# CS 489 / 698: Software and Systems Security

**Module 8: Defenses against Common Vulnerabilities**
authentication and capabilities

Meng Xu *(University of Waterloo)*

Winter 2024

# Outline

1. Introduction to authentication
2. Password — the protocol-design perspective
3. Capabilities and a case study on seL4

# 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 get 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 get 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

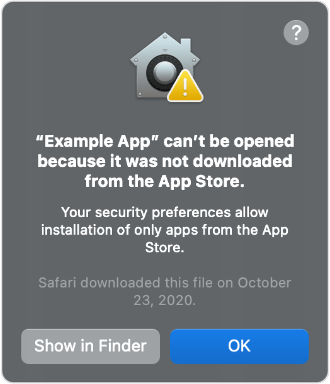
- User authentication
  - Something we all know
- Program authentication
  - Something you might have seen
- Process authentication
  - *What does this even mean?*

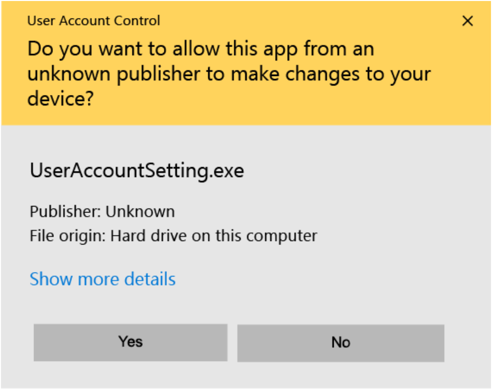
# Program authentication

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

# Program authentication

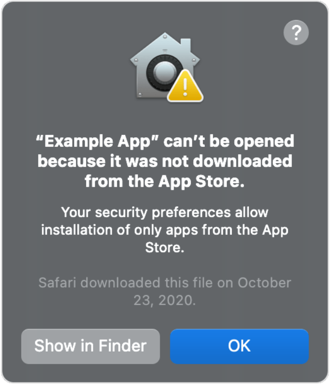
**Goal**: prove to the operating system (or to the end user) that the program originates from a trusted source and is unmodified.

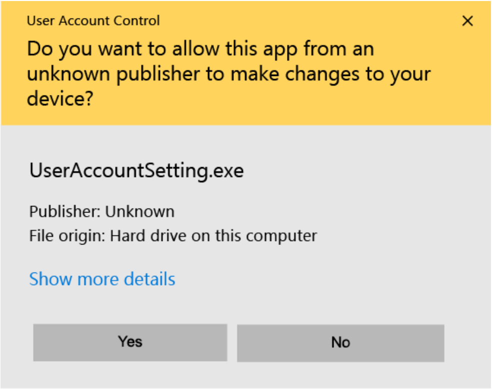


# Program authentication

**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

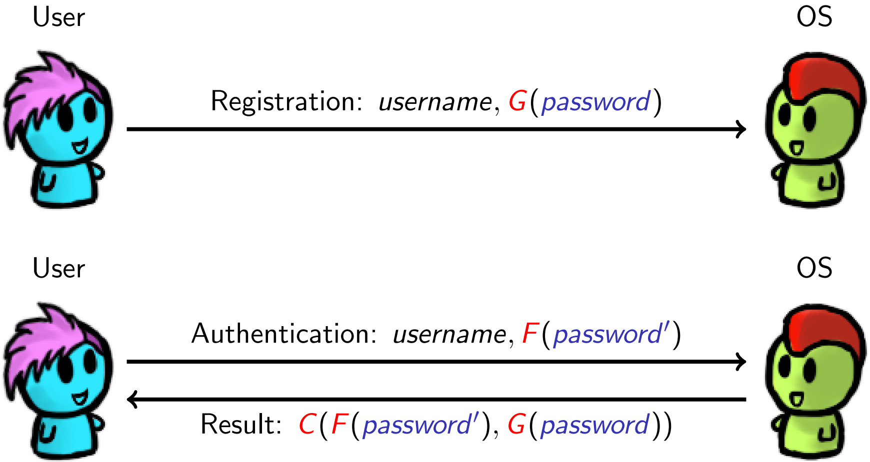
# Outline

# A formal modeling of password

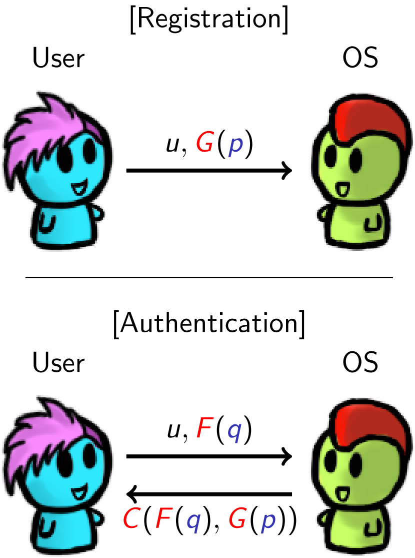
A formal model is useful for examining the pros and cons of several password-based authentication protocols.

