CS 489 / 698: Software and Systems Security

Module 3: Operating System Security
authentication

Meng Xu *(University of Waterloo)*
Spring 2023

Outline

1. Introduction to authentication
2. Password — the user perspective
3. Password — the protocol-design perspective
4. Other alternatives for authentication

Why this topic?

Q: Recap: what does an operating system do?

A: Resource sharing — An operating system (OS) allows different "entities" to access different resources in a shared way.

- OS makes resources available to entities **if** required by them and **when** permitted by some policy (and availability).
  - What is a resource?
  - What is an entity?
  - How does an entity request for a resource?
  - How does a policy gets specified?
  - How is the policy enforced?

Why this topic?

Q: Recap: what does an operating system do?

A: Resource sharing — An operating system (OS) allows different "entities" to access different resources in a shared way.

- OS makes resources available to entities **if** required by them and **when** permitted by some policy (and availability).
  - What is a resource?
  - What is an entity?
  - How does an entity request for a resource?
  - How does a policy gets specified?
  - How is the policy enforced?

All based on the requirement that:

- an entity can correctly identify itself **AND**,
- the OS can correctly authenticate the entity.

Authentication for different entities

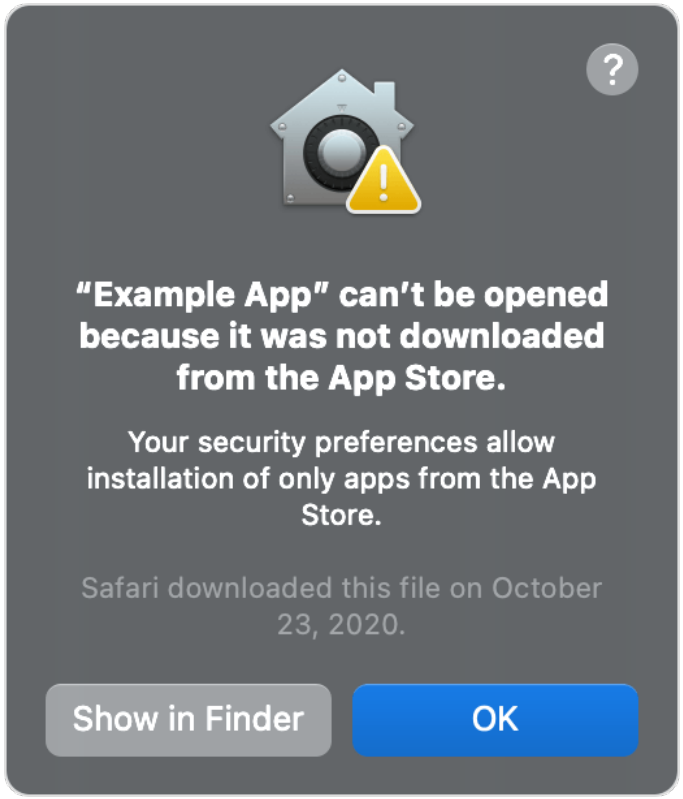
- User authentication
  - Something we all know
- Program authentication
  - Something you might have seen
- Process authentication
  - *What does this even mean?*

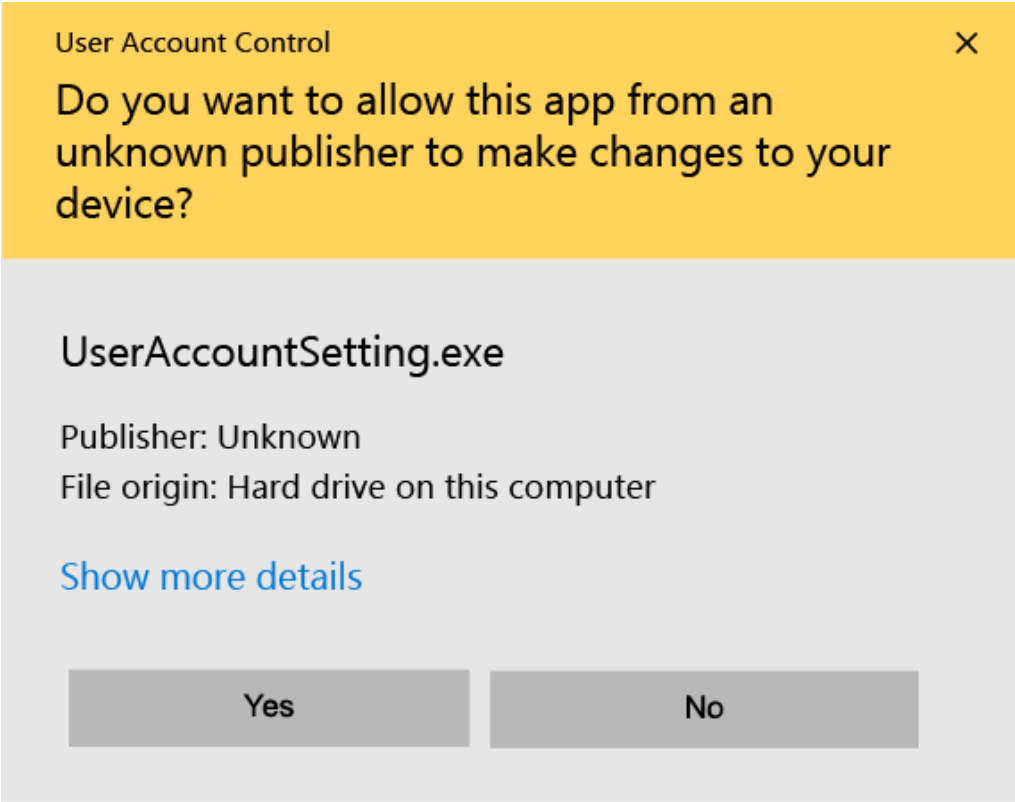
Program authentication

Goal: prove to the operating system (or to the end user) that the program originates from a trusted source and is unmodified.

Program authentication

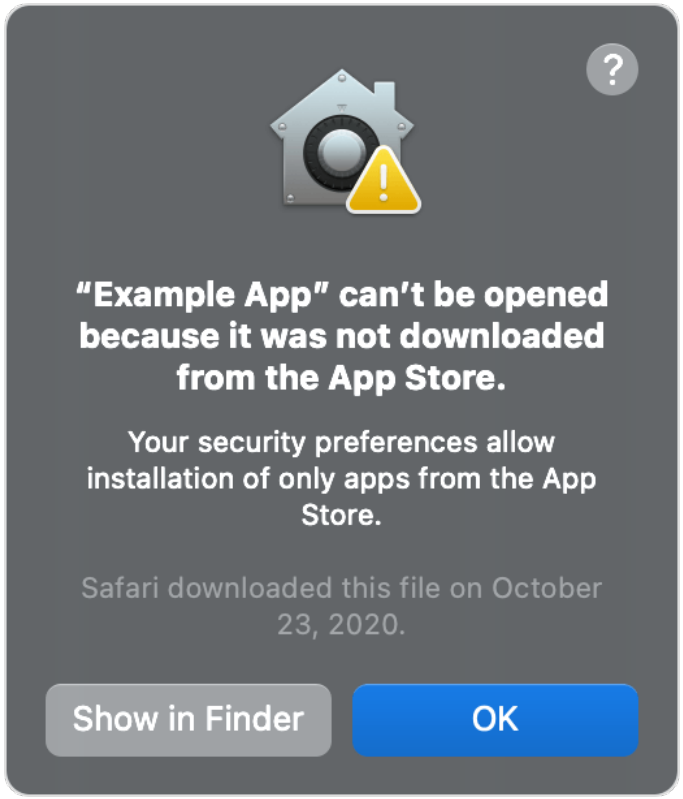
Goal: prove to the operating system (or to the end user) that the program originates from a trusted source and is unmodified.

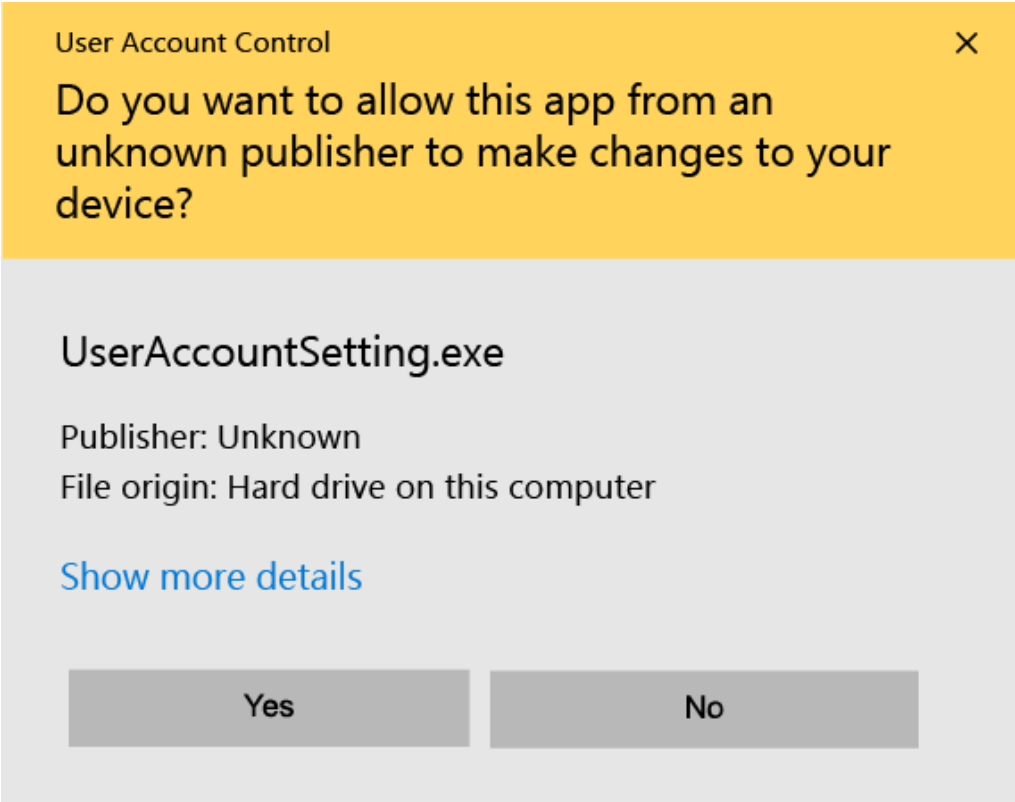




Program authentication

Goal: prove to the operating system (or to the end user) that the program originates from a trusted source and is unmodified.





Typically done via public key infrastructure (PKI) (covered later)

Process authentication

Goal: prove to the operating system that the running process is indeed originated from the program it claims to be.

Process authentication

Goal: prove to the operating system that the running process is indeed originated from the program it claims to be.

For example, if a malicious program hides itself with path "`/bin/chrome.exe`" and claims to be Chrome, at runtime, it needs to attest to the operating system (once at launch or periodically while running) that it indeed has some secret only Chrome knows.

Process authentication

Goal: prove to the operating system that the running process is indeed originated from the program it claims to be.

For example, if a malicious program hides itself with path "`/bin/chrome.exe`" and claims to be Chrome, at runtime, it needs to attest to the operating system (once at launch or periodically while running) that it indeed has some secret only Chrome knows.

Disclaimer: The concept just comes from my effort on systematizing the knowledge. It is not well-defined nor generally accepted and I haven't seen an actual adoption.

The closest academic work I can find is Process Authentication for High System Assurance published in IEEE TDSC 2014. At the core is a challenge-response protocol, which will be covered later.

User authentication

Goal: prove to the operating system that the user is indeed who he/she/they claims to be.

User authentication

Goal: prove to the operating system that the user is indeed who he/she/they claims to be.

- Authentication is easy among people that know each other
  - For your friends, you do it based on their face or voice
- More difficult for computers to authenticate people sitting in front of them
- Even more difficult for computers to authenticate people accessing them remotely

Authentication factors

- Something the user knows
  - Password, PIN, answer to "secret question"
- Something the user has
  - ATM card, badge, browser cookie, physical key, uniform, smartphone
- Something the user is
  - Biometrics (fingerprint, voice pattern, face,...)
  - Have been used by humans forever, but only recently by computers

Authentication factors

- Something the user knows
  - Password, PIN, answer to "secret question"
- Something the user has
  - ATM card, badge, browser cookie, physical key, uniform, smartphone
- Something the user is
  - Biometrics (fingerprint, voice pattern, face,. . .)
  - Have been used by humans forever, but only recently by computers

Authentication should also be aware of user's context, e.g., location, time, devices in proximity, etc.

Multi-factor authentication (MFA)

Different classes of authentication factors can be combined for more secure authentication.

- bank card + PIN
- password + SMS

Multi-factor authentication (MFA)

Different classes of authentication factors can be combined for more secure authentication.

- bank card + PIN
- password + SMS

However, using multiple factors from the same class might not provide better authentication.

- password + PIN

SIM-based MFA

Caveat about SIM-based authentication:

SMS (or phone call) is an approximation of "something you have", a phone number, or more specifically, a SIM card. But if it is implemented by checking routability of a SMS message or call, it can be subverted by an attacker who *does NOT* have the phone, e.g., via SIM-jacking or SS7 attacks.

SIM-based MFA

Caveat about SIM-based authentication:

SMS (or phone call) is an approximation of "something you have", a phone number, or more specifically, a SIM card. But if it is implemented by checking routability of a SMS message or call, it can be subverted by an attacker who *does NOT* have the phone, e.g., via SIM-jacking or SS7 attacks.

Alternatives?

SIM-based MFA

Caveat about SIM-based authentication:

SMS (or phone call) is an approximation of "something you have", a phone number, or more specifically, a SIM card. But if it is implemented by checking routability of a SMS message or call, it can be subverted by an attacker who *does NOT* have the phone, e.g., via SIM-jacking or SS7 attacks.

Alternatives?

