Computer Security (COM-301)

Authentication

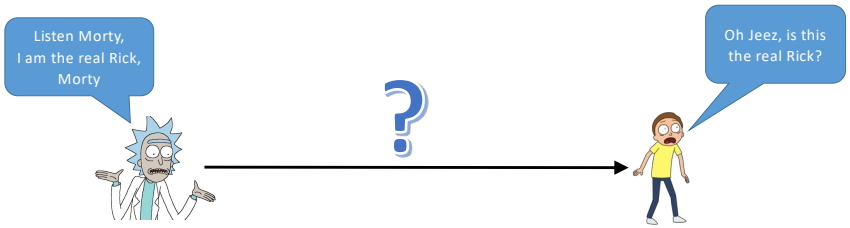
Basics and passwords

Carmela Troncoso
SPRING Lab
carmela.troncoso@epfl.ch

Some slides/ideas adapted from: Tuomas Aura, Yoshi Kohno, Trent Jaeger

What is authentication?

AUTHENTICATION
The process of verifying a claimed identity

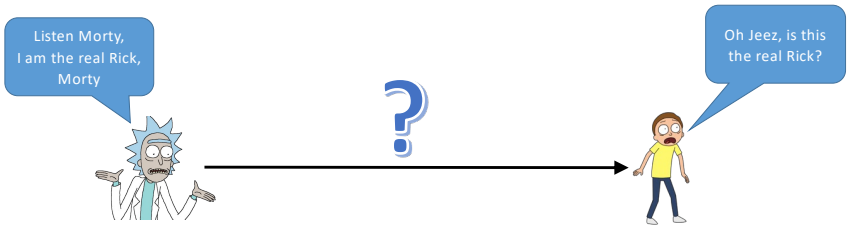


2

Authentication is the process that entities use to verify that other entities are who they claim to be.

What is authentication?

AUTHENTICATION
The process of verifying a claimed identity



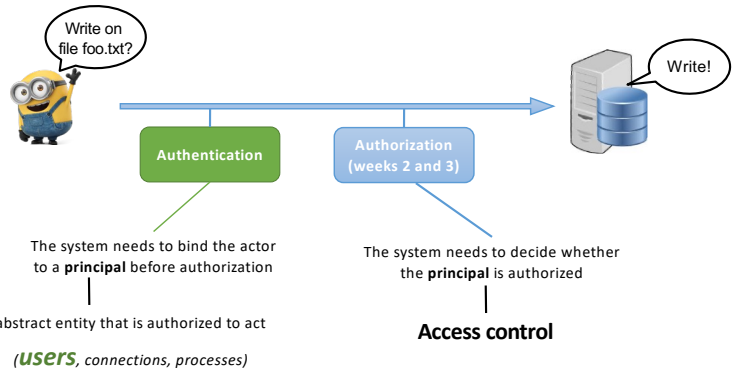
!= MESSAGE AUTHENTICATION
The message comes from the designated sender, and has not been modified

3

Authentication must *not* to be confused with *Message authentication* in which we verify:

message provenance (the message is sent by whom we believe has sent it)
message integrity (the message that we receive is the message that the sender sent)

Where does Authentication fit?



4

Recall that *access control* establishes whether a principal is authorized to operate on an asset. Before the access control mechanism can make this decision, **Authentication** needs to happen to ensure that the principal is who he says he is.

We will mostly cover authentication of **users**, and a bit of machine authentication.

Ways to Prove Who You Are

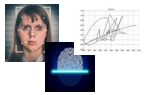
TRADITIONAL

What you know

password, secret key

What you are

biometrics



What you have

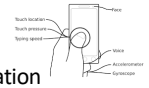
Smart card, secure tokens

MODERN

Where you are

location, IP address

How you act



behavioural authentication

Who you know

social ties

Many others…

5

Traditionally, we considered three ways of proving your identity:

- What you know: a secret that only you can know, and therefore proving knowledge of the secret ensures that it is you
- What you are: a trait that is inherent to you and no-one else has, and therefore proving this trait ensures that it is you
- What you have: an object that only you can have, and therefore proving possession ensures that it is you

In modern life, the traditional means are augmented by other traits are often unique (some times called soft biometrics):

- where you are: applications detect your common locations and trigger alarms if you are far
- how you act: how you type, how you move the mouse, how you pressure the keyboard
- who are your friends: social networks tend to be unique

Ways to Prove Who You Are

TRADITIONAL

What you know

pas

What you a

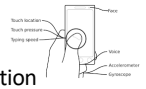
biom

What you ha

Smart card, secure tokens

MODERN

s



tication

Who you know

social ties

6

In fact Bank applications already use all of that, and more!

What you know: Passwords

PASSWORD
Secret shared between user and system

User has a secret password → System checks it to authenticate the user

The most typical “what you know” are passwords.

What you know: Passwords

PASSWORD
Secret shared between user and system

User has a secret password → System checks it to authenticate the user

PROBLEMS TO BE SOLVED

Secure transfer: the password may be eavesdropped when communicated

When using passwords: one must take into account 4 aspects

1) We will need to send the password, and we cannot guarantee the transfer will be done via a secure channel. We need to transfer it securely to maintain it secret

What you know: Passwords

PASSWORD
Secret shared between user and system

User has a secret password → System checks it to authenticate the user

PROBLEMS TO BE SOLVED

Secure transfer: the password may be eavesdropped when communicated

Secure check: naïve checks may leak information about the password

2) We need to make sure that when trying an erroneous password, the answer does not reveal anything about the actual password

What you know: Passwords

PASSWORD
Secret shared between user and system

User has a secret password → System checks it to authenticate the user

PROBLEMS TO BE SOLVED

Secure transfer: the password may be eavesdropped when communicated

Secure check: naïve checks may leak information about the password

Secure storage: if stolen the full system is compromised!

3) To keep passwords secret, they should also be securely stored

What you know: Passwords

PASSWORD
Secret shared between user and system

User has a secret password → System checks it to authenticate the user

PROBLEMS TO BE SOLVED

Secure transfer: the password may be eavesdropped when communicated

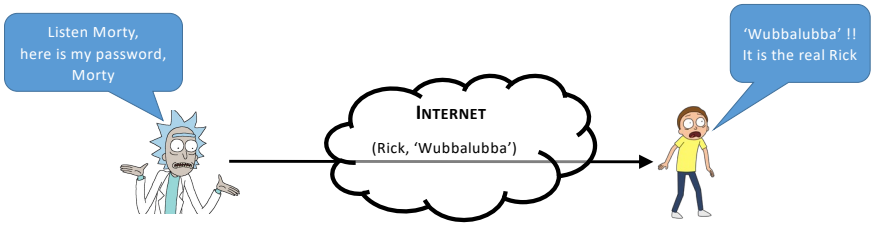
Secure check: naïve checks may leak information about the password

Secure storage: if stolen the full system is compromised!