# A formal modeling of password

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

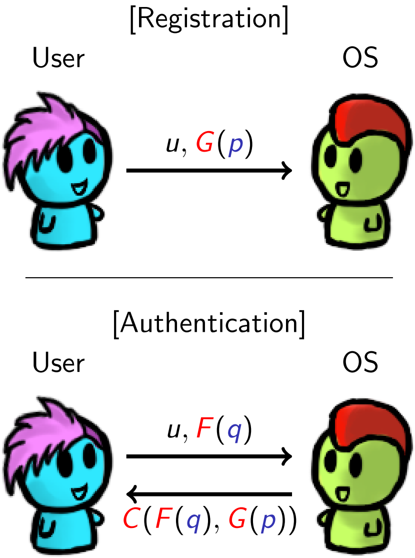


# Design space




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

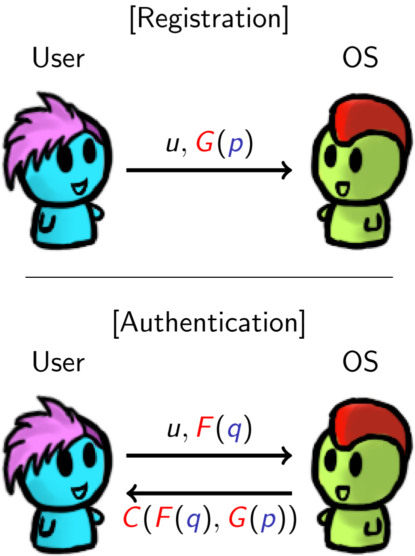
# Design space




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



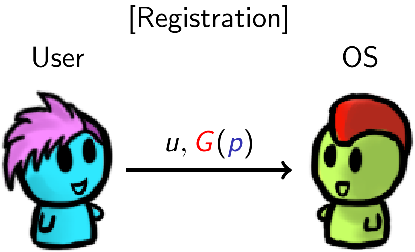

The design space of a password-based authentication protocol is around functions $G(p)$, $F(q)$, and $C(F(q), G(p))$

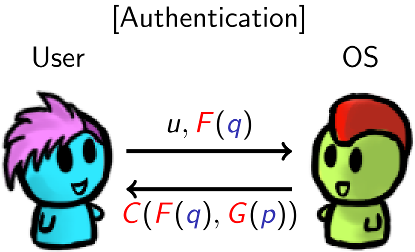
**Q**: What is the correctness requirement of the protocol?

**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




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:

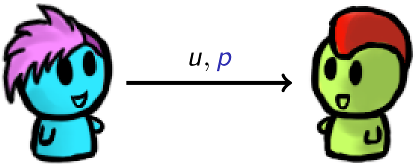
- $p = q \implies C(F(q), G(p)) = \mathrm{T}$
- $p \neq q \implies C(F(q), G(p)) = \mathrm{F}$

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

# Option 1: plaintext password

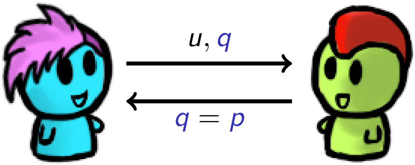
[Registration]

User OS



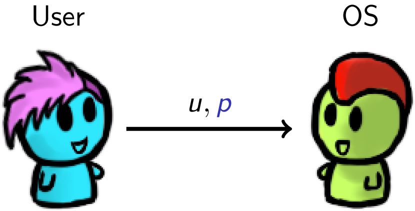

[Authentication]

User OS

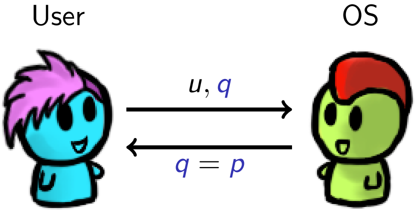


# Option 1: plaintext password

[Registration]