- Authenticator apps
  - vulnerable to malware on the phone
  - vulnerable to loss of device
- Separate tokens/fobs
  - vulnerable to loss of device

Outline

1. Introduction to authentication
2. Password — the user perspective
3. Password — the protocol-design perspective
4. Other alternatives for authentication

Password

Password is probably the oldest authentication mechanism used in computer systems.

Password

Password is probably the oldest authentication mechanism used in computer systems.

- 1936: Alan Turing's paper On Computable Numbers laid the groundwork for what many consider the first modern computers.
- 1948: the first stored-program computer, Manchester Baby, ran its first program.
- 1961: a password program was invented for the Compatible Time-Sharing System (CTSS) in MIT.

Security problems with passwords

If password is the only authentication factor, once the password disclosed to an unauthorized individual, the individual can immediately access the protected resource.

Security problems with passwords

If password is the only authentication factor, once the password disclosed to an unauthorized individual, the individual can immediately access the protected resource.

... and there are too many ways a password can be leaked:

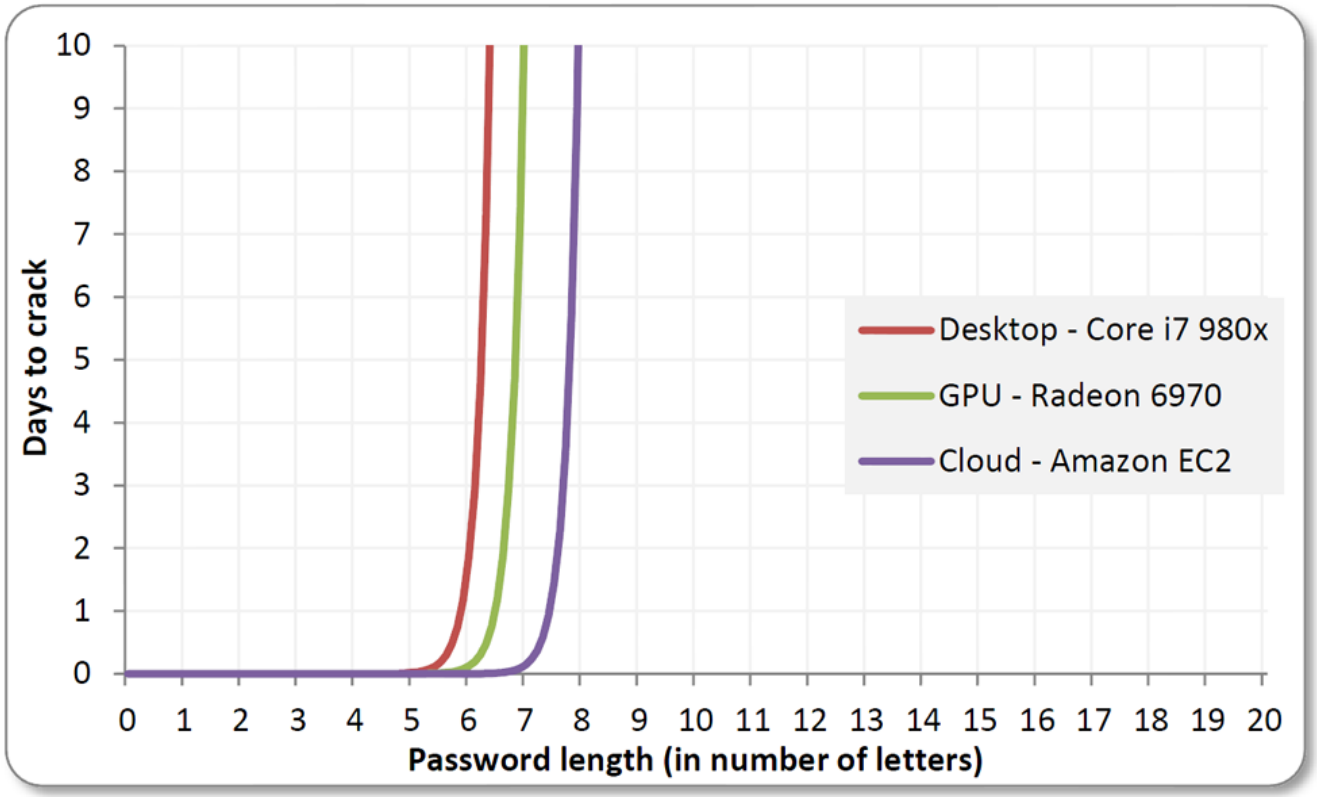
- Shoulder surfing
- Keystroke logging
- Interface illusions / phishing
- Password re-use across sites
- Password guessing

Password guessing attacks

Brute-force: Try all possible passwords using exhaustive search

- Can test http://arstechnica.com/security/2012/12/25-gpu-cluster-cracks-every-standard-windows-password-in-6-hours/ Windows NTLM passwords per second on a cluster of 25 AMD Radeon graphics cards
- Can try 95^8 combinations in 5.5 hours
- Enough to brute force every possible 8-character password containing upper- and lower-case letters, digits, and symbols

Brute-forcing passwords is exponential



Source of image: common misconceptions of password cracking

Password guessing attacks

Exhaustive search assumes that people choose passwords randomly, which is often not the case.

Attacker can do much better by exploiting this.

- For example, assume that a password consists of a **root** and a pre- or postfix **appendage**
  - "password1", "abc123", "123abc"
- **Root** is from dictionaries (passwords from previous password leaks, names, English words, ...)
- **Appendage** is combination of digits, date, single symbol, ...

>90% of 6.5 million LinkedIn password hashes leaked in June 2012 were cracked within six days.

Password hygiene

- Use a password manager to create and store passwords
  - At least for low- and medium-security passwords
  - All (most) eggs are now in one basket, so keep your computer's software up to date
  - Prevents password re-use across sites
- Use a pass phrase
  - Phrase of randomly chosen words, avoid common phrases (e.g., advertisement slogans)
- Don't reveal passwords to others

Advice for developers (NIST 2017)

- No password composition rules
  - e.g., a password contain at least one lowercase letter, uppercase letter, number, and symbol.
  - Otherwise everybody uses the same simple tricks to follow rule
- 8 characters minimum length
- 64 characters maximum length
- Allow any characters, including space, Unicode, and emoji
- Detect and prevent frequently used or compromised passwords (from password leaks)
- Avoid password hints or "secret questions"
- Don't ask users to periodically change passwords
  - Leads to password cycling and similar passwords
    * "myFavoritePwd" -> "dummy" -> "myFavoritePwd"
    * goodPwd."1" -> goodPwd."2" -> goodPwd."3"
- Allow passwords to be copy-pasted into password fields
- Use two-factor authentication
  - but avoid SMS-based second factor

Outline

1. Introduction to authentication
2. Password — the user perspective
3. Password — the protocol-design perspective
4. Other alternatives for authentication

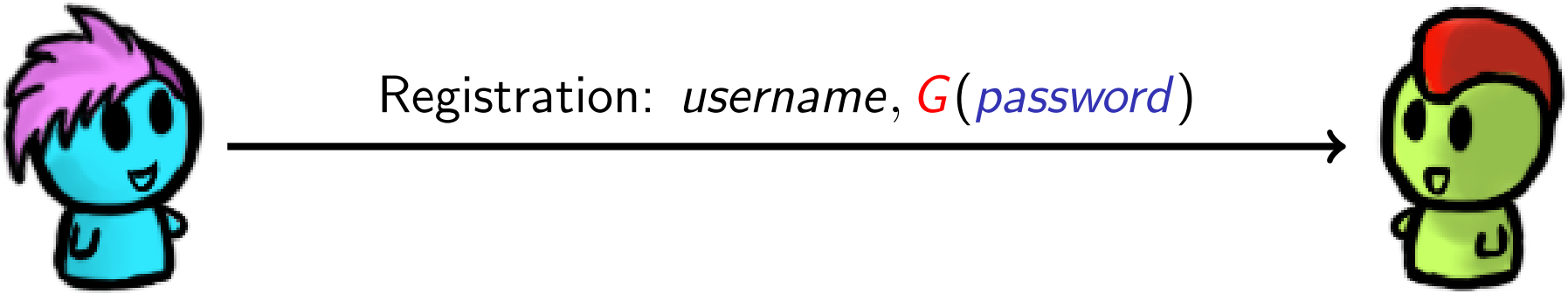
A formal modeling of password

A formal model is useful for examining the pros and cons of several password-based authentication protocols.

A formal modeling of password

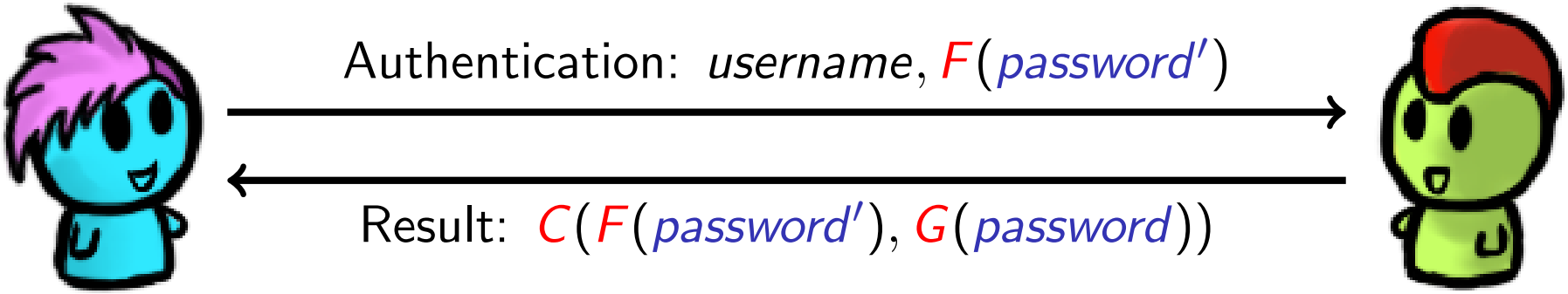
A formal model is useful for examining the pros and cons of several password-based authentication protocols.

User OS



Registration: $\mathit{username}, G(\mathit{password})$ →

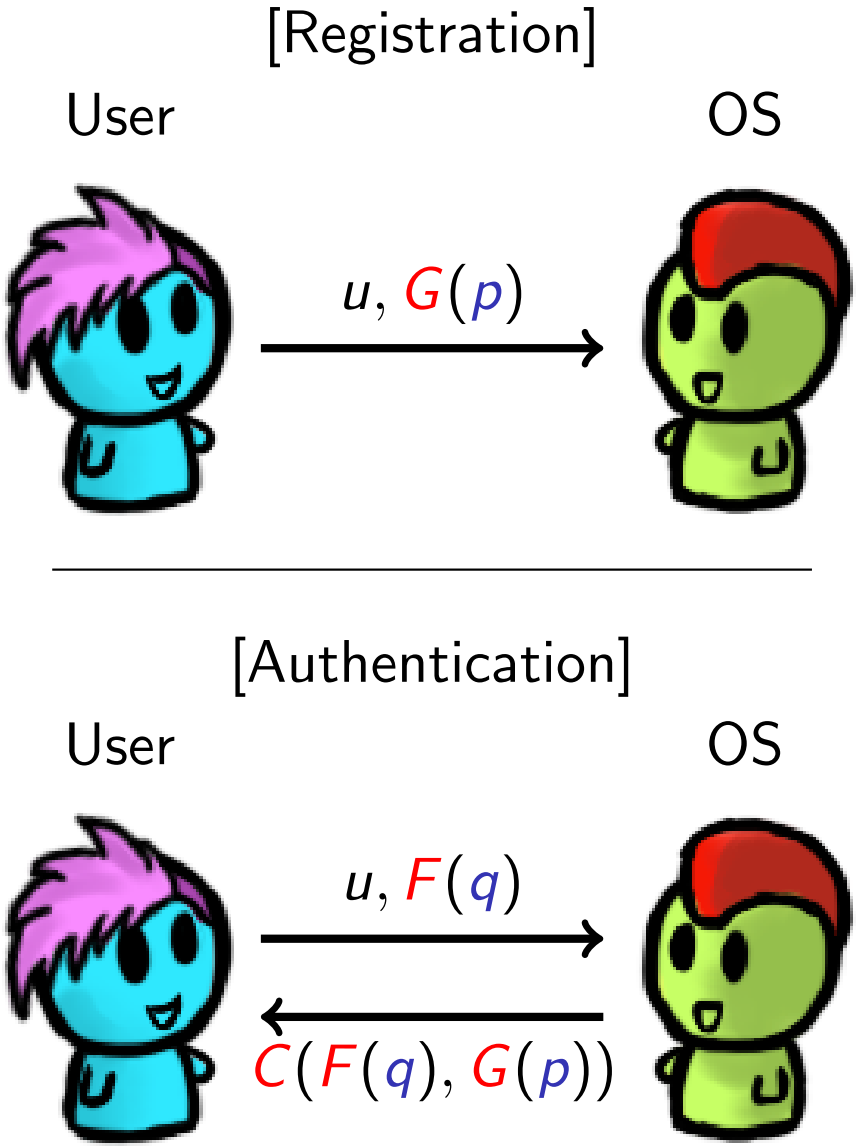
User OS



Authentication: $\mathit{username}, F(\mathit{password}')$ →

← Result: $C(F(\mathit{password}'), G(\mathit{password}))$

Design space

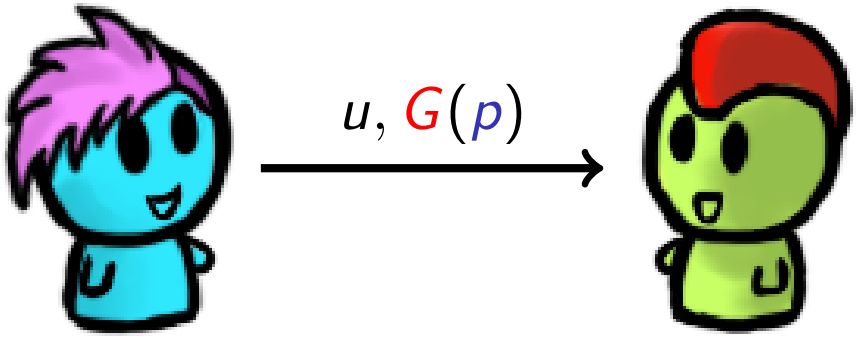


The design space of a password-based authentication protocol is around functions $G(p)$, $F(q)$, and $C(F(q), G(p))$