Secure passwords: easy-to-remember passwords tend to be easy to guess

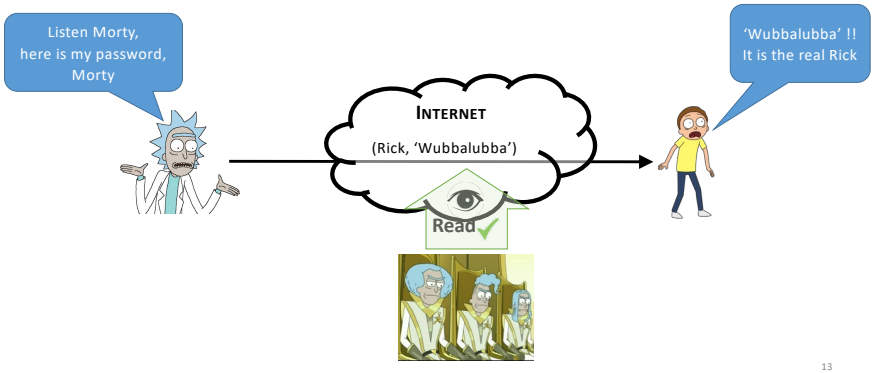
4) The secrecy of the password depends on how easy it is to predict. If the password can be guessed, it does not matter all the above is secured. The password is still not secret.

Secure transfer



To achieve authentication, we would send the password together with the "login"

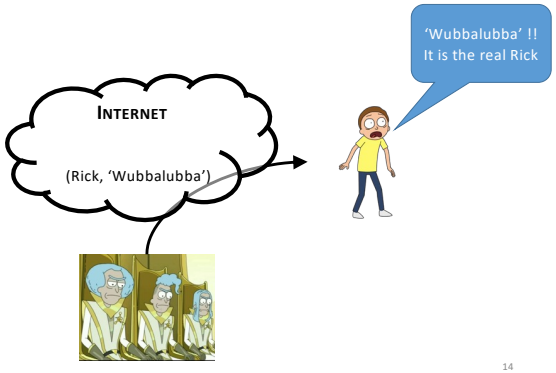
Secure transfer



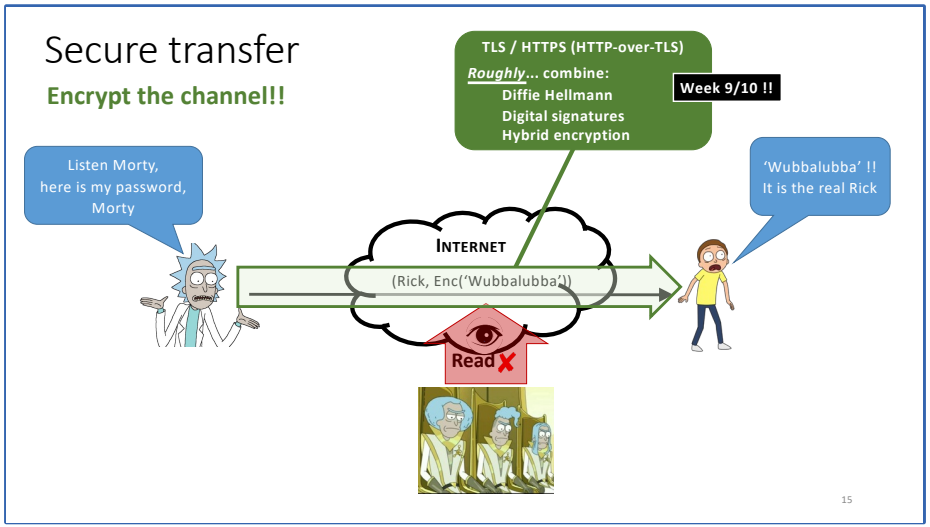
13

But if the password is sent in the clear together with the login, anyone eavesdropping on the link can learn this tuple

Secure transfer

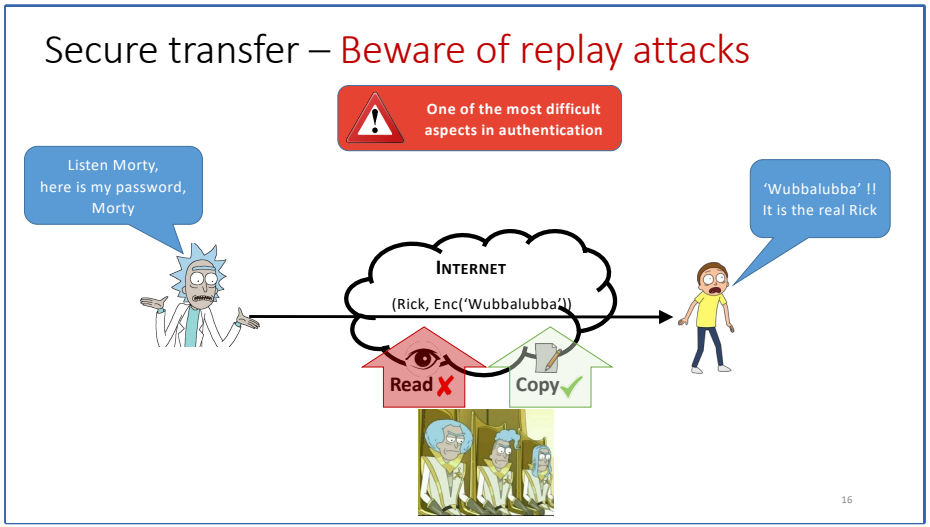


Once they learn the tuple they can use it to impersonate the entity at any point in time!



We have already seen in the class the means to secure the transfer: encrypt the tuple login password in transit so that it cannot be eavesdropped.

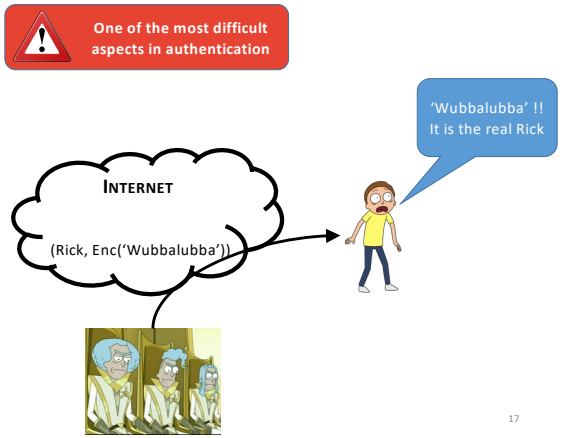
Typically one does not just encrypt this information, one also **encrypts the channel** using a standard protocol (like TLS or HTTPS, which combine the concepts we have seen in a safe way – DO **NOT** build your own protocol)



But encrypting the password may not completely solve the problem (e.g., if we don't use a standard protocol to encrypt the channel, but we only encrypt the password; or if the eavesdropper)

Even if the password is encrypted, one can copy the content

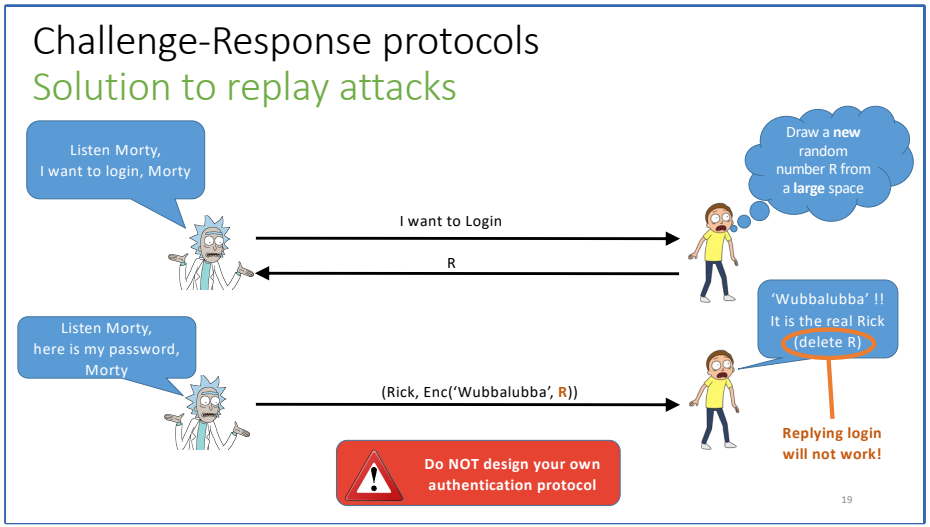
Secure transfer – Beware of replay attacks



