByV4Tlm9awqRNxlPbaTh9FBR9Aq/W9jcNtlmXj+17k4a5Oc+PJsikcyR6iyPGd4uxsqXK9TQkr+Imp1jLk3dFhEF8rAPuHldAVVcDUlTcgZxw0srsJMWtdB7 Generated-by-Nova

Displaying 2 items



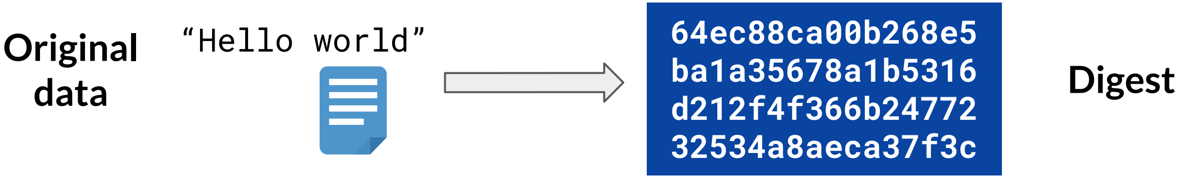
```
ssh -J gate.cloudveneto.it -i ~/private/my_private_key.pem ubuntu@10.67.22.231
```

As always… carefully check the documentation first ⟶ https://userguide.cloudveneto.it/en/latest/GettingStarted.html#creating-a-keypair

CRYPTOGRAPHIC HASH FUNCTIONS

Hashing refers to the scrambling of raw information to the extent that it cannot be reproduced back to its original form

Hashing algorithms (MD5, SHA-1*, SHA-2*, …) transform data of any size into a **fixed-size** short version of it, a ***digest***



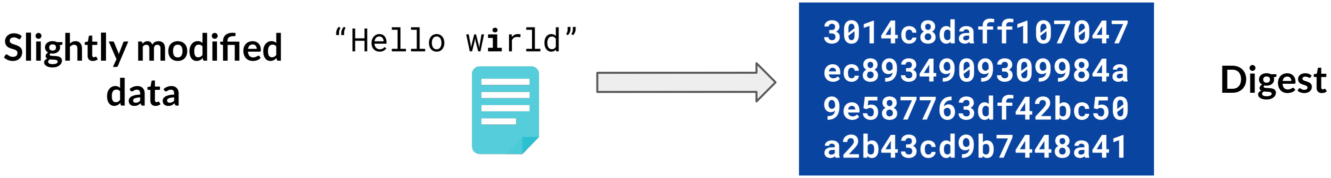
Original data

```
“Hello world”
```

```
64ec88ca00b268e5
ba1a35678a1b5316
d212f4f366b24772
32534a8aeca37f3c
```

Digest

Any small difference in the original data will result in a totally different digest



Slightly modified data

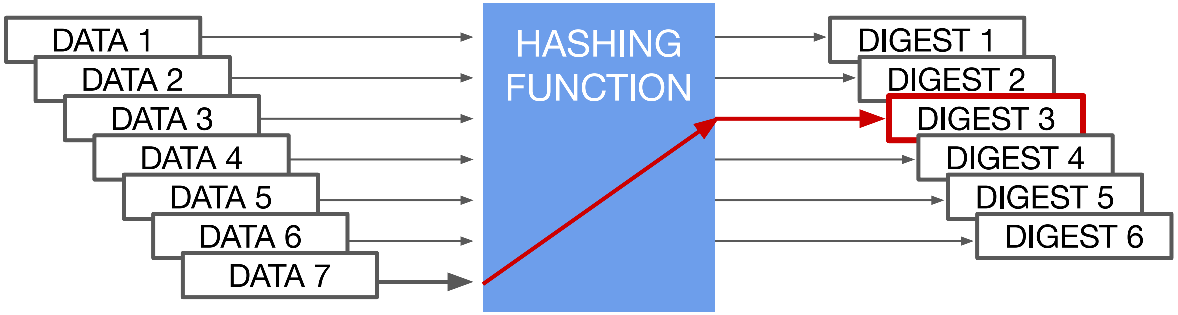
```
“Hello wirld”
```

```
3014c8daff107047
ec8934909309984a
9e587763df42bc50
a2b43cd9b7448a41
```