Design space

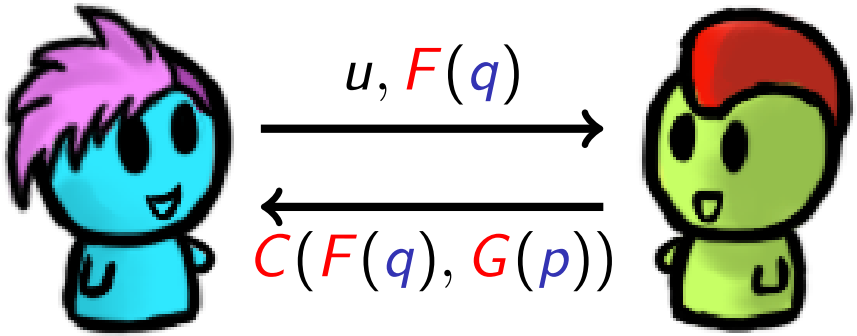
[Registration]

User OS



[Authentication]

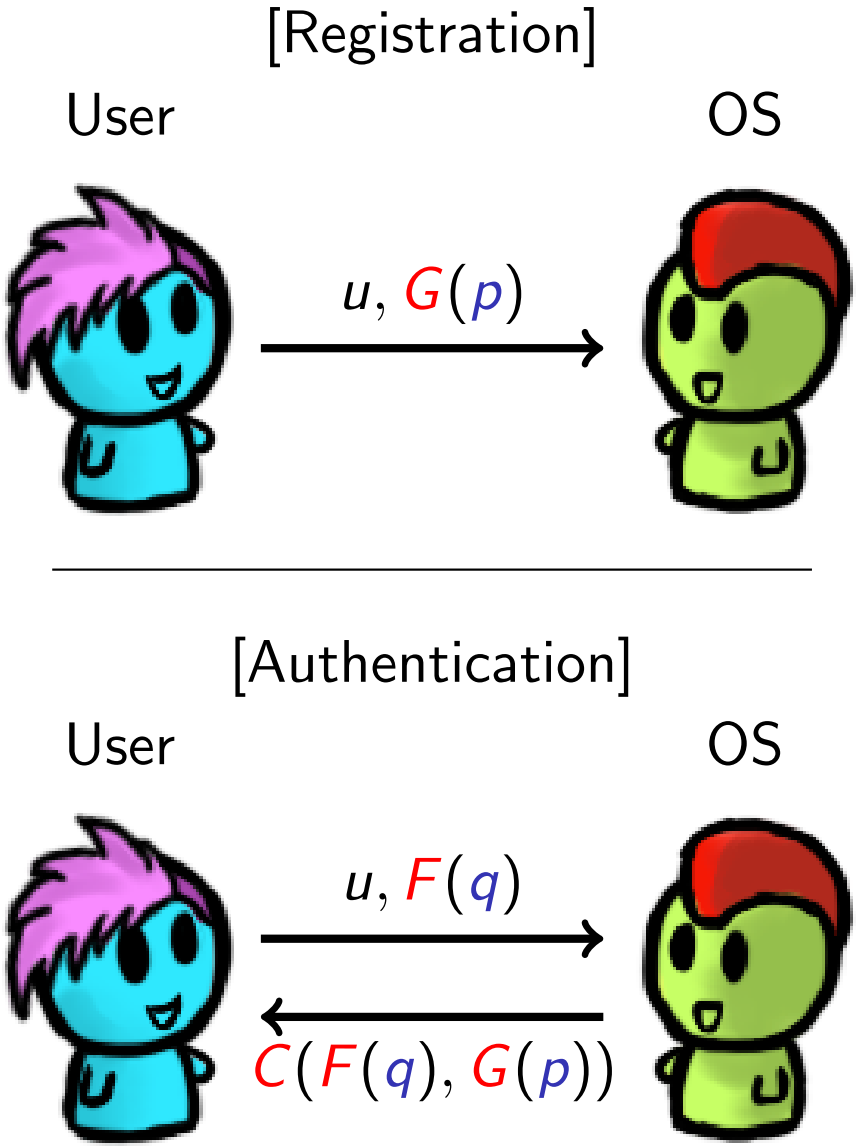
User OS



The design space of a password-based authentication protocol is around functions $G(p)$, $F(q)$, and $C(F(q), G(p))$

Q: What is the correctness requirement of the protocol?

Design space



The design space of a password-based authentication protocol is around functions $G(p)$, $F(q)$, and $C(F(q), G(p))$

Q: What is the correctness requirement of the protocol?

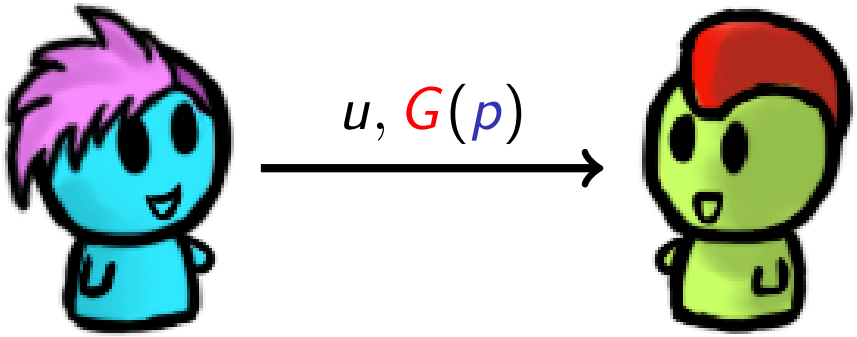
A: Two properties:

- $p = q \implies C(F(q), G(p)) = \mathrm{T}$
- $p \neq q \implies C(F(q), G(p)) = \mathrm{F}$

Design space

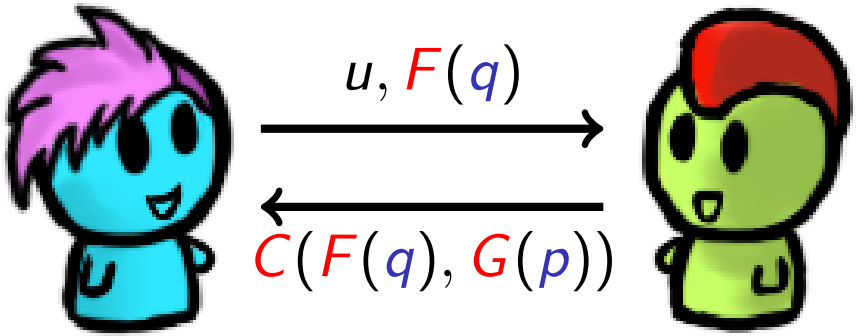
[Registration]

User OS



[Authentication]

User OS



The design space of a password-based authentication protocol is around functions $G(p)$, $F(q)$, and $C(F(q), G(p))$

Q: What is the correctness requirement of the protocol?

A: Two properties:

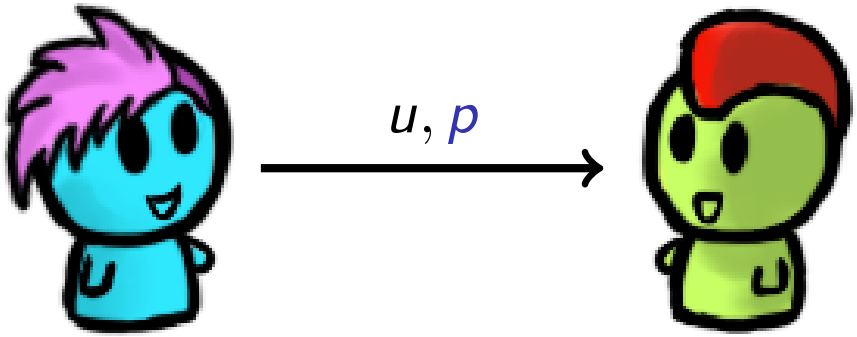
- $p = q \implies C(F(q), G(p)) = \mathrm{T}$
- $p \neq q \implies C(F(q), G(p)) = \mathrm{F}$

Q: Can you design a protocol that satisfies this requirement?

Option 1: plaintext password

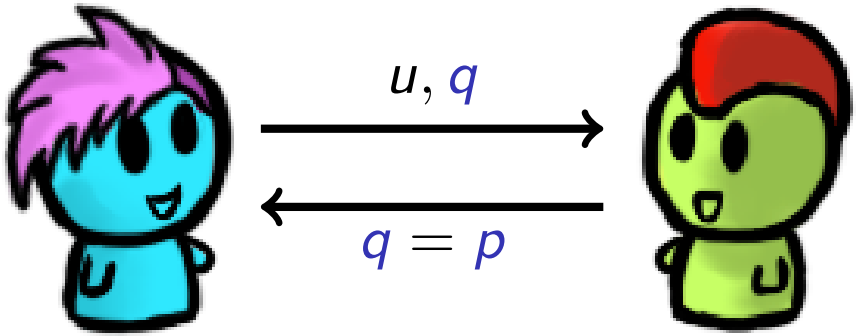
[Registration]

User OS



[Authentication]

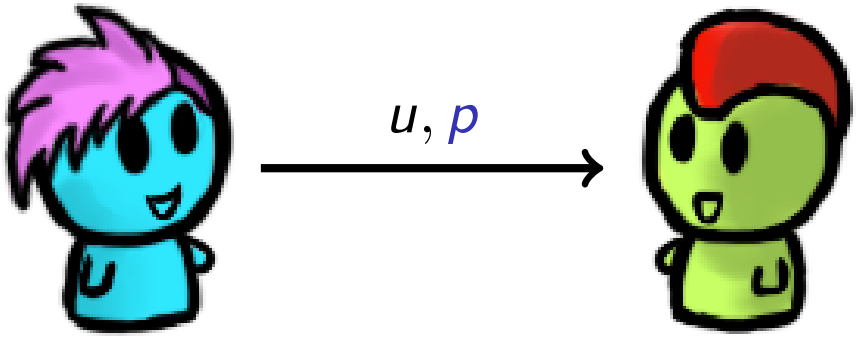
User OS



Option 1: plaintext password

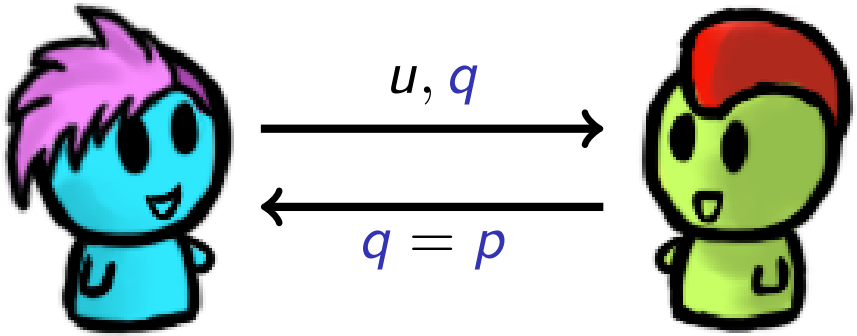
[Registration]

User OS



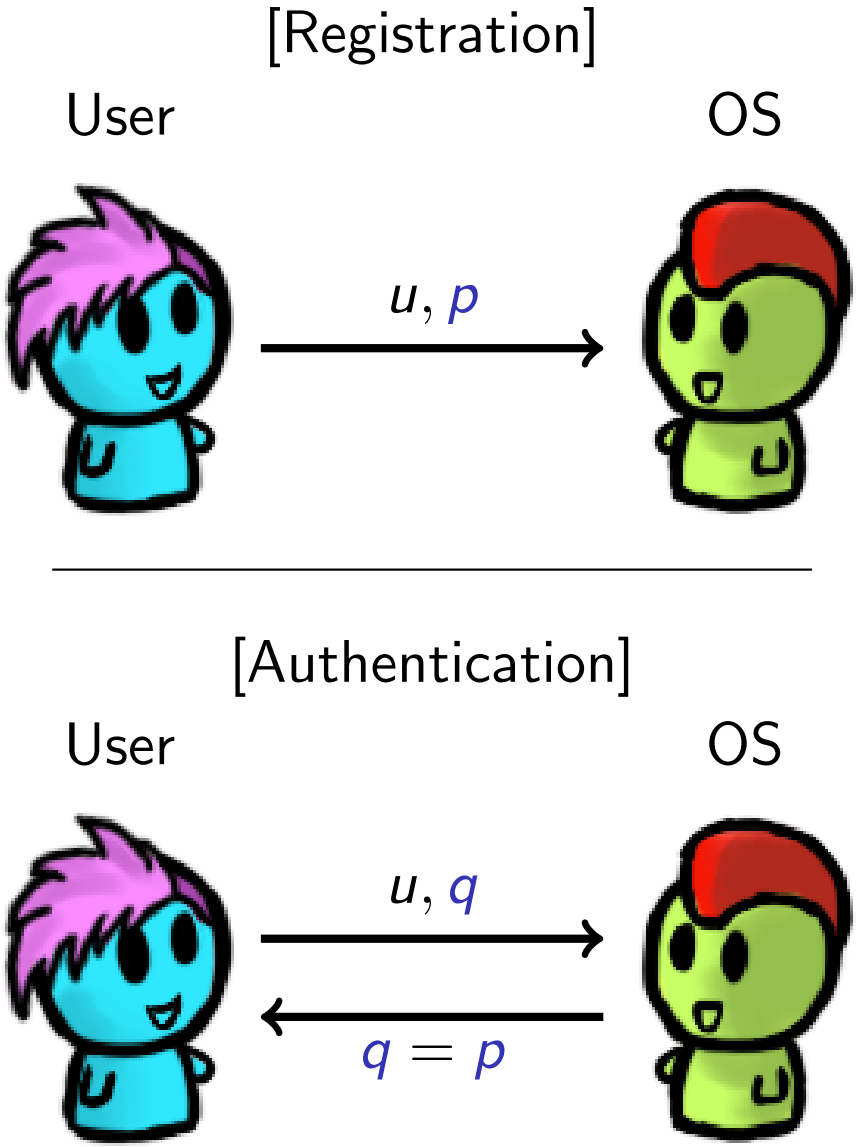
[Authentication]

User OS



Q: What is wrong with this scheme?