And then reply the encrypted content…

Challenge-Response protocols

Solution to replay attacks

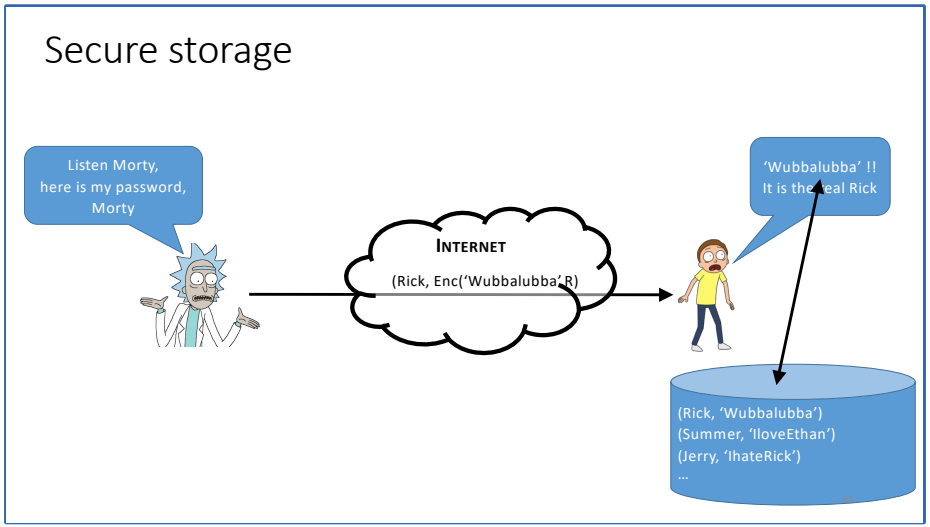


To avoid reply attacks (for passwords and in general) one uses **Challenge-Response** protocols.

In these protocols :

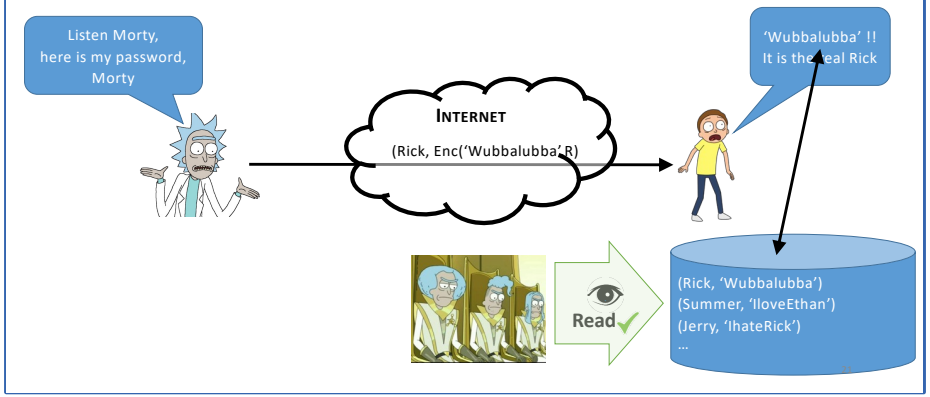
1) The principal that wants to authenticate first declares their intention to login.
2) The party authenticating the principal sends a ***challenge***, a random number (from a large space, to ensure that there are no repetitions and cannot be predicted)
3) The principal encrypts the password *together* with the challenge. This ensures that i) the password is never encrypted to the same value, and ii) the encryption is fresh (i.e., corresponds to the declaration in step 1).

After checking the password, the authenticating party **must** delete the challenge so that the message cannot be replied.

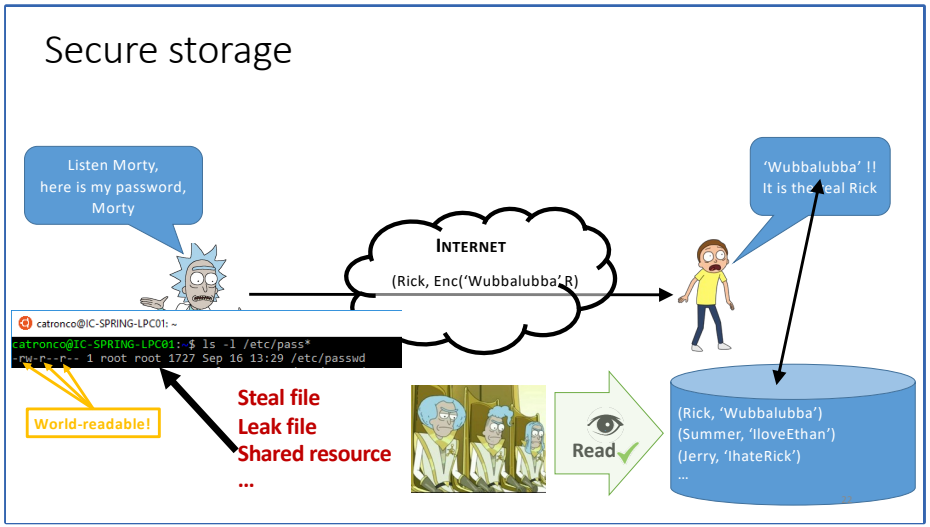


When Morty receives the tuple, in order to be sure that this tuple indeed authenticates Rick, Morty needs to know what is the correspondence between Rick and his password. This is typically stored in some sort of database.

Secure storage



But if passwords are stored in the clear, an adversary can read them (e.g., in UNIX the password file is readable by anyone!) and then use them to impersonate the users.



These can be read if the file is leaked, stolen, or if people have access to the same machine

Password database compromises

Source: Tom Ristenpart

23

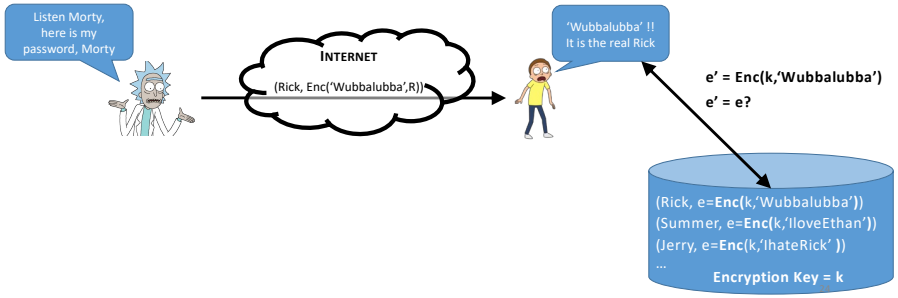
Also, remember the asymmetry of the adversary vs. the defender. We do not need all the adversaries to be able to read the passwords or breach into the database. It suffices that one attacker gets access and leaks the passwords of millions of people

(fortunately most of these databases are not in the clear, although that does not mean they are safe, see below).

Secure storage - Do not store in the clear!

OPTION 1

Store password encrypted



A first idea can be to encrypt the passwords under storage.