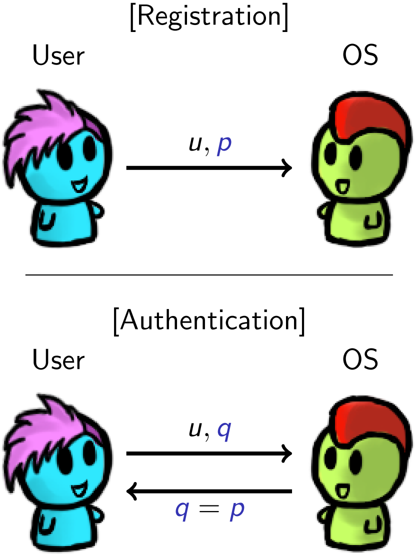



[Authentication]




**Q**: What is wrong with this scheme?

# Option 1: plaintext password



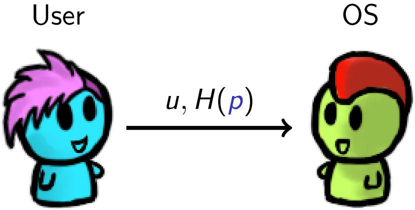

**Q**: What is wrong with this scheme?

**A**: Storing passwords in plaintext is extremely dangerous

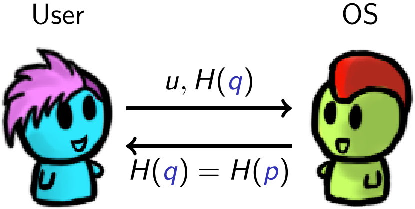
- Password file might end up on backup tapes
- Intruder into OS might get access to password file
- System administrators have access to the file and might use passwords to impersonate users at other systems
  - Many people re-use passwords across multiple systems

# Option 2: password fingerprint

[Registration]




[Authentication]



# 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

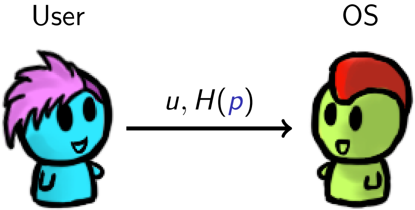
- Common examples: MD5, SHA-1, SHA-2, SHA-3 (a.k.a., Keccak, from 2012 on), where MD5 and SHA-1 are not considered safe now.

A hash function is cryptographically secure if it has three properties

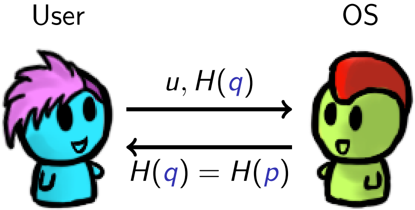
1. Preimage-resistance:
   - Given $y$, it's hard to find $x$ such that $h(x) = y$
     i.e., a "preimage" of $y$
2. Second preimage-resistance:
   - Given $x$, it's hard to find $x' \neq x$ such that $h(x) = h(x')$
     i.e., a "second preimage" of $h(x)$
3. Collision-resistance:
   - It's hard to find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

# Option 2: password fingerprint

[Registration]



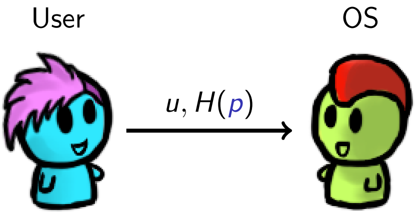

[Authentication]



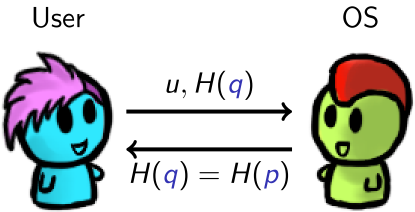

$H$ is a cryptographic hash function (e.g., SHA-2, SHA-3)

# Option 2: password fingerprint

[Registration]



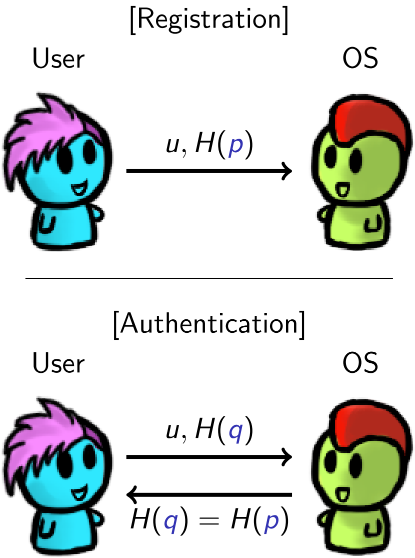

[Authentication]