Option 1: plaintext password



Q: What is wrong with this scheme?

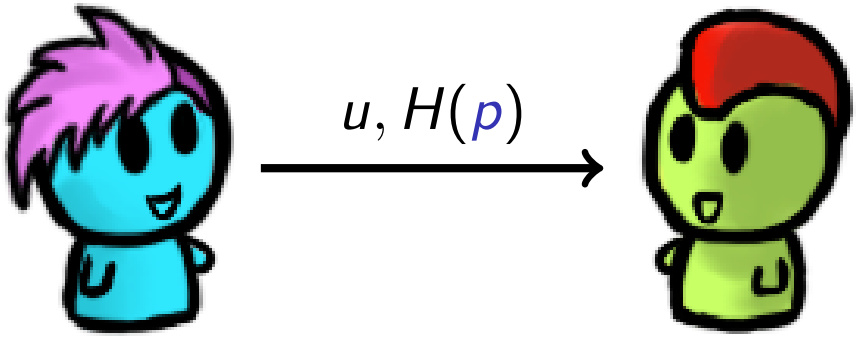
A: Storing passwords in plaintext is extremely dangerous

- Password file might end up on backup tapes
- Intruder into OS might get access to password file
- System administrators have access to the file and might use passwords to impersonate users at other systems
  - Many people re-use passwords across multiple systems

Option 2: password fingerprint

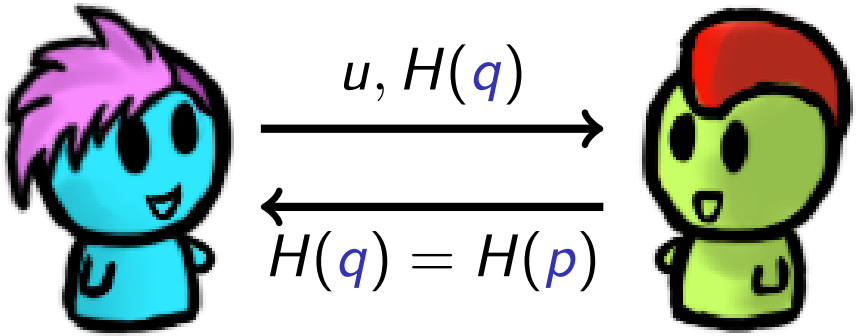
[Registration]

User OS



[Authentication]

User OS



Cryptographic hash function

A hash function h takes an arbitrary length string x and computes a fixed length string $y = h(x)$ called a message digest

- Common examples: MD5, SHA-1, SHA-2, SHA-3 (a.k.a., Keccak, from 2012 on), where MD5 and SHA-1 are not considered safe now.

Cryptographic hash function

A hash function h takes an arbitrary length string x and computes a fixed length string $y = h(x)$ called a message digest

- Common examples: MD5, SHA-1, SHA-2, SHA-3 (a.k.a., Keccak, from 2012 on), where MD5 and SHA-1 are not considered safe now.

A hash function is cryptographically secure if it has three properties

Cryptographic hash function

A hash function h takes an arbitrary length string x and computes a fixed length string $y = h(x)$ called a message digest

- Common examples: MD5, SHA-1, SHA-2, SHA-3 (a.k.a., Keccak, from 2012 on), where MD5 and SHA-1 are not considered safe now.

A hash function is cryptographically secure if it has three properties

1. Preimage-resistance:
   - Given y, it's hard to find x such that $h(x) = y$
     i.e., a "preimage" of y

Cryptographic hash function

A hash function h takes an arbitrary length string x and computes a fixed length string $y = h(x)$ called a message digest

- Common examples: MD5, SHA-1, SHA-2, SHA-3 (a.k.a., Keccak, from 2012 on), where MD5 and SHA-1 are not considered safe now.

A hash function is cryptographically secure if it has three properties

1. Preimage-resistance:
   - Given y, it's hard to find x such that $h(x) = y$
     i.e., a "preimage" of y
2. Second preimage-resistance:
   - Given x, it's hard to find $x' \neq x$ such that $h(x) = h(x')$
     i.e., a "second preimage" of $h(x)$

Cryptographic hash function

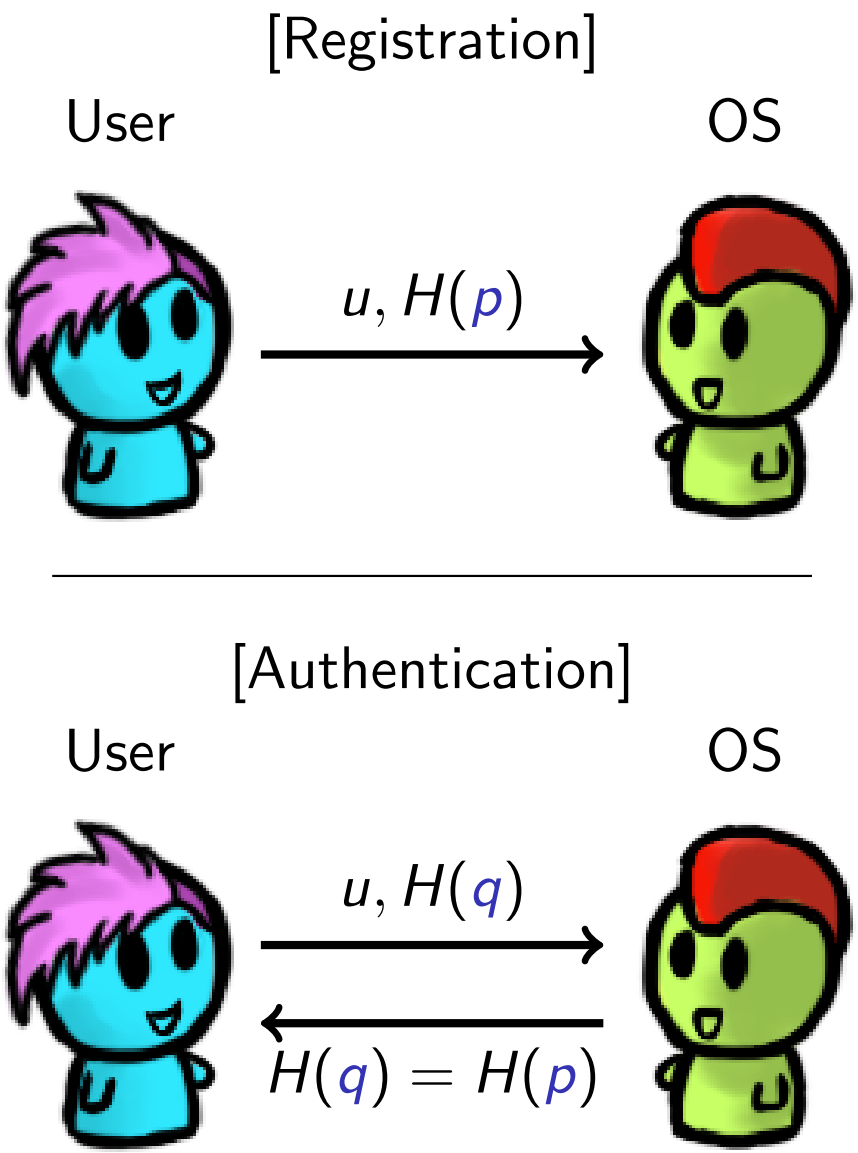
A hash function h takes an arbitrary length string x and computes a fixed length string $y = h(x)$ called a message digest

- Common examples: MD5, SHA-1, SHA-2, SHA-3 (a.k.a., Keccak, from 2012 on), where MD5 and SHA-1 are not considered safe now.

A hash function is cryptographically secure if it has three properties

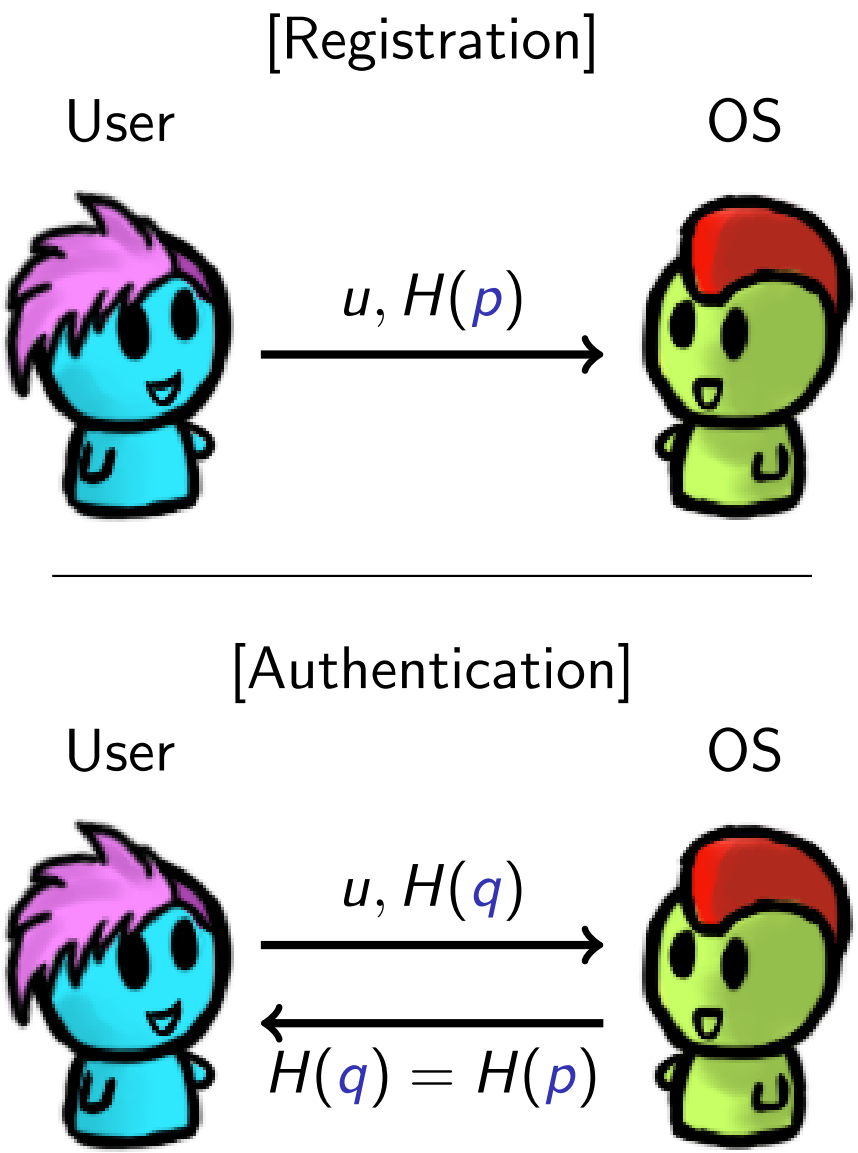
1. Preimage-resistance:
   - Given y, it's hard to find x such that $h(x) = y$
     i.e., a "preimage" of y
2. Second preimage-resistance:
   - Given x, it's hard to find $x' \neq x$ such that $h(x) = h(x')$
     i.e., a "second preimage" of $h(x)$
3. Collision-resistance:
   - It's hard to find any two distinct values x, x' such that $h(x) = h(x')$
     i.e., a "collision"

Option 2: password fingerprint



H is a cryptographic hash function (e.g., SHA-2, SHA-3)

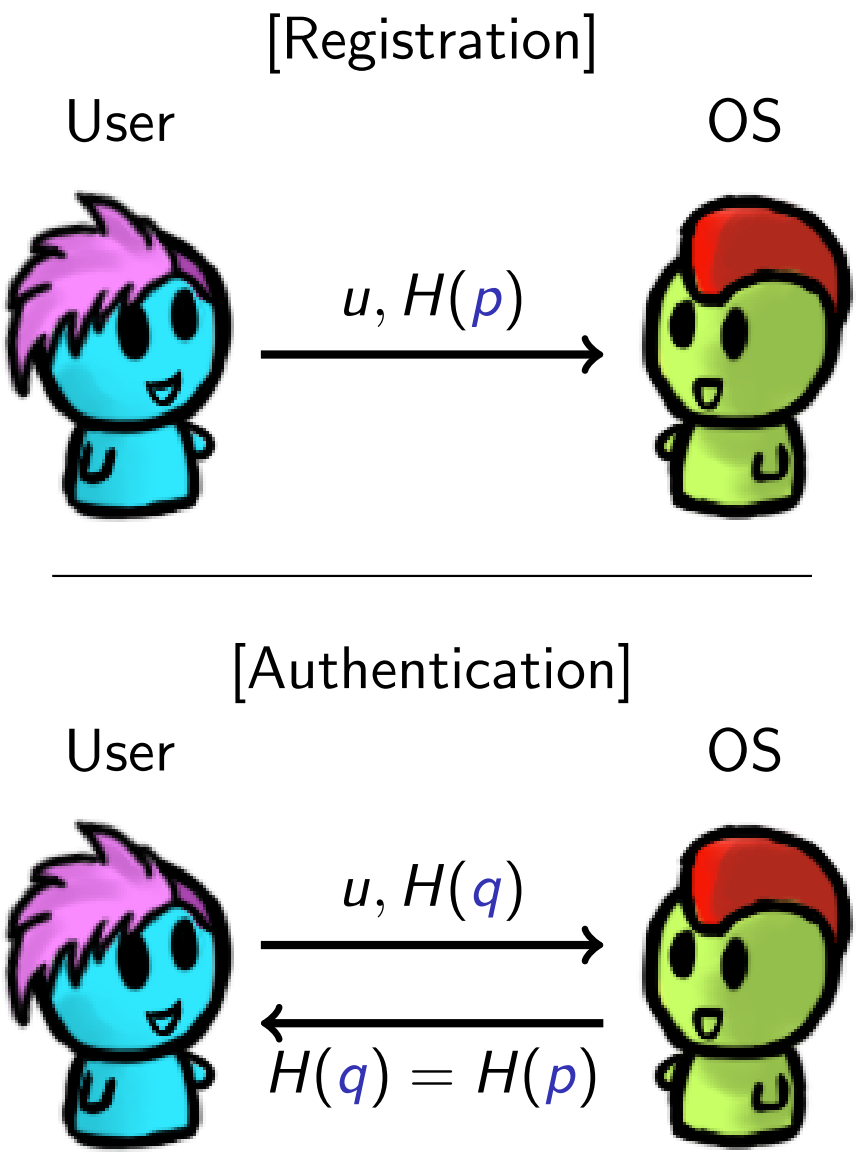
Option 2: password fingerprint



H is a cryptographic hash function (e.g., SHA-2, SHA-3)

Q: Does this protocol satisfy the correctness requirement?