When Morty receives the password, decrypts it, and checks the challenge.
If the challenge is fresh, then to check if it corresponds to the encryption stored in the database he can:

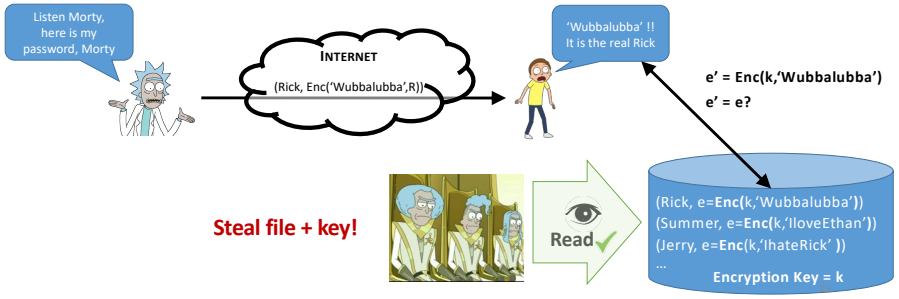
1) encrypt the received password and compare it to the stored value (this is shown in the slide)

2) decrypt the stored password and compare it to the encrypted value

Secure storage - Do not store in the clear!

OPTION 1

Store password encrypted

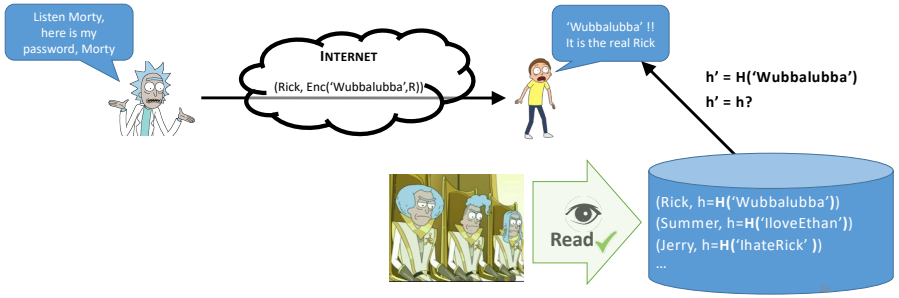


But note that encryption is as secure as the security of the key. To encrypt/decrypt the passwords the key needs to be on the same machine, so if the adversary can access the database, can also steal the key.

Secure storage - Do not store in the clear!

OPTION 2

Store password as a “hash” of its value



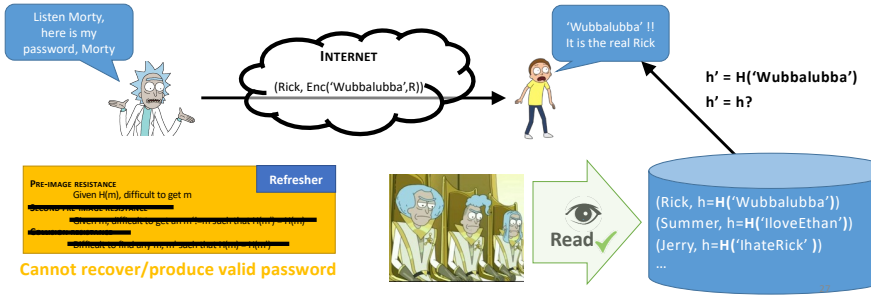
A primitive that does not require a key and allows for checking is a hash function.

When Morty receives the password, decrypts it, and checks the challenge.
If the challenge is fresh, Morty hashes the received password and compares it to the stored value.

Secure storage - Do not store in the clear!

OPTION 2

Store password as a “hash” of its value

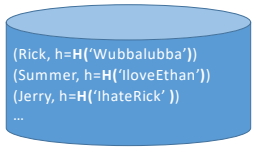


In this case, what is important is that if the adversary gets access to the database cannot recover the password from the hashes: i.e., the hash function must be pre-image resistant.

Secure storage - Do not store in the clear!

OPTION 2

Store password as a “hash” of its value



OFFLINE ATTACKS – DICTIONARY ATTACK

Anyone can compute a hash

Passwords not truly random

- 52 upper- and lower-case letters, 10 digits and 32 punctuation symbols,
- 94^8 eight-character passwords (around 2^{52}) possibilities

Users use a limited set of passwords (reduced search space)

https://en.wikipedia.org/wiki/List_of_the_most_common_passwords

https://resources.infosecinstitute.com/10-popular-password-cracking-tools/#gref

But again this still does not solve the problem.

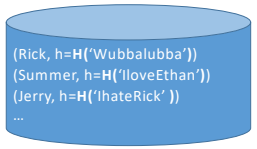
Combine the following:

- A hash **does not** require a key, anyone can compute a hash.
- Passwords are not truly random: we use mostly letters (upper and lower) and punctuation symbols. The space is not inexplorable
- And also, among those, not every password is equally likely!

Secure storage - Do not store in the clear!

OPTION 2

Store password as a “hash” of its value



OFFLINE ATTACKS – DICTIONARY ATTACK

Anyone can compute a hash

Passwords not truly random

- 52 upper- and lower-case letters, 10 digits and 32 punctuation symbols,
- 94^8 eight-character passwords (around 2^{52}) possibilities

Users use a limited set of passwords (reduced search space)

Attacker can compute H(word) for every word in the dictionary and see if the result is in the password file!

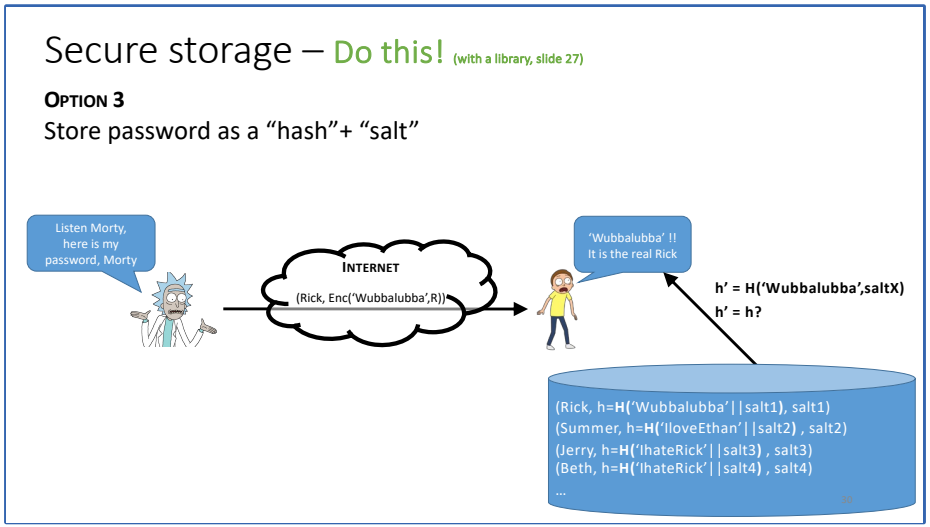
- Can reuse the dictionary
- Parallel cracking with GPU accelerates search
- Other tricks: rainbow tables, pre-computation,…

https://en.wikipedia.org/wiki/List_of_the_most_common_passwords

https://resources.infosecinstitute.com/10-popular-password-cracking-tools/#gref

One can compute the hash of the most typical passwords and start trying them.

More tricks are available to speed up: use parallel computing, rainbow tables, etc.



How we solve this is with so-called **salts**.

