ICA0002: IT Infrastructure Services

SSH Basics

Roman Kuchin
Juri Hudolejev
2025

SSH: Secure Shell

Remote shell operated securely over insecure network

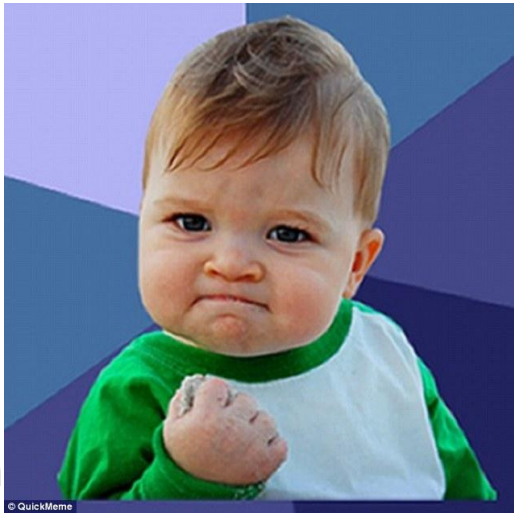