Option 2: password fingerprint



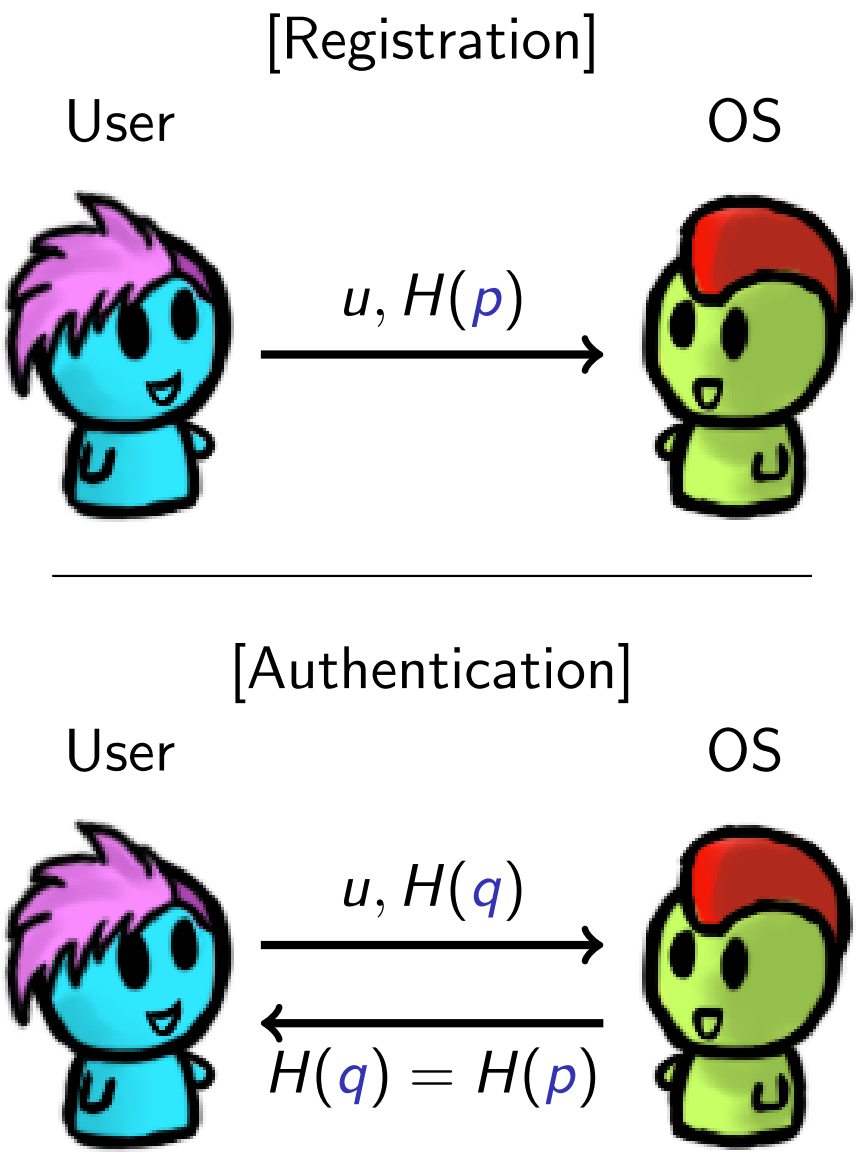
H is a cryptographic hash function (e.g., SHA-2, SHA-3)

Q: Does this protocol satisfy the correctness requirement?

A: Two properties:

- $p = q \implies C(F(q), G(p)) = \mathsf{T}$
- $p \neq q \implies$ $\mathbf{Pr}[C(F(q), G(p)) = \mathsf{T}] < \epsilon$

Option 2: password fingerprint



H is a cryptographic hash function (e.g., SHA-2, SHA-3)

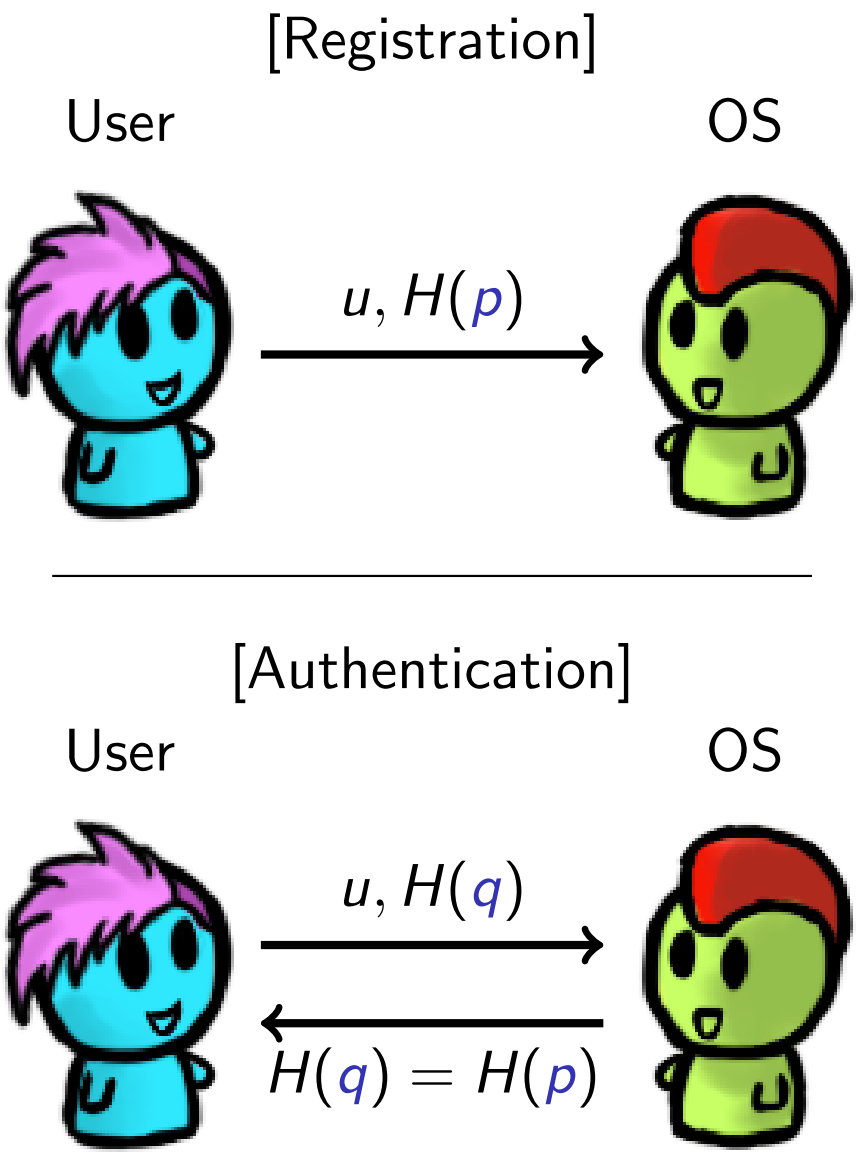
Q: Does this protocol satisfy the correctness requirement?

A: Two properties:

- $p = q \implies C(F(q), G(p)) = \mathsf{T}$
- $p \neq q \implies$ $\mathbf{Pr}[C(F(q), G(p)) = \mathsf{T}] < \epsilon$

Q: What other weaknesses this protocol may have?

Option 2: password fingerprint



H is a cryptographic hash function (e.g., SHA-2, SHA-3)

Q: Does this protocol satisfy the correctness requirement?

A: Two properties:

- $p = q \implies C(F(q), G(p)) = \mathsf{T}$
- $p \neq q \implies$ $\mathbf{Pr}[C(F(q), G(p)) = \mathsf{T}] < \epsilon$

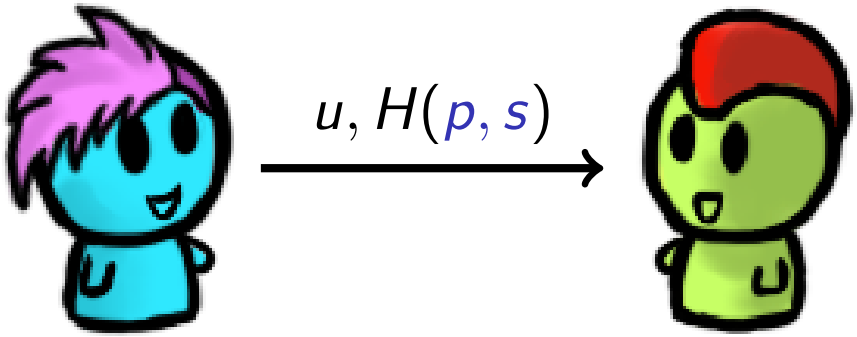
Q: What other weaknesses this protocol may have?

A: Same password, same fingerprint

Option 3a: salted password fingerprint

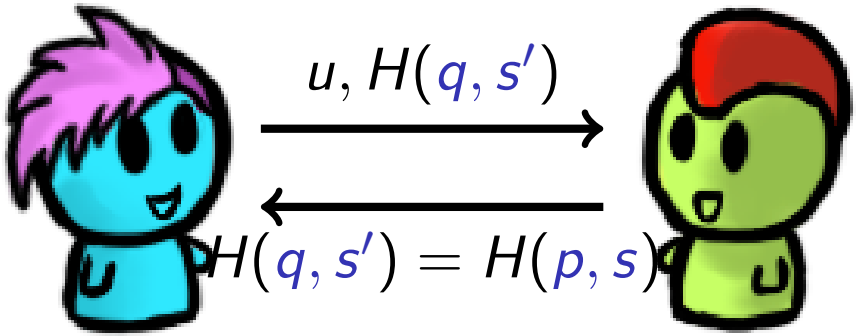
[Registration]

User OS

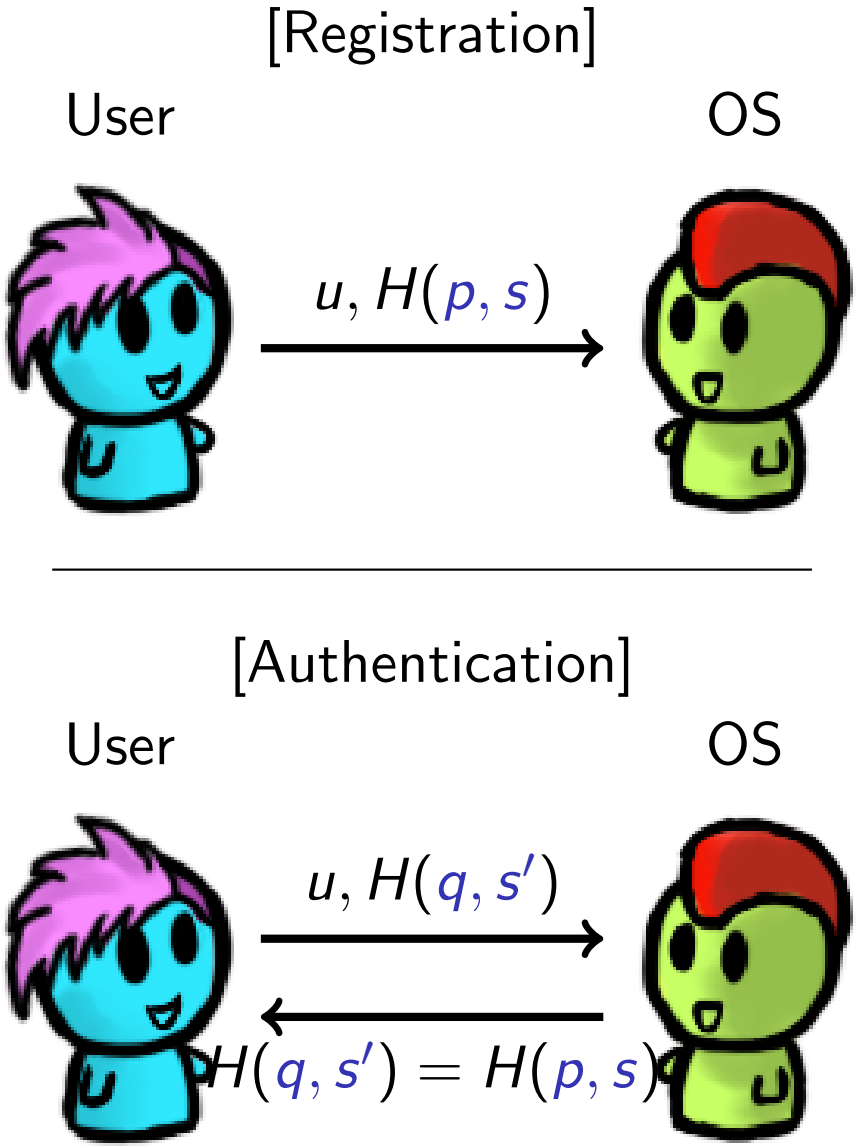


[Authentication]

User OS



Option 3a: salted password fingerprint



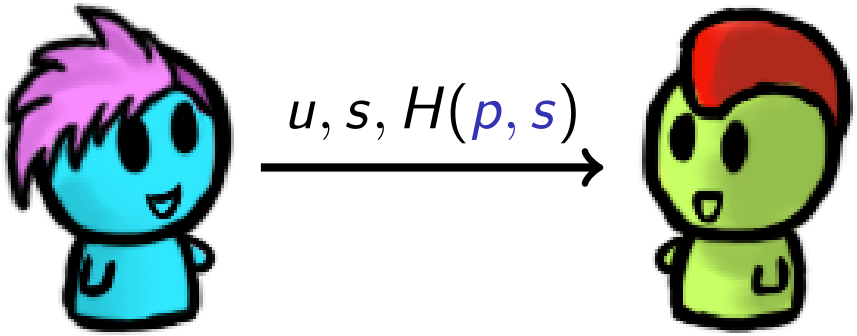
In this scheme, the user (or the client program) is responsible for remembering and managing the salt.