When using salts, one:
- chooses a salt for every password
- hashes the password concatenated with the corresponding hash -> Note that this means that a password will look different when salted with different values
- stores the hash together with the salt (the salt is in the clear)

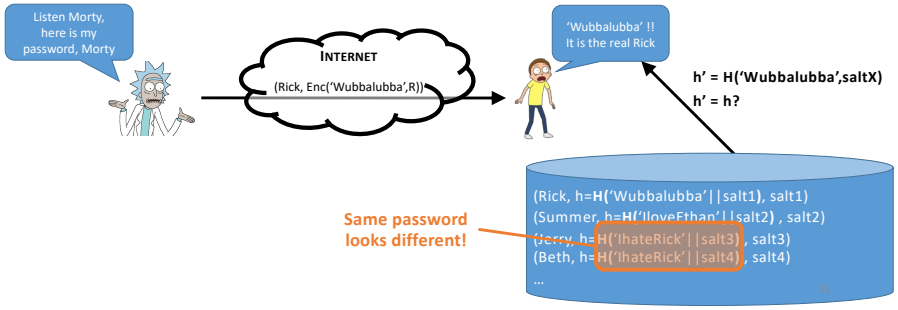
When verifying a password Morty receives the password, decrypts it, and checks the challenge.
If the challenge is fresh, Morty finds the corresponding salt (the one in Rick’s entry) in the database, concatenates it with the received password and computes the hash. Then compares the result to the stored value.

Note that breaking one password is still possible. As the salt is in the clear one can try to compute the hash of all possible passwords with that hash. So one targeted attack is equally cheap than before, but retrieving the passwords for all entries in the database is very expensive.

Secure storage – Do this! (with a library, slide 27)

OPTION 3

Store password as a “hash”+ “salt”

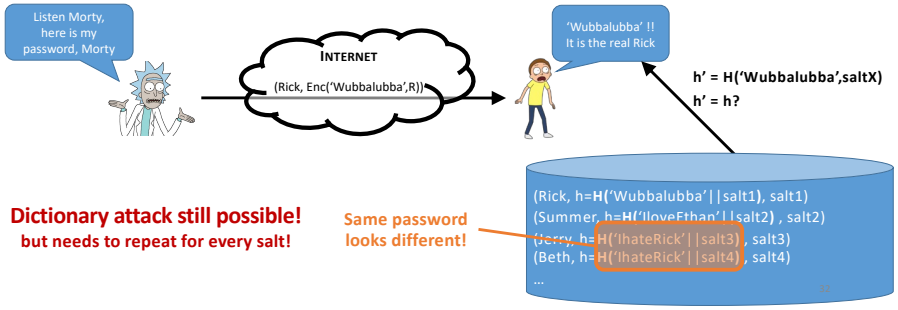


- hashes the password concatenated with the corresponding hash **-> Note that this means that a password will look different when salted with different values**

Secure storage – Do this! (with a library, slide 27)

OPTION 3

Store password as a “hash”+ “salt”



Breaking one password is still possible! As the salt is in the clear one can compute the hash of all possible passwords with that hash. So one targeted attack as cheap as before, but retrieving the passwords for all entries in the database is very expensive.

Secure storage – Do this!

OPTION 3

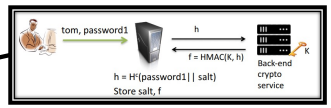
Store password as a “hash”+ “salt”

COMPLEMENTARY DEFENSES

Use of hash functions designed to be **slow** (bcrypt, scrypt, argon2)
Repeat several times (e.g., 1000)

Require specific elements in passwords
Increase entropy

Split check, require a second server
Invalidate offline attacks



Access control! (`/etc/shadow` in UNIX only accessible by root)

33

Actually, people do even more things:

- Instead of using one normal hash function designed to be fast, use one that is slow, so as to slow down the adversary that tries to compute the dictionary. Then repeat several times, not because it is more secure, but because it slows the attack even more. *Important*: slow down means that if one needs to compute this for many passwords the difference is noticeable, but for every login check there is virtually no different.
- Ensure that people do not choose typical passwords, and require specific elements so that they are random and it is harder to guess them
- Require a server to do a computation. This means that the adversary cannot compute an offline dictionary.
- And of course, hinder as much as possible access to the passwords database! Access control is your friend.

Facebook password onion

```
$cur  = 'password'
$cur  = md5($cur)
$salt = randbytes(20)
$cur  = hmac_sha1($cur, $salt)
$cur  = remote_hmac_sha256($cur, $secret)
$cur  = scrypt($cur, $salt)
$cur  = hmac_sha256($cur, $salt)
```

Why this onion?

Source: Tom Ristenpart

34

Coping with legacy!

Password database compromises

	year	# stolen	% recovered	format
rockyou	2012	32.6 million	100%	plaintext (!)
LinkedIn	2012	117 million	90%	Unsalted SHA-1
Adobe	2013	36 million	??	ECB encryption
YAHOO!	2014	~500 million	??	bcrypt + ??
ASHLEY MADISON Life is short. Have an affair.	2015	36 million	33%	Salted bcrypt + MD5