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

**Q**: Does this protocol satisfy the correctness requirement?

# Option 2: password fingerprint



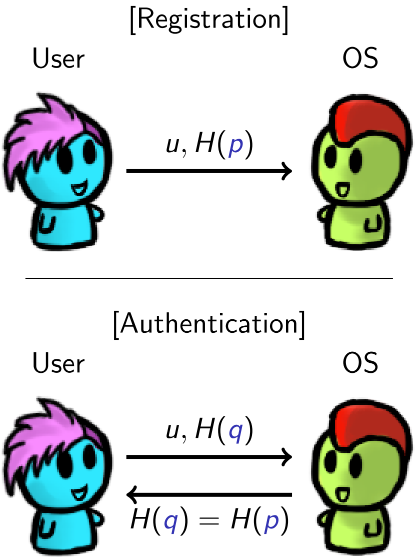

$H$ is a cryptographic hash function (e.g., SHA-2, SHA-3)

**Q**: Does this protocol satisfy the correctness requirement?

**A**: Two properties:

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

# Option 2: password fingerprint




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

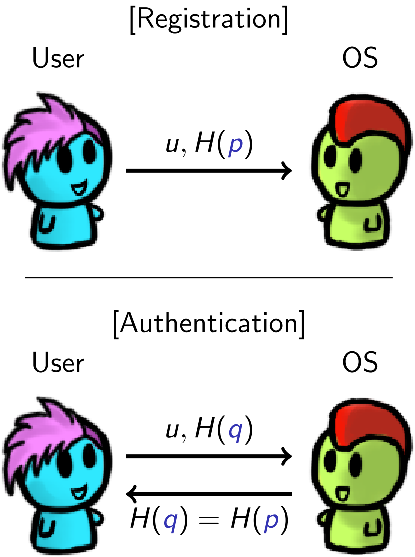
**Q**: Does this protocol satisfy the correctness requirement?

**A**: Two properties:

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

**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:

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

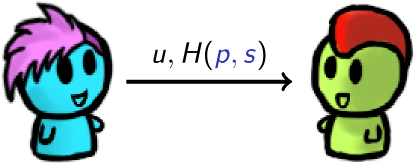
**Q**: What other weaknesses this protocol may have?

**A**: Same password, same fingerprint

# Option 3a: salted password fingerprint

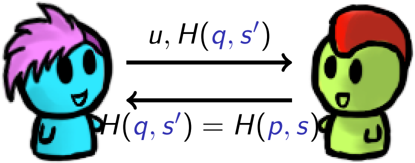
[Registration]

User OS

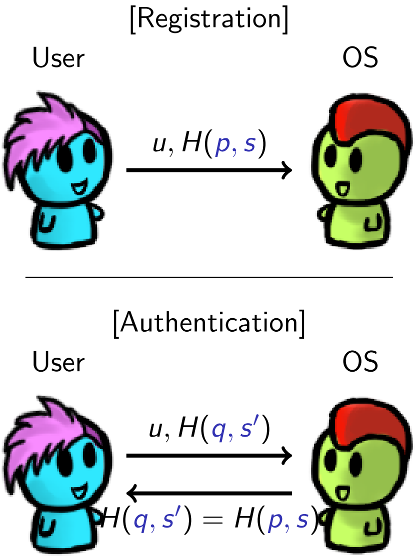



[Authentication]

User OS



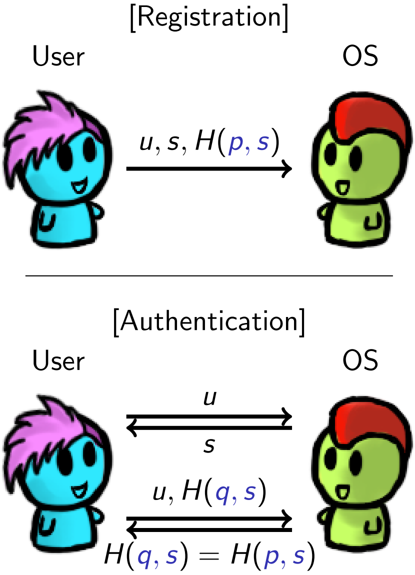
# Option 3a: salted password fingerprint



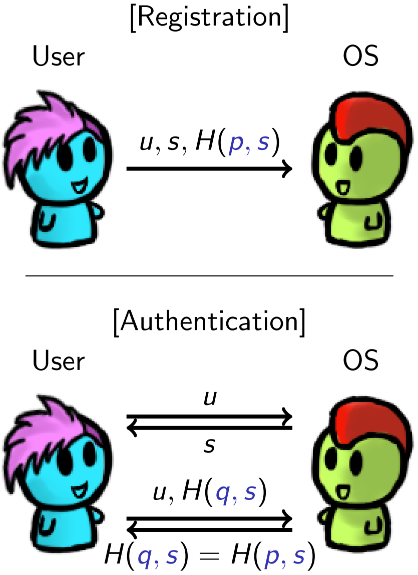

In this scheme, the user (or the client program) is responsible for remembering and managing the salt.