Despite the fact that the salt doesn't have to be secretive, managing it can still be inconvenient.

Option 3b: salted password fingerprint

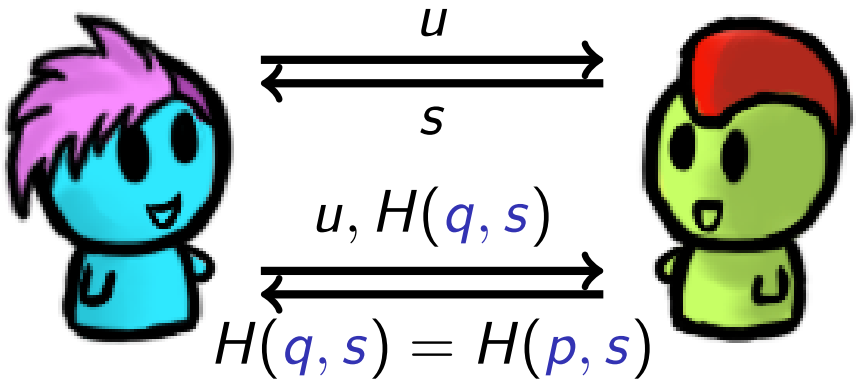
[Registration]

User OS

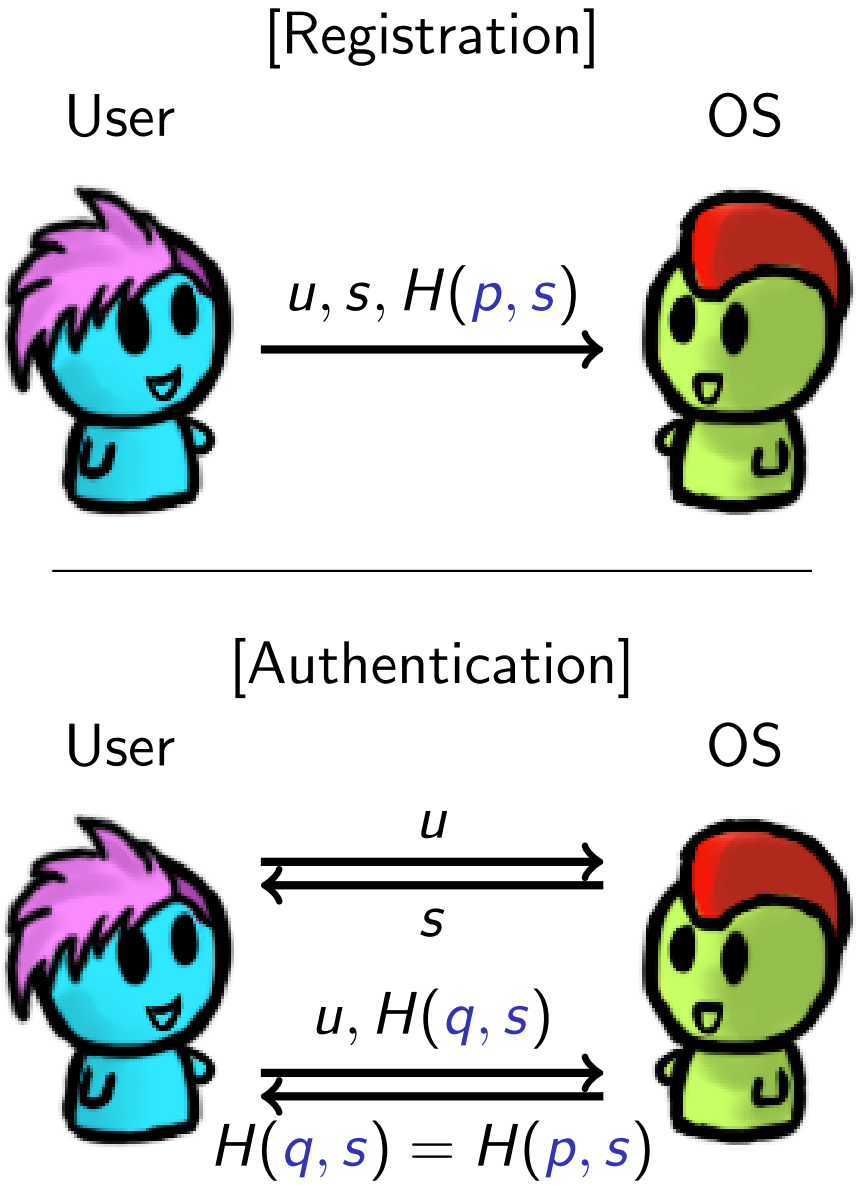


[Authentication]

User OS



Option 3b: salted password fingerprint



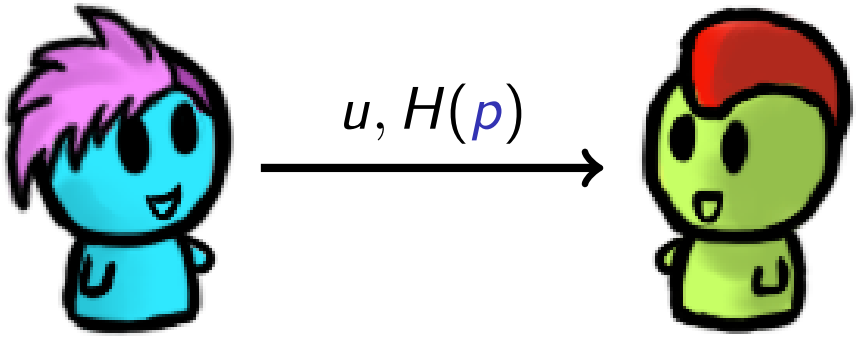
In this scheme, the OS (or the server program) is responsible for remembering and managing the salt.

The downside is that it adds an extra roundtrip in the protocol and may enable user-probing attacks.

Option 3c: salted password fingerprint

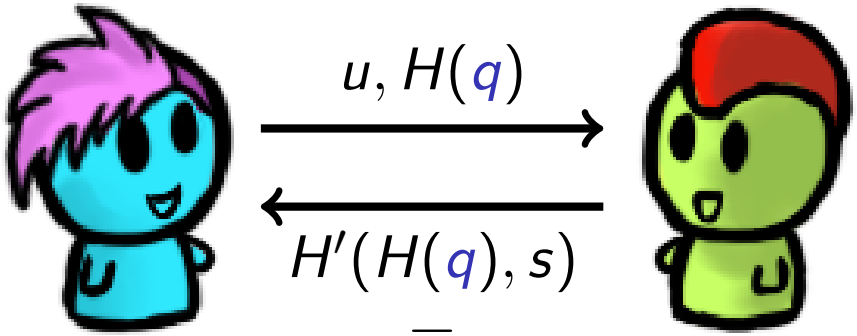
[Registration]

User OS



[Authentication]

User OS

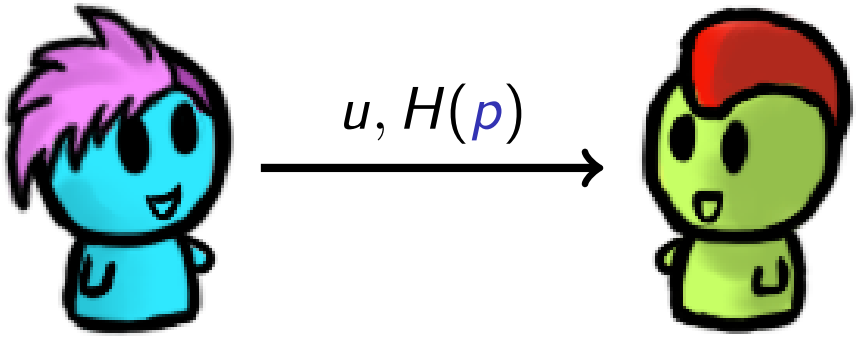


$$= H'(H(p), s)$$

Option 3c: salted password fingerprint

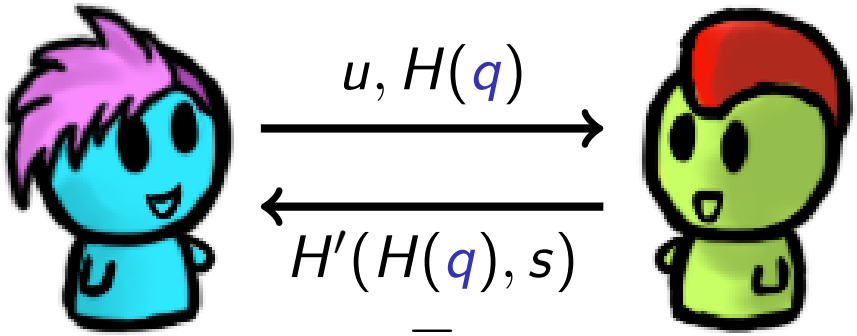
[Registration]

User OS



[Authentication]

User OS



$$= H'(H(p), s)$$

In this scheme, the salt is assigned by the OS and is oblivious to the user.

It prevents offline dictionary attacks when the password file is leaked from the OS (e.g., via breach), but has little protection over eavesdropping attacks over the network.

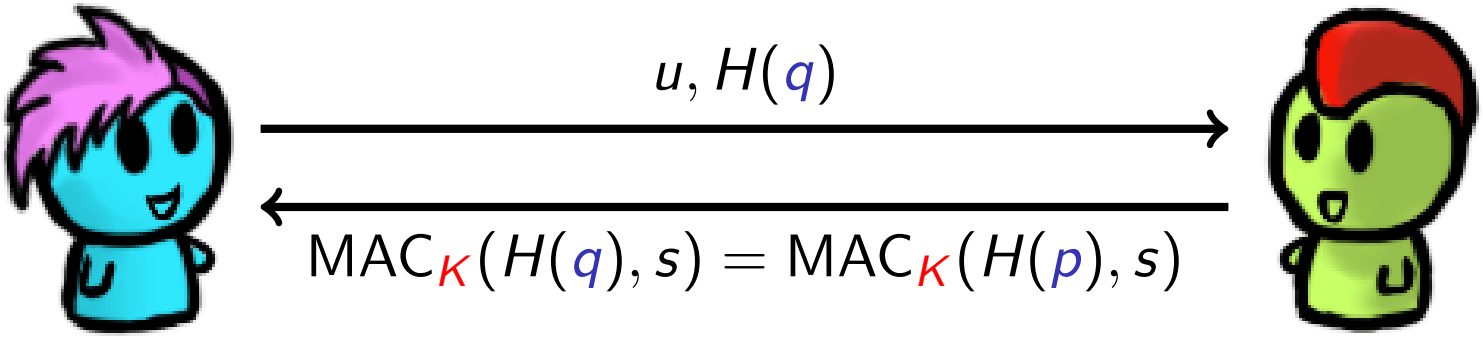
Further protections against offline guessing attacks

- Use expensive iterated hash functions to compute the fingerprint.
  - Standard cryptographic hash (e.g., SHA-2, SHA-3) is relatively cheap to compute (microseconds).
  - Iterated hash functions (e.g., bcrypt, scrypt) can take hundreds of milliseconds and even use a lot memory.
  - This slows down a guessing attack significantly, but is barely noticed in the entire authentication protocol.

Further protections against offline guessing attacks

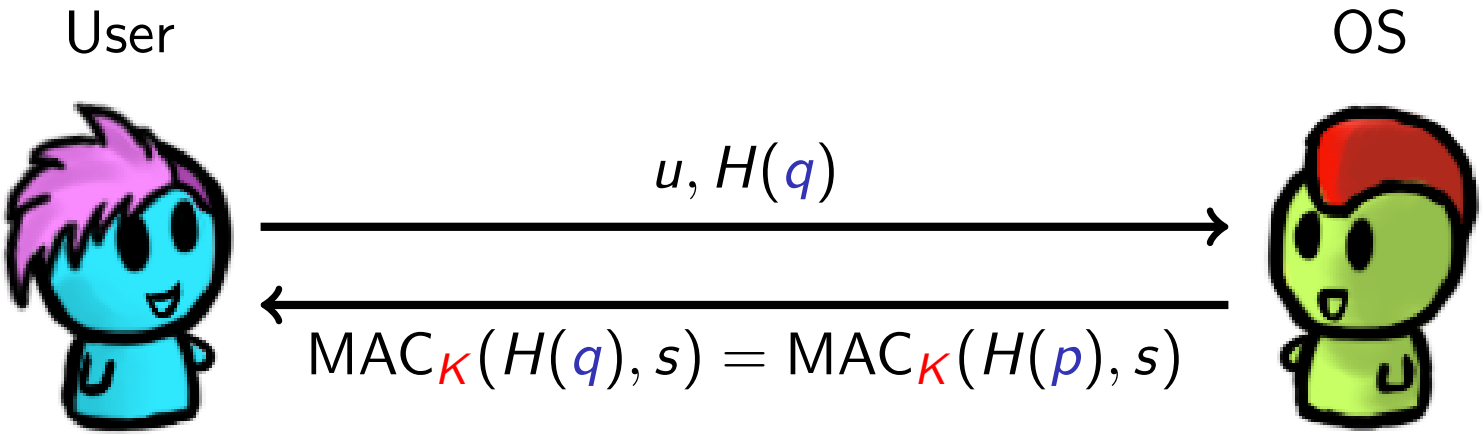
- Use message authentication code (MAC) to calculate a tag.