Source: Tom Ristenpart

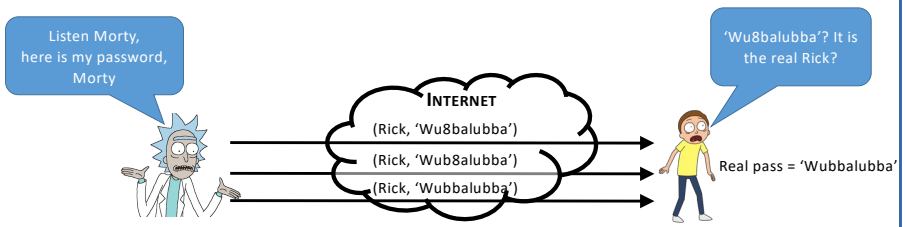
35

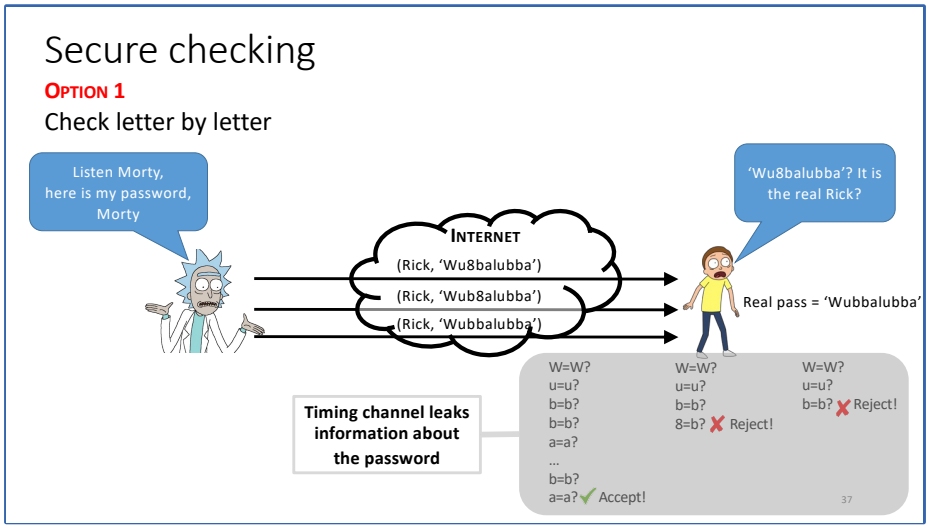
Also, there are leaks…

Secure checking

OPTION 1

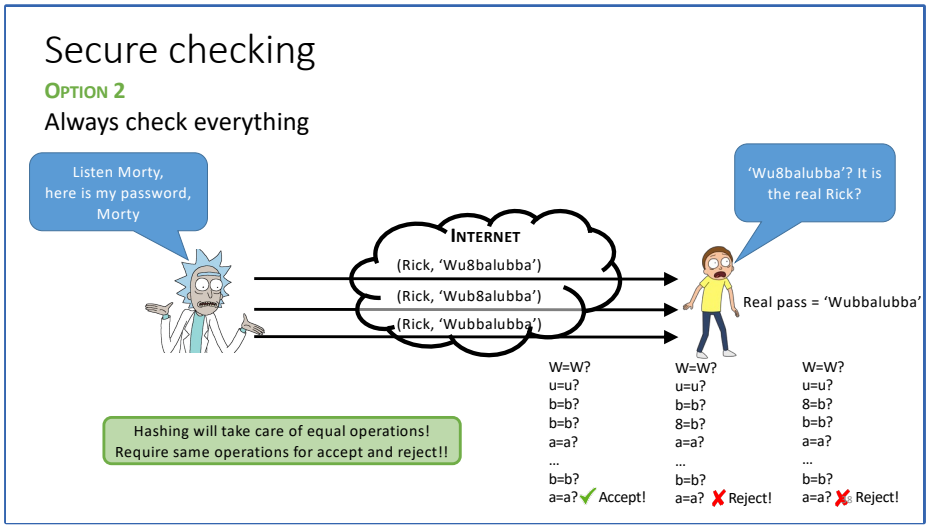
Check letter by letter





When checking passwords or PINs, the time taken to check and reject may leak information on the password.

If you check letter by letter, when the check ends reveals how many letters are correct.



So you should always try to take the same time regardless of when it fails.

(Note that with hashing this happens by default!)

Do not program your own password checker, there are many authentication libraries out there!

(Oauth is very convenient, but implies to leak information to third parties about which users log into your service and when)

Problems with passwords

Strong passwords are difficult to remember

- Written passwords
- Reuse across systems

Problems with passwords

Strong passwords are difficult to remember
- Written passwords
- Reuse across systems

Can be stolen
- Keylogger
- Shoulder surfing
- Phishing
- Social engineering

Problems with passwords

Strong passwords are difficult to remember
- Written passwords
- Reuse across systems

Can be stolen
- Keylogger
- Shoulder surfing
- Phishing
- Social engineering

Jul 6, 2017, 10:10am

Help! Hackers Stole My Password Just By Listening To Me Type On Skype!

Thomas Brewster Forbes Staff
Security
I cover crime, privacy and security in digital and physical forms.

For many, everyday life involves sitting in front of a computer typing endless emails, presentation documents and reports. Then there's the frequent typing of passwords just to get access to those files. But beware: researchers have hacked together a tool that can harvest what's being typed simply by listening to the sounds of the keys.

They've created the Skype&Type program for snooping on Skype

Ways to Prove Who You Are

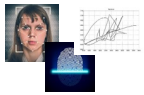
TRADITIONAL

What you know

password, secret key

What you are

biometrics



What you have

Smart card, secure tokens

43

Traditionally, we considered three ways of proving your identity:

- What you know: a secret that only you can know, and therefore proving knowledge of the secret ensures that it is you
- What you are: a trait that is inherent to you and no-one else has, and therefore proving this trait ensures that it is you
- What you have: an object that only you can have, and therefore proving possession ensures that it is you

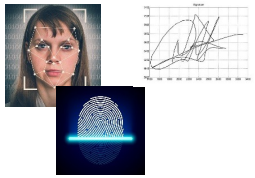
In modern life, the traditional means are augmented by other traits are often unique (some times called soft biometrics):

- where you are: applications detect your common locations and trigger alarms if you are far
- how you act: how you type, how you move the mouse, how you pressure the keyboard
- who are your friends: social networks tend to be unique

What you are: Biometrics

BIOMETRICS
is the measurement and statistical analysis of people's unique physical characteristics *(modern: also behavioral)*

Popular biometrics
Fingerprint, face recognition, retina, voice, handwritten signature, DNA



44

To avoid having to remember, another authentication mean is **Biometrics**. These are unique physical characteristics that can be extracted and measured such that they can be used as a "secret" for authentication.

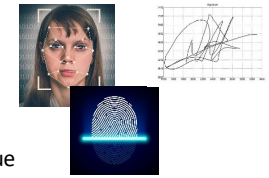
Many of them do not require users to act, they can be checked passively

They are also very difficult to delegate, as they are physical.

What you are: Biometrics

BIOMETRICS
is the measurement and statistical analysis of people's unique physical characteristics *(modern: also behavioral)*

Popular biometrics
- Fingerprint, face recognition, retina, voice, handwritten signature, DNA



Advantages
- Nothing to remember
- Passive
- Difficult to delegate
- If the algorithm is very accurate, they are unique

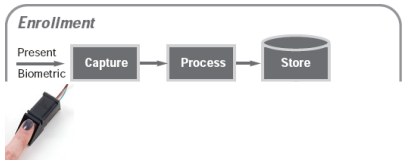
45

To avoid having to remember, another authentication mean is **Biometrics**. These are unique physical characteristics that can be extracted and measured such that they can be used as a "secret" for authentication.

Many of them do not require users to act, they can be checked passively

They are also very difficult to delegate, as they are physical.

Biometrics authentication: 1) Enrollment

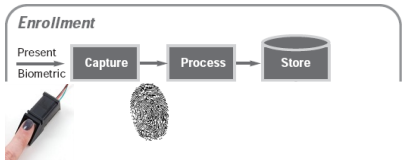


46

The biometric authentication process has two phases. The first one is **Enrollment**, which is when the biometric of the user is introduced in the system and associated to their login.

The biometric is captured by a sensor (fingerprint reader, camera, etc), and processed to convert it into a stream of bits. This stream is called biometric template.

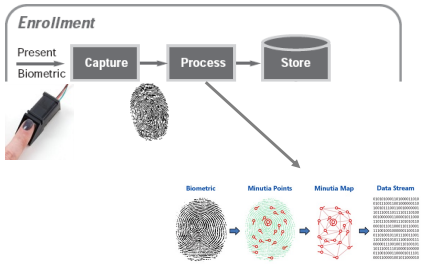
Biometrics authentication: 1) Enrollment



The biometric authentication process has two phases. The first one is **Enrollment**, which is when the biometric of the user is introduced in the system and associated to their login.

The biometric is captured by a sensor (fingerprint reader, camera, etc), and processed to convert it into a stream of bits. This stream is called biometric template.

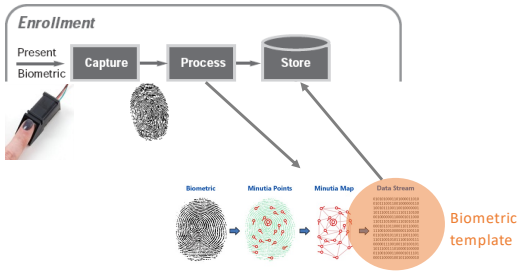
Biometrics authentication: 1) Enrollment



The biometric authentication process has two phases. The first one is **Enrollment**, which is when the biometric of the user is introduced in the system and associated to their login.

The biometric is captured by a sensor (fingerprint reader, camera, etc), and processed to convert it into a stream of bits. This stream is called biometric template.