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

# Option 3b: salted password fingerprint



# 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

$u, H(p)$ →

---

[Authentication]

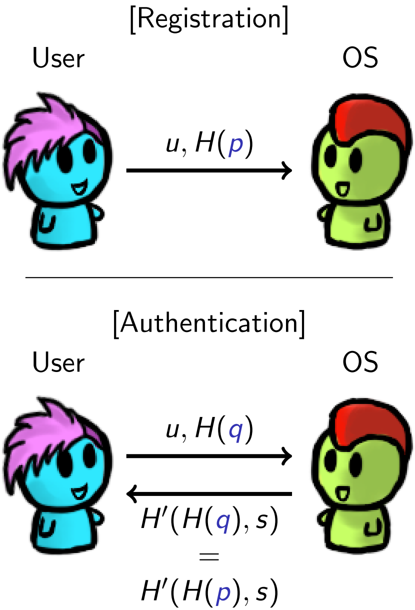
User — OS

$u, H(q)$ →

← 

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

# Option 3c: salted password fingerprint




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

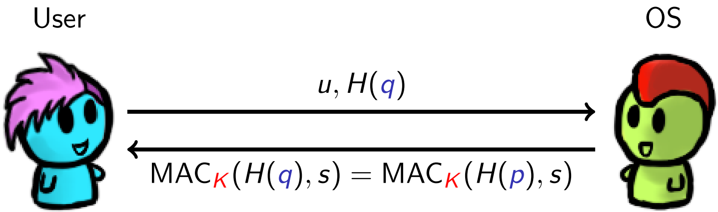
It prevents offline dictionary attacks when the password file is leaked from the OS (e.g., via breach), but has little protection over eavesdropping attacks over the network.

# 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.

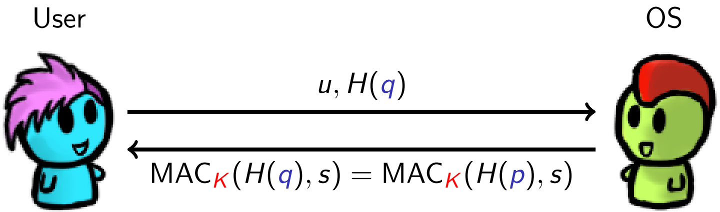
# Further protections against offline guessing attacks

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



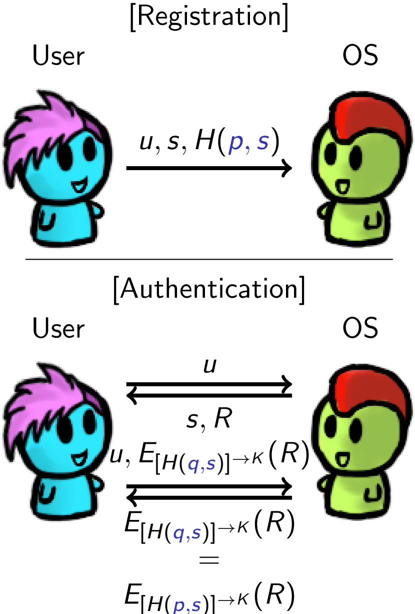
# Further protections against offline guessing attacks

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



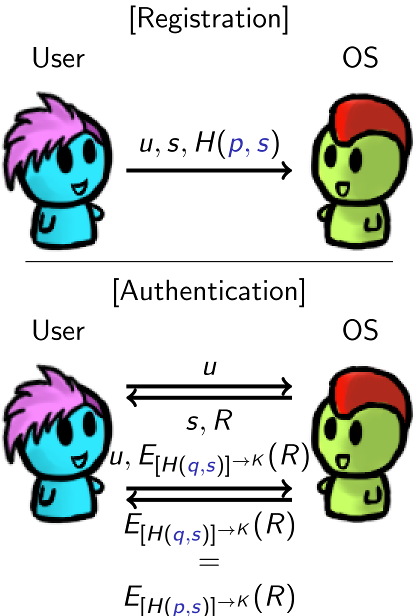

  - Protect the secret key by embedding it in tamper-resistant hardware.
  - If the key does leak, the scheme remains as secure as a scheme based on a cryptographic hash.