User OS



Further protections against offline guessing attacks

- Use message authentication code (MAC) to calculate a tag.

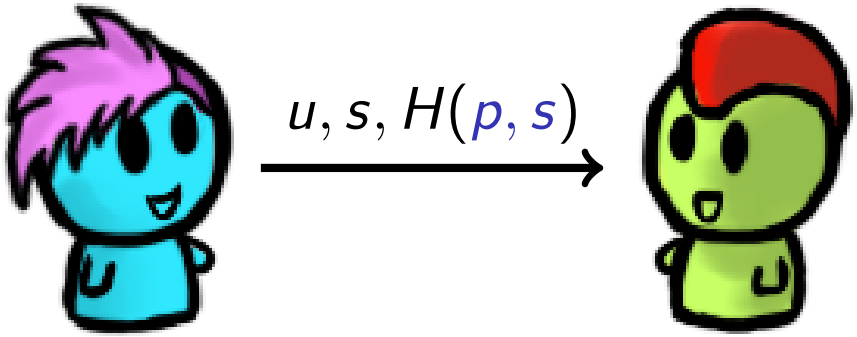


- Protect the secret key by embedding it in tamper-resistant hardware.
- If the key does leak, the scheme remains as secure as a scheme based on a cryptographic hash.

Option 4: challenge-response protocol

[Registration]

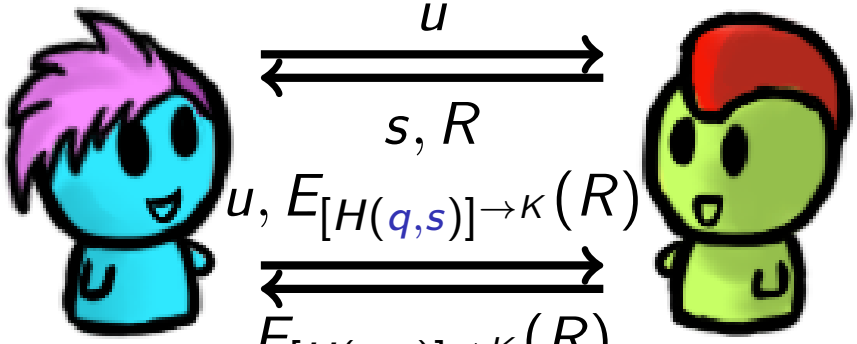
User OS



$$\xrightarrow{u, s, H(p, s)}$$

[Authentication]

User OS



$$\xrightarrow{u}$$

$$\xleftarrow{s, R}$$

$$\xrightarrow{u, E_{[H(q,s)]\to K}(R)}$$

$$\xleftarrow{E_{[H(q,s)]\to K}(R)}$$

$$=$$

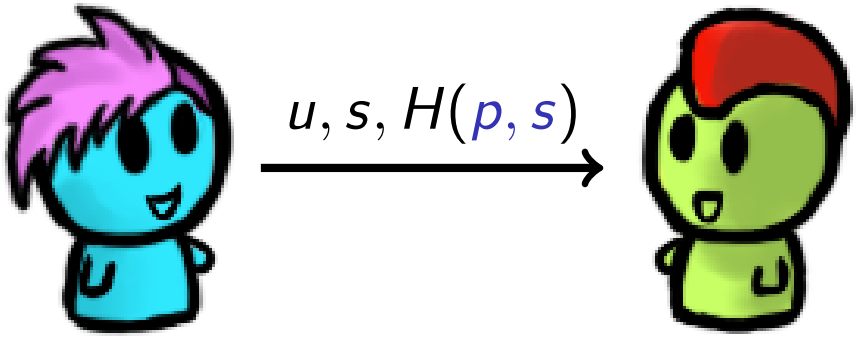
$$E_{[H(p,s)]\to K}(R)$$

Goal: even if the eavesdropper captures all message exchanges over the entire authentication process, it cannot re-compute p (other than brute-forcing).

Option 4: challenge-response protocol

[Registration]

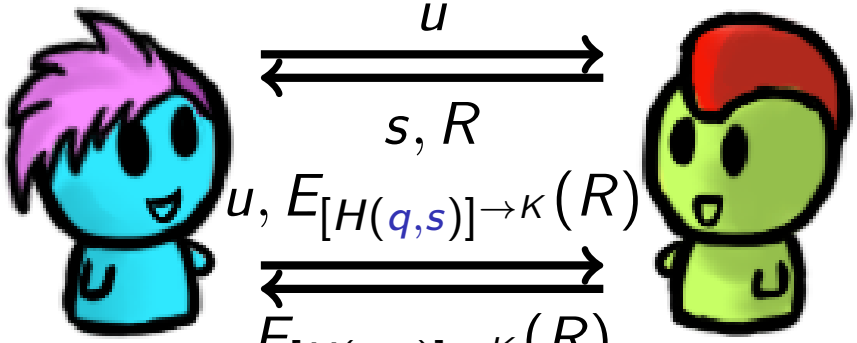
User OS



$$\xrightarrow{u,\, s,\, H(p,s)}$$

[Authentication]

User OS



$$\xrightarrow{u}$$

$$\xleftarrow{s,\, R}$$

$$\xrightarrow{u,\, E_{[H(q,s)]\to K}(R)}$$

$$\xleftarrow{E_{[H(q,s)]\to K}(R)}$$

$$= E_{[H(p,s)]\to K}(R)$$

Goal: even if the eavesdropper captures all message exchanges over the entire authentication process, it cannot re-compute p (other than brute-forcing).

Q: What are the potential problems with this protocol?

Option 4: challenge-response protocol

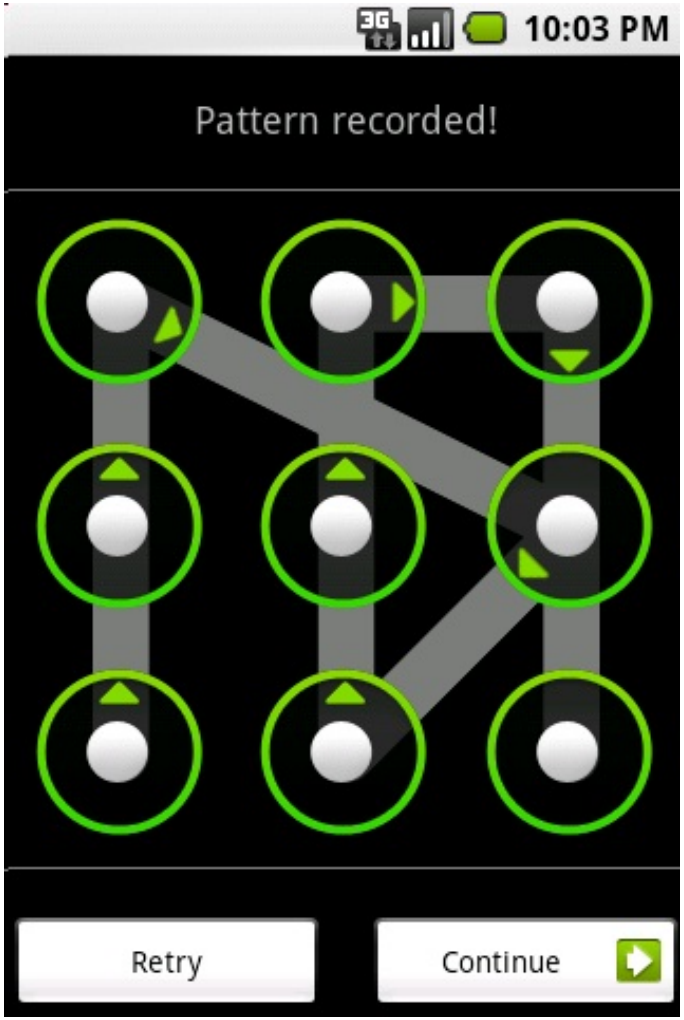
For serious designs of challenge-response protocol, please refer to:

- SCRAM: Salted Challenge Response Authentication Mechanism
- SRP: Secure Remote Password protocol
- OPAQUE: The OPAQUE Asymmetric PAKE Protocol
- SPAKE2+: SPAKE2+, an Augmented PAKE

Outline

1. Introduction to authentication
2. Password — the user perspective
3. Password — the protocol-design perspective
4. Other alternatives for authentication

Unlock patterns



Graphical passwords

- Graphical passwords are an alternative to text-based passwords
- Multiple techniques, e.g.,
  - User chooses a picture; to log in, user has to re-identify this picture in a set of pictures
  - User chooses set of places in a picture; to log in, user has to click on each place
- Issues similar to text-based passwords arise
  - e.g., choice of places is not necessarily random
- Shoulder surfing becomes a problem
- Ongoing research

Graphical passwords illustrated

Pictures adapted from paper Human-Seeded Attacks and Exploiting Hot-Spots in Graphical Passwords published in USENIX Security 2007

Biometrics

Biometrics have been hailed as a way to get rid of the problems with password and token-based authentication

- Idea: Authenticate user based on physical characteristics
  - Fingerprints, iris scan, voice, handwriting, typing pattern, ...
- If observed trait is sufficiently close to previously stored trait, accept user
  - Observed fingerprint will never be completely identical to a previously stored fingerprint of the same user

Biometrics

Biometrics have been hailed as a way to get rid of the problems with password and token-based authentication

- Idea: Authenticate user based on physical characteristics
  - Fingerprints, iris scan, voice, handwriting, typing pattern, ...
- If observed trait is sufficiently close to previously stored trait, accept user
  - Observed fingerprint will never be completely identical to a previously stored fingerprint of the same user

Unfortunately, biometrics have their own problems.

False positives

Face-recognition software with (unrealistic) accuracy of 99.9% is used in a football stadium to detect terrorists.

- 1-in-1,000 chance that a terrorist is not detected
- 1-in-1,000 chance that innocent person is flagged as terrorist

False positives

Face-recognition software with (unrealistic) accuracy of 99.9% is used in a football stadium to detect terrorists.

- 1-in-1,000 chance that a terrorist is not detected
- 1-in-1,000 chance that innocent person is flagged as terrorist

$\Longrightarrow$ If one in 1 million stadium attendees is a known terrorist, there will be 1,000 false alarms.

Remember "The Boy Who Cried Wolf"?

Other problems with biometrics

- Privacy
  - Why should my employer (or a website) have information about my fingerprints, iris,..?
  - What if this information leaks? Getting a new password is easy, but much more difficult for biometrics
- Accuracy: False negatives are annoying
  - What if there is no other way to authenticate?
  - What if I grow a beard, hurt my finger, . . . ?
- Secrecy: Some of your biometrics are not particularly secret
  - Face, fingerprints,...
- Legal protection: The law may allow the police to put your finger on your phone's fingerprint reader (or simply hold your phone's camera in front of you). But the law may protect you from you having to reveal your password (depending on the country).

Other problems with biometrics

SIGN IN / UP The Register®

Carjackers swipe biometric Merc, plus owner's finger

Sometimes you might not want such great security...

John Lettice Mon 4 Apr 2005 // 13:52 UTC

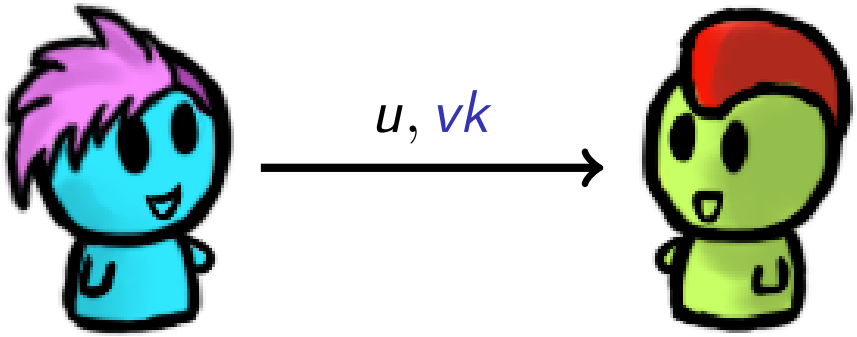
A Malaysian businessman has lost a finger to car thieves impatient to get around his Mercedes' fingerprint security system. Accountant K Kumaran, the BBC reports, had at first been forced to start the S-class Merc, but when the carjackers wanted to start it again without having him along, they chopped off the end of his index finger with a machete.