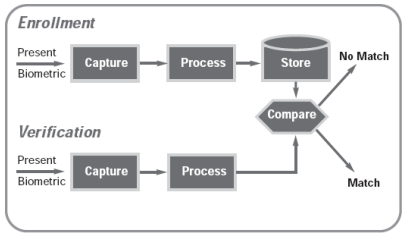
Biometrics authentication: 1) Enrollment



The biometric authentication process has two phases. The first one is **Enrollment**, which is when the biometric of the user is introduced in the system and associated to their login.

The biometric is captured by a sensor (fingerprint reader, camera, etc), and processed to convert it into a stream of bits. This stream is called biometric template.

Biometrics authentication: 2) Verification



The second phase is **Verification.** Here the biometric is captured and processed as before, and the template obtained is compared to the stored one.

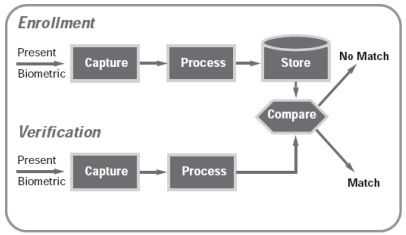
Depending on which the following three phases happen we different security and privacy trade-offs

Capture: the sensor captures the biometric
Process: the captured biometric is converted on the template
Store: the template is stored

- Local: in the device where the assets that the biometrics protect are stored
- Remote: on a server remote from the assets being protected

Storing/processing locally is more privacy-preserving, but harder to secure and difficult to update.
Storing/processing remotely is less privacy-friendly (the server sees the biometrics), but is easy to secure and update.

Biometrics authentication: 2) Verification



WHERE DO THESE PROCESSES HAPPEN?

CAPTURE	PROCESS	STORE
Local	Local	Local
Local	Local	Remote
Local	Remote	Remote

51

The second phase is **Verification.** Here the biometric is captured and processed as before, and the template obtained is compared to the stored one.

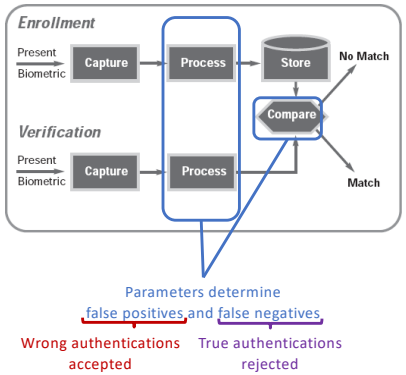
Depending on which the following three phases happen we different security and privacy trade-offs

Capture: the sensor captures the biometric
Process: the captured biometric is converted on the template
Store: the template is stored

- Local: in the device where the assets that the biometrics protect are stored
- Remote: on a server remote from the assets being protected

Storing/processing locally is more privacy-preserving, but harder to secure and difficult to update.
Storing/processing remotely is less privacy-friendly (the server sees the biometrics), but is easy to secure and update.

What you are: Biometrics



Decreasing false negatives increases false positives!!

Configuration depends on applications

Bank: low false positive even if legitimate users need to repeat

Gym: low false negative even if some non-users get in

52

As opposed to the hash in passwords, this comparison *is not exact*. Because the capture may contain differences (e.g. dirt of fingers, light when taking photos, etc), one has to decide a threshold to define what a match means.

When deciding this threshold one has to make a balance between false positives (how many times a biometrics that is not from the user is considered a match) and false negatives (how many times the user provides her biometrics, but these are rejected). What balance depends on the assets you are securing and what is more important, to keep them safe or to keep users happy.

Computer Security (COM-301)

Authentication

Tokens

Carmela Troncoso
SPRING Lab
carmela.troncoso@epfl.ch

Some slides/ideas adapted from: Tuomas Aura, Yoshi Kohno, Trent Jaeger

Ways to Prove Who You Are

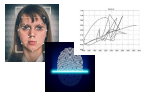
TRADITIONAL

What you know

password, secret key

What you are

biometrics



What you have

Smart card, secure tokens

54

Traditionally, we considered three ways of proving your identity:

- What you know: a secret that only you can know, and therefore proving knowledge of the secret ensures that it is you
- What you are: a trait that is inherent to you and no-one else has, and therefore proving this trait ensures that it is you
- What you have: an object that only you can have, and therefore proving possession ensures that it is you

In modern life, the traditional means are augmented by other traits are often unique (some times called soft biometrics):

- where you are: applications detect your common locations and trigger alarms if you are far
- how you act: how you type, how you move the mouse, how you pressure the keyboard
- who are your friends: social networks tend to be unique

Problems with Biometrics

Hard to keep secret

- Signature on ID card
- Fingerprint left on glasses, door handle, ...
- Photos (nowadays, everywhere!)

} Liveness detection

Revocation is difficult (impossible?)

Sorry, your iris has been compromised, please create a new one... !

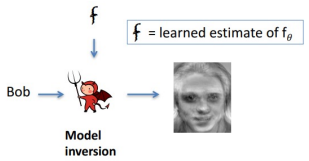
Identifiable and unique

Linking across systems

May reveal private information

Iris → disease

Face → identity



Source: Tom Ristenpart

Not always universal or immutable

Fingerprints disappear, iris changes with lenses,...

55

Liveness detection are measures in the capture/processing steps in which the users are required to do actions to demonstrate they are actually alive.

Uniqueness is problematic, as if the biometric is stolen, it may be used elsewhere. Also, once stolen the templates may reveal attributes.

It is a lie that they do not change.

What you have: Tokens

56

Finally, we also have authentication by proving ownership of a token.

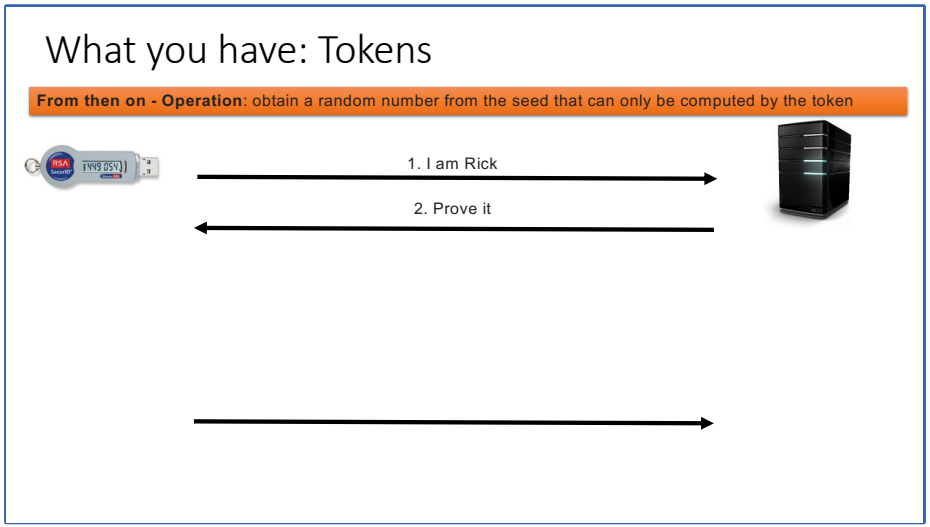
This proof can come by having a chip running a protocol on a secret. For instance, a smart card signs a challenge sent by the ATM to prove knowledge of a key.

Or from a device that contains a secret shared with the server. These devices output a number based on this secret and time (require synchronization). The device is authenticated when it outputs the same number as the server.