# Option 4: challenge-response protocol




**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




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

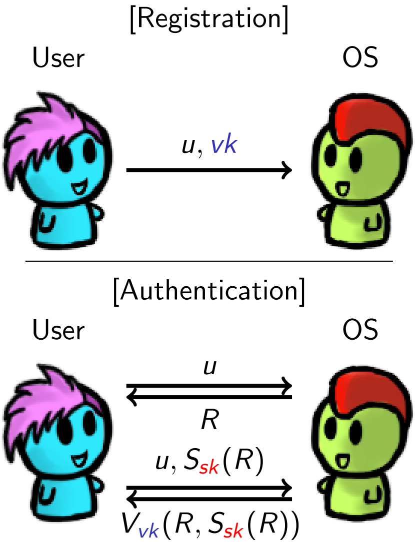
**Q**: What are the potential problems with this protocol?

# Option 4: challenge-response protocol

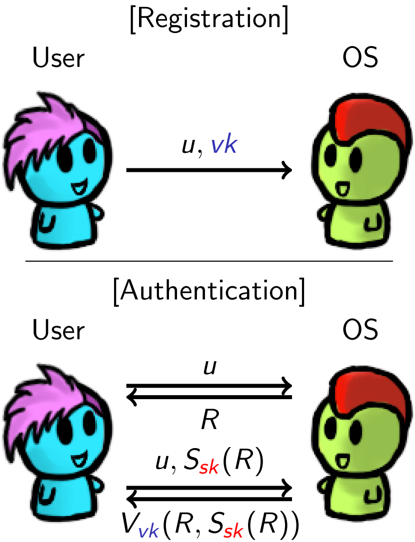
For serious designs of challenge-response protocol, please refer to:

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

# Passkey

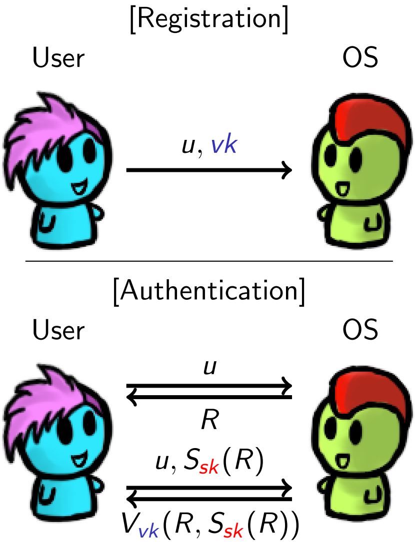

[Registration]
User
OS
$u$, $vk$
[Authentication]
User
OS
$u$
$R$
$u, S_{sk}(R)$
$V_{vk}(R, S_{sk}(R))$

# Passkey




This is essentially what you do with passwordless SSH.

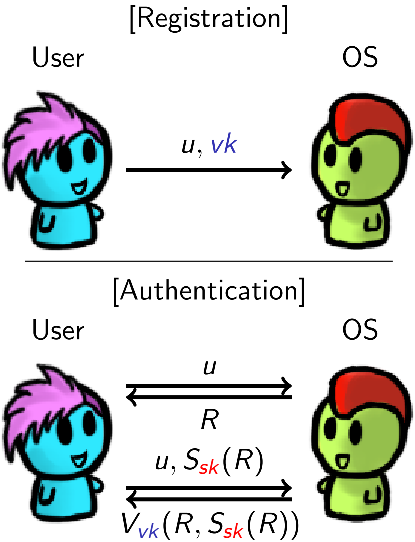
# Passkey




This is essentially what you do with passwordless SSH.

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

# Passkey



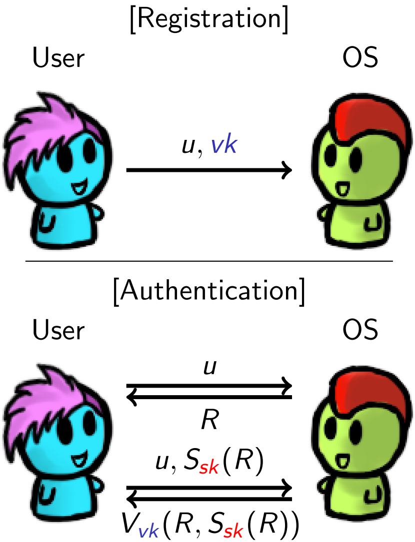

This is essentially what you do with passwordless SSH.

**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.

# Outline

# Capabilities

A capability is an unforgeable token that gives its owner some access rights to an object.

Example:

- C1: {File 1:w}, C2: {File 2:r}, C3: {File 3: o}, C4: {File 2: x}
- Alice: {C1, C2, C3, C4}, Bob: {C2, C4}, Carol: {C4}

# Capabilities

A capability is an unforgeable token that gives its owner some access rights to an object.