Source of information can be found in this article

Passkey

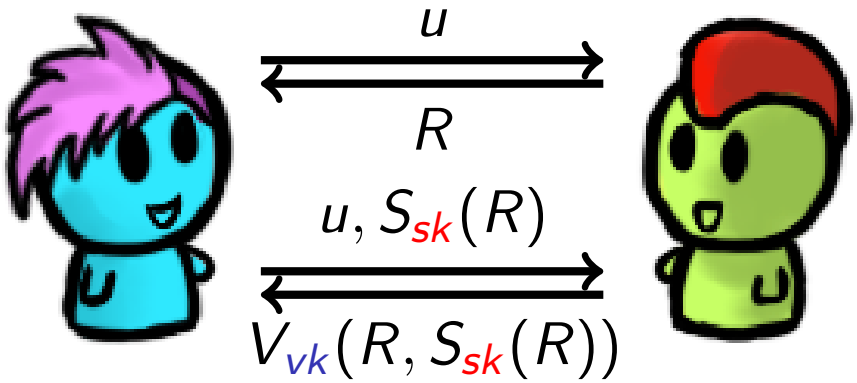
[Registration]

User OS

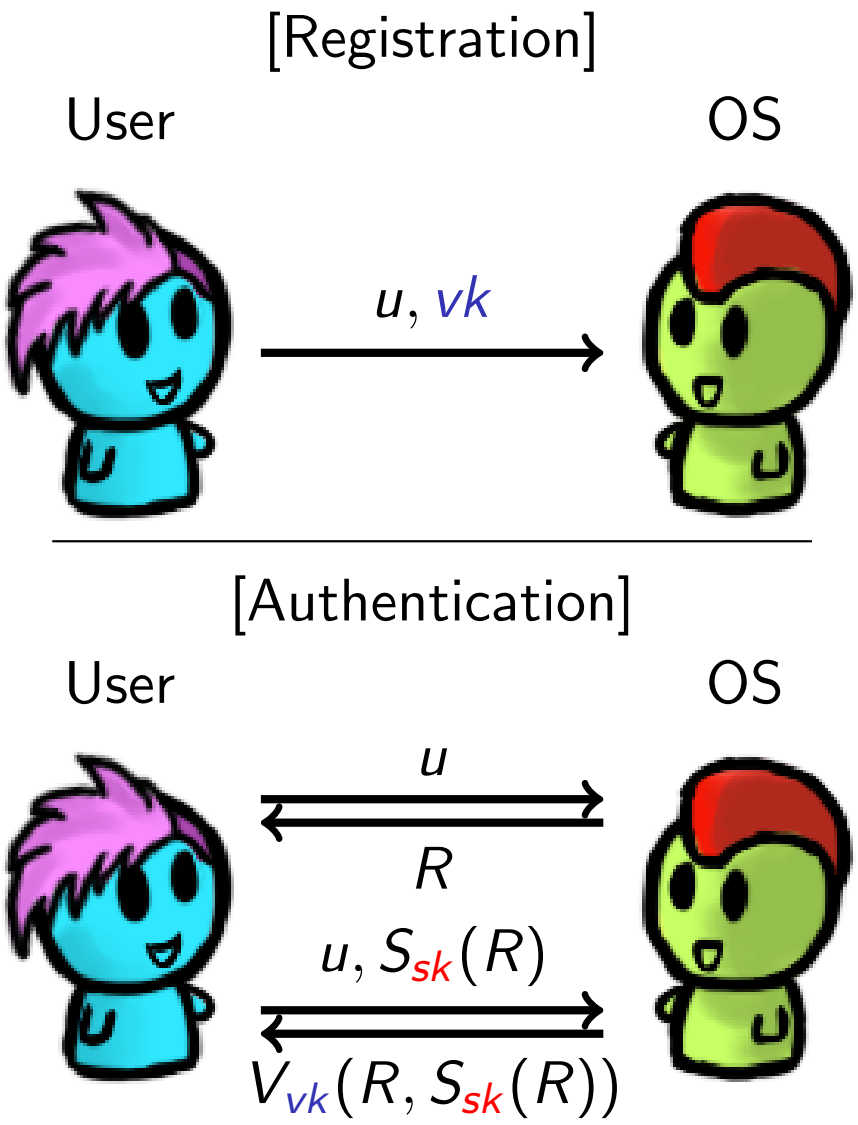


[Authentication]

User OS

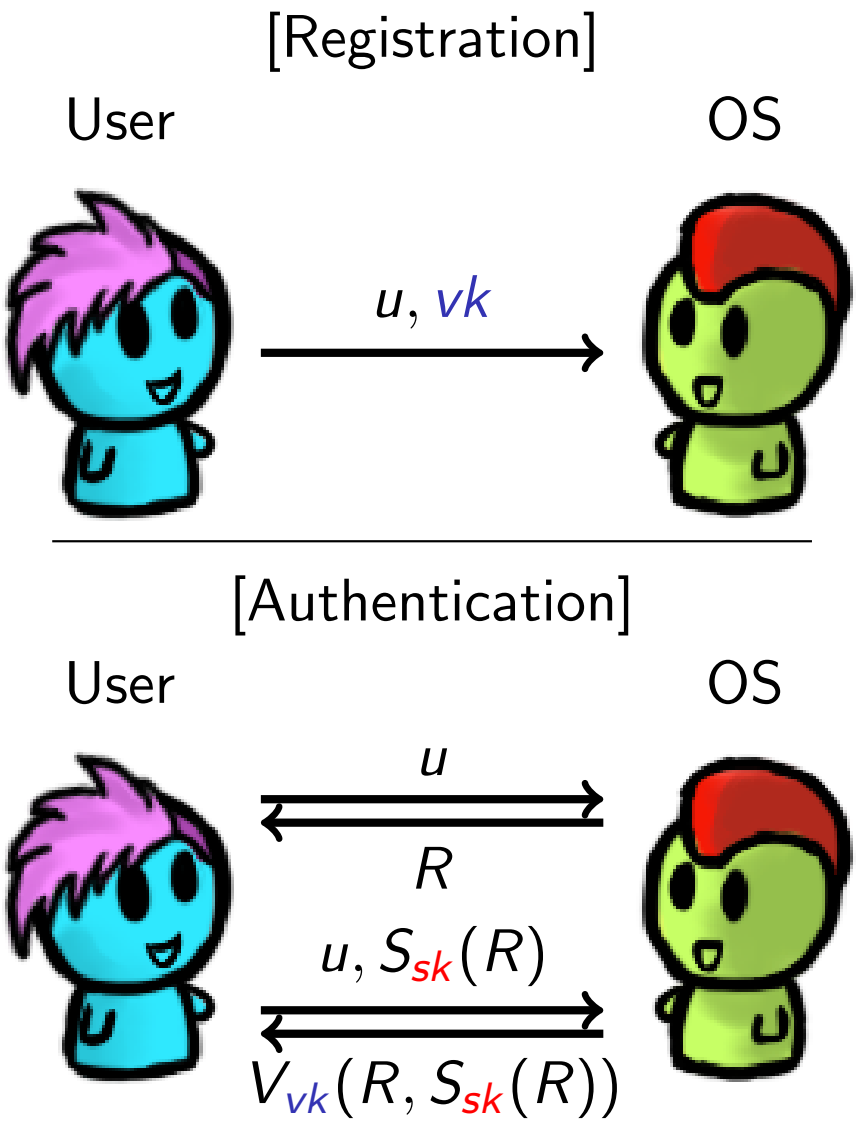


Passkey



This is essentially what you do with passwordless SSH.

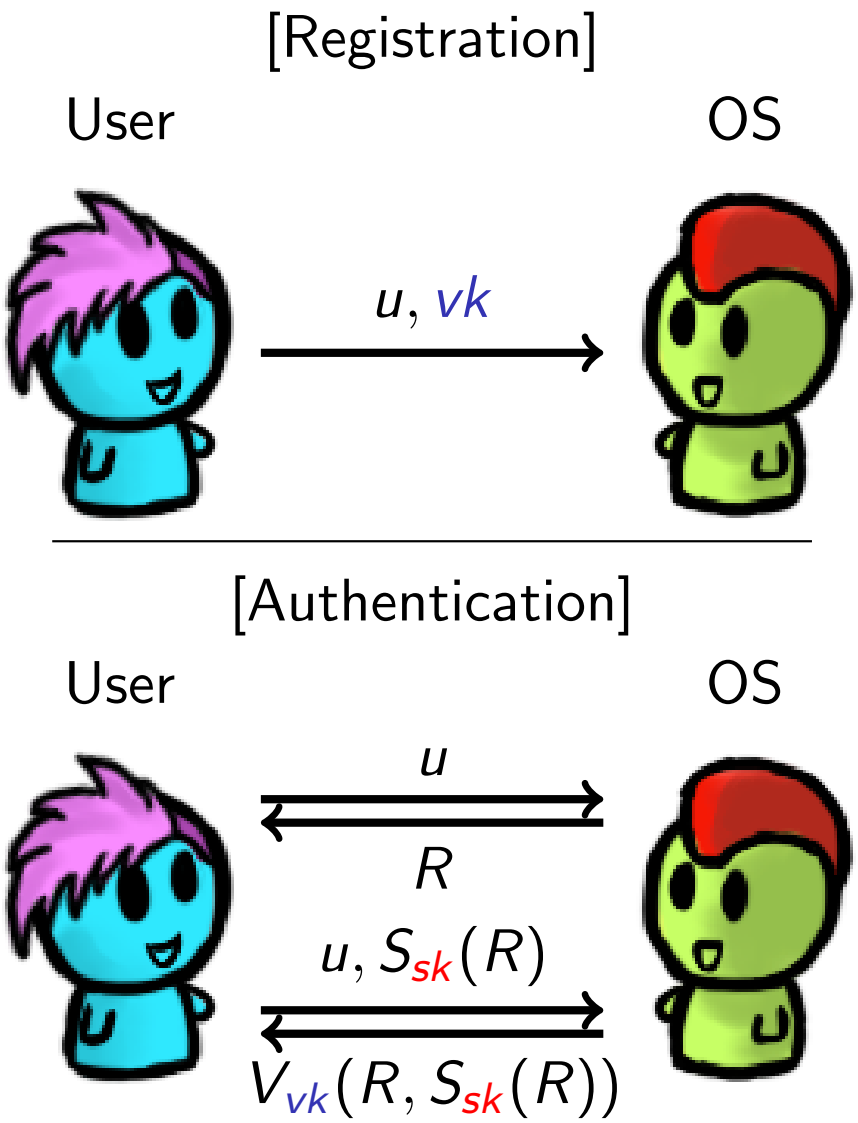
Passkey



This is essentially what you do with passwordless SSH.

Q: How do you manage the signing key (private key)?

Passkey



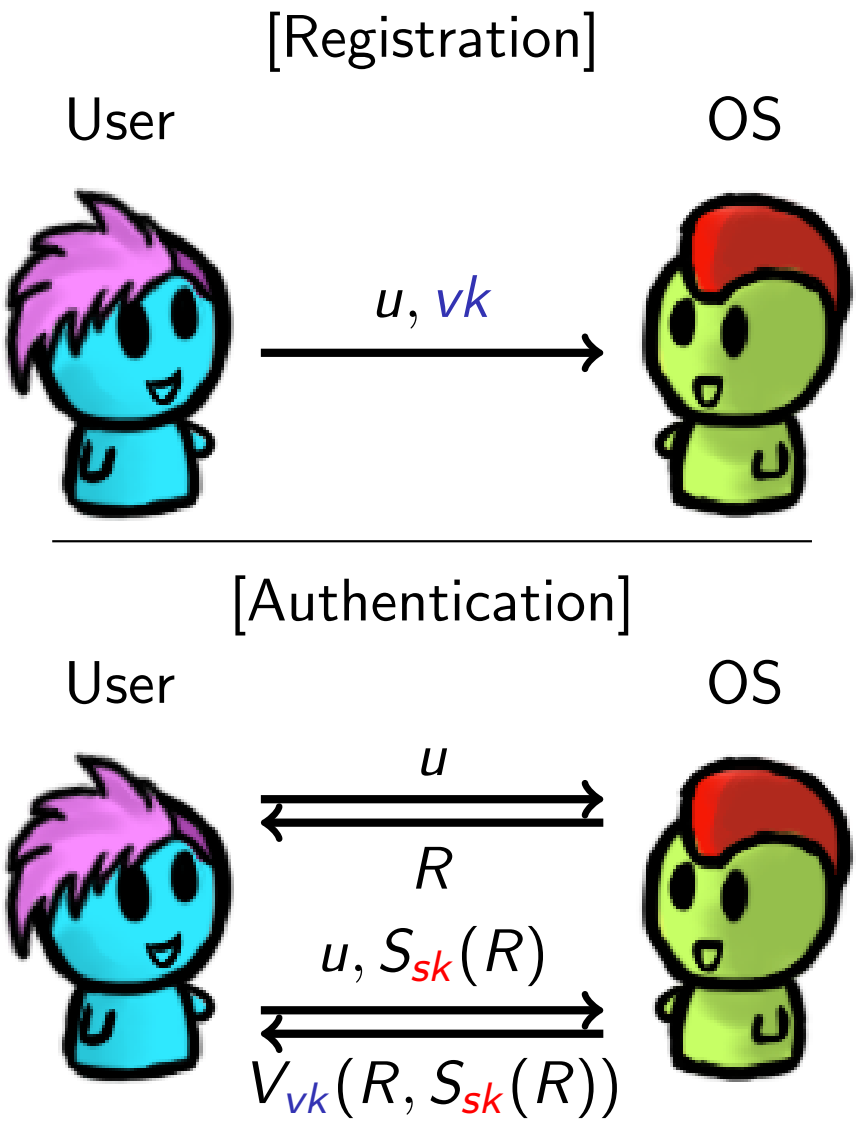
This is essentially what you do with passwordless SSH.

Q: How do you manage the signing key (private key)?

A: Hide it in some "secret vault" which can only be unlocked after local authentication, e.g.,

- password
- biometrics
- unlock patterns
- hardware tokens

Passkey



This is essentially what you do with passwordless SSH.

Q: How do you manage the signing key (private key)?

A: Hide it in some "secret vault" which can only be unlocked after local authentication, e.g.,

- password
- biometrics
- unlock patterns
- hardware tokens

See the announcement and blog post from Google on May 3rd, 2023.

$\langle$ **End** $\rangle$