What you have: Tokens

Step 1 – Offline - Initialization: token and server establish a common “seed” & synchronize their clocks

“seed” = common random number



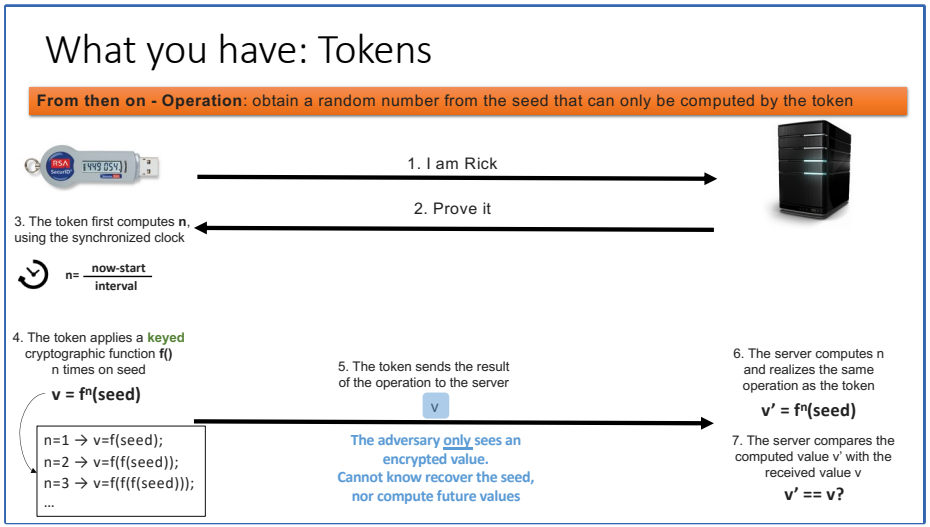
Consider discrete time intervals: 1,2,3,4,.....n-1,n,n+1,.....

When Rick wants to prove identity

The server asks to prove, the token computes which interval it is with repect to the time it was synchronized with the server.

Then applies a keyed cryptographic function using as key the secret shared with the server **n** times on the seed, and sends the result to the server. Notice that from this value an adversary cannot recover the seed or the key (properties of encryption), and cannot compute future values because she does not have the key.

The server does the same operation and checkes



Consider discrete time intervals: 1,2,3,4,.....n-1,n,n+1,.....

When Rick wants to prove identity

The server asks to prove, the token computes which interval it is with repect to the time it was synchronized with the server.

Then applies a keyed cryptographic function using as key the secret shared with the server **n** times on the seed, and sends the result to the server. Notice that from this value an adversary cannot recover the seed or the key (properties of encryption), and cannot compute future values because she does not have the key.

The server does the same operation and checkes

Why the cryptographic function cannot be a hash

From then on - Operation: obtain a random number from the seed that can only be computed by the token

$v_n = h^n(seed)$

I am Rick →

← Prove it

v_n →

61

Why a keyed function and not a hash?

Recall that everyone can compute a hash! If we use a hash instead of a keyed function given v I can produce any future v by hashing the value!

Why the cryptographic function cannot be a hash

From then on - Operation: obtain a random number from the seed that can only be computed by the token

$v_n = h^n(seed)$

I am Rick →

← Prove it

v_n →

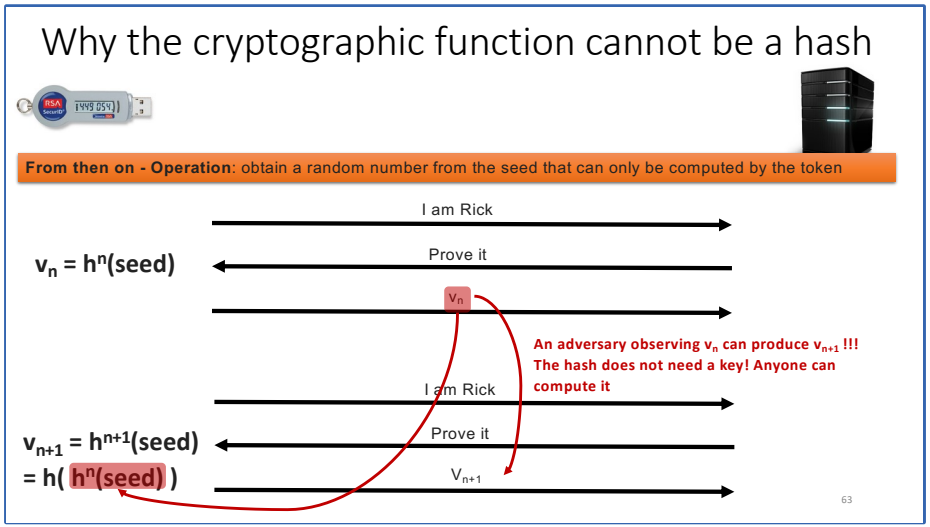
I am Rick →

← Prove it

62

Why a keyed function and not a hash?

Recall that everyone can compute a hash! If we use a hash instead of a keyed function given v I can produce any future v by hashing the value!



Why a keyed function and not a hash?

Recall that everyone can compute a hash! If we use a hash instead of a keyed function given v I can produce any future v by hashing the value!

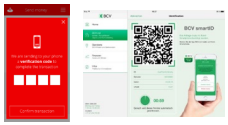
What you have: 2FA – Two factor authentication

Combine two out of the three factors:
(What you know, what you have, what you are)

what you have
what you know

what you have
what you know

what you have
what you know



what you have
what you know

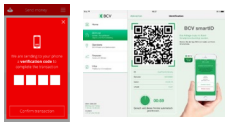
What you have: 2FA – Two factor authentication

Combine two out of the three factors:
(What you know, what you have, what you are)

Card = what you have
+ PIN = what you know

Token = what you have
Identification number = what you know

Token = what you have
(+ Card = what you have)
+ identification number = what you know



Modern approaches: mobile phone = what you have
The phone cannot hold a key (is not secure). Prove via SMS or showing a QR code

What machines have: Secret key

Use secret keys to produce **Digital signatures** to authenticate parties
e.g., used in internet protocols HTTPS/TLS to authenticate **the server**
(and can be used also to authenticate the client)

Finally, when we authenticate machines and not users, this is done using public key cryptography (e.g., to authenticate web servers). The authenticating party signs with their secret key.

What machines have: Secret key

Use secret keys to produce **Digital signatures** to authenticate parties

e.g., used in internet protocols HTTPS/TLS to authenticate **the server**

(and can be used also to authenticate the client)

Building authentication protocols **is hard**!

defending from **man in the middle – Use signatures**

defending from **replay attacks – Use challenges / nonces**

Finally, when we authenticate machines and not users, this is done using public key cryptography (e.g., to authenticate web servers). The authenticating party signs with their secret key.

What machines have: Secret key

Use secret keys to produce **Digital signatures** to authenticate parties

e.g., used in internet protocols HTTPS/TLS to authenticate **the server**

(and can be used also to authenticate the client)

Building authentication protocols **is hard**!

defending from **man in the middle – Use signatures**

defending from **replay attacks – Use challenges / nonces**

Still difficult to get right!

Use well established protocols!! (TLS 1.3, ISO 9798-3)

68

Finally, when we authenticate machines and not users, this is done using public key cryptography (e.g., to authenticate web servers). The authenticating party signs with their secret key.

Summary of the lecture

Authentication is the process by which an entity proves its identity

Three flavours:

- What you know: passwords – hard to handle!
- What you are: biometrics – difficult to revoke and not infallible
- What you have: tokens – usability issues (you need to have the device

Machines authenticate using keys