Example:

- C1: {File 1:w}, C2: {File 2:r}, C3: {File 3: o}, C4: {File 2: x}
- Alice: {C1, C2, C3, C4}, Bob: {C2, C4}, Carol: {C4}

Some properties about capabilities-based system:

- Unforgeability enforced by either
  - a component running at a higher privilege level (e.g., kernel)
  - cryptographic mechanisms (e.g., digital signatures)
- Tokens might be transferable (or non-transferable)
- Tokens might be copyable (or non-copyable)
- Tokens serve both authentication and access control

# Capabilities

A capability is an unforgeable token that gives its owner some access rights to an object.

Example:

- C1: {File 1:w}, C2: {File 2:r}, C3: {File 3: o}, C4: {File 2: x}
- Alice: {C1, C2, C3, C4}, Bob: {C2, C4}, Carol: {C4}

Some properties about capabilities-based system:

- Unforgeability enforced by either
  - a component running at a higher privilege level (e.g., kernel)
  - cryptographic mechanisms (e.g., digital signatures)
- Tokens might be transferable (or non-transferable)
- Tokens might be copyable (or non-copyable)
- Tokens serve both authentication and access control

Some research/experimental OSs (e.g., Fuchsia, seL4) have fine-grained support for tokens.

# Capabilities

**Q**: Which of the following can we do quickly for capabilities?

- Determine set of allowed users per object
- Determine set of objects that a user can access
- Revoke a user's access right to an object
- Revoke a user's access right to all objects
- Revoke all users' access rights to an object

# Capabilities

**Q**: Which of the following can we do quickly for capabilities?

- Determine set of allowed users per object
- Determine set of objects that a user can access
- Revoke a user's access right to an object
- Revoke a user's access right to all objects
- Revoke all users' access rights to an object

**A**: Hard, Easy, Easy, Easy, Easy

# What is seL4?

**Overview**: seL4 is an open source, high-assurance, high-performance operating system microkernel.

# What is seL4?

**Overview**: seL4 is an open source, high-assurance, high-performance operating system microkernel.

- Available on GitHub under GPLv2 license
- Contains a comprehensive set of mathematical proofs for correctness and security
- Arguably the fastest microkernel in the world
- Aims to be a piece of software that runs at the heart of any system and controls all accesses to resources

# Monolithic kernel vs microkernel

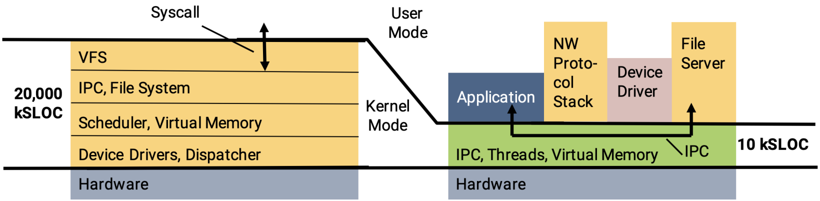
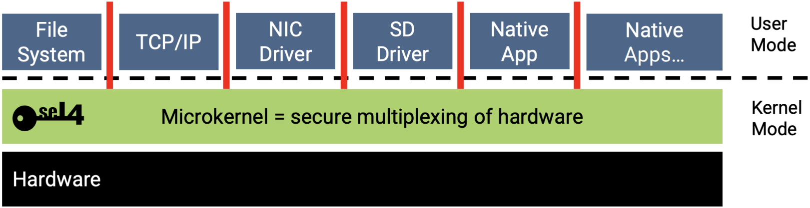



Figure illustrating the difference between

- monolithic kernel (e.g., the Linux kernel) on the left and
- microkernel (e.g., seL4) (on the right)

Adapted from seL4 Whitepaper.

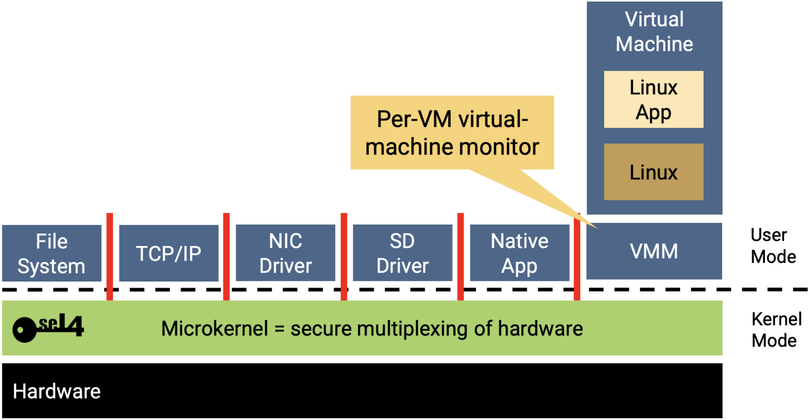
# Microkernel




All operating-system services are user-level processes:

- file systems
- device drivers
- network stack
- power management
- . . .