Digest

CRYPTOGRAPHIC HASH FUNCTIONS

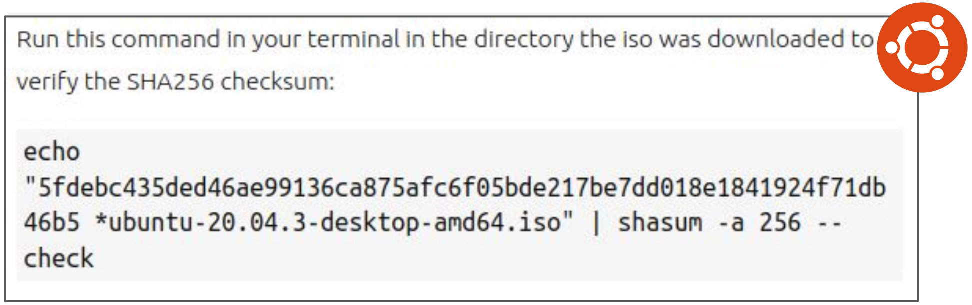
Hashing algorithms are designed to **avoid producing 2 identical digests for 2 different raw-data** → this concept is known as ***hash collision****



Hashing algorithms are used to:

- Mask/protect data → e.g. avoid storing username/password in plain text
- Data integrity verifications → check if digest of copied data is the same as the original one
- Digital signature verification →hash a document + encrypt the hash to enable sign verification

password	website database
123456	d3c29a3a629
password	ca12020c92f
0987654321	ebd9034323e
p@ssword	9cbaf436906


Run this command in your terminal in the directory the iso was downloaded to verify the SHA256 checksum:

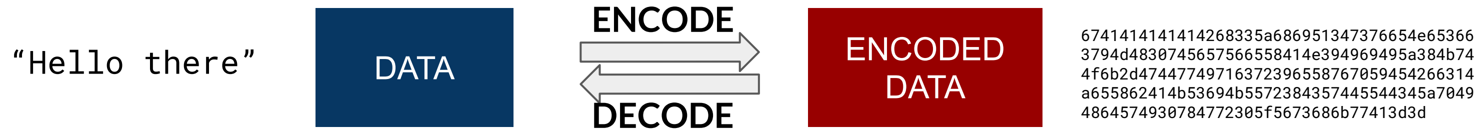
```
echo
"5fdebc435ded46ae99136ca875afc6f05bde217be7dd018e1841924f71db
46b5 *ubuntu-20.04.3-desktop-amd64.iso" | shasum -a 256 --
check
```

*see pigeonhole principle on wikipedia

ENCRYPTION VS HASHING

ENCRYPTION

- **Encryption algorithms** *encode and compress data* for confidentiality and to secure transfer
- They must be **fully reversible** to allow decryption
- Can be computationally expensive

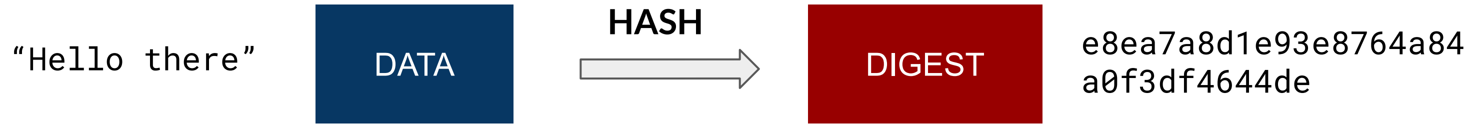


“Hello there”

```
6741414141414268335a686951347376654e65366
3794d4830745657566558414e394969495a384b74
4f6b2d47447749716372396558767059454266314
a655862414b53694b557238435744554434 5a7049
486457493078477 2305f5673686b77413d3d
```

HASHING

- **Hashing algorithms** scramble data into *fixed-size digest* to hide/check data
- They must **not be reversible** and must produce a **completely different digest for any small difference in data** (even 1 bit)
- Must be computationally efficient and avoid *hash collision*



“Hello there”

```
e8ea7a8d1e93e8764a84
a0f3df4644de
```