# Microkernel as hypervisor




Adapted from seL4 Overview Slides on seL4 Summit 2022

# seL4 capability system

**General principle**: anything goes through seL4 needs a capability!

# seL4 capability system

**General principle**: anything goes through seL4 needs a capability!

# seL4 capability system

**General principle**: anything goes through seL4 needs a capability!


A capability is an object reference that conveys specific rights to a particular object

- Capability = Access Token: prima-facie evidence of privilege
- Access rights include read, write, send, reply, execute, . . .
- Kernel object is one of ten object types

# seL4 capability system

**General principle**: anything goes through seL4 needs a capability!

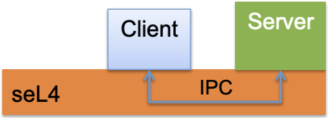

A capability is an object reference that conveys specific rights to a particular object

- Capability = Access Token: prima-facie evidence of privilege
- Access rights include read, write, send, reply, execute, . . .
- Kernel object is one of ten object types

**Any system call is invoking a capability**: `r = cap.method(args);`

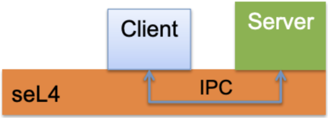
# seL4 protected procedure calls (IPC)

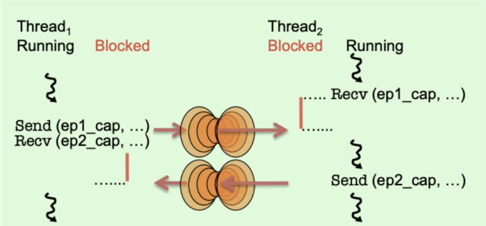
Protected procedure call (IPC for historical reasons) is a fundamental operation in seL4.



# seL4 protected procedure calls (IPC)

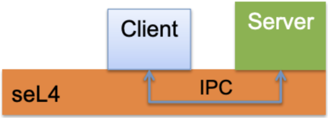
Protected procedure call (IPC for historical reasons) is a fundamental operation in seL4.

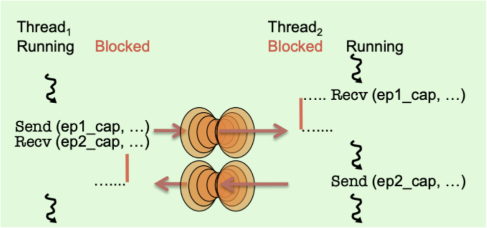


# seL4 protected procedure calls (IPC)

Protected procedure call (IPC for historical reasons) is a fundamental operation in seL4.

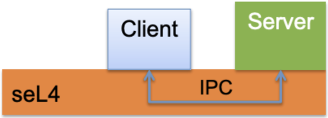


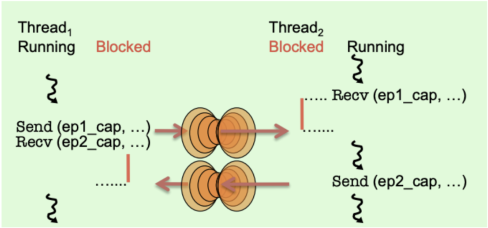




**Q**: How would a normal open syscall be like in seL4?

# seL4 protected procedure calls (IPC)

Protected procedure call (IPC for historical reasons) is a fundamental operation in seL4.




**Q**: How would a normal open syscall be like in seL4?

**A**: `Call(ext4fs_endpoint_cap, OPEN_FILE, <extra-args>)`

- Mint `reply_cap`
- `Send(ext4fs_endpoint_cap, reply_cap, ...)`
- `Recv(reply_cap, ...)`

# seL4 kernel objects

- **Endpoints** are used to perform protected function calls
- **Reply Objects** represent a return path from a protected procedure call
- **Address Spaces** provide the sandboxes around components (thin wrappers abstracting hardware page tables)
- **Cnodes** store capabilities representing a component's access rights
- **Thread Control Blocks** represent threads of execution
- **Scheduling Contexts** represent the right to access a certain fraction of execution time on a core
- **Notifications** are synchronisation objects (similar to semaphores)
- **Frames** represent physical memory that can be mapped into address spaces
- **Interrupt Objects** provide access to interrupt handling
- **Untypeds** unused (free) physical memory that can be converted ("retyped") into any of the other types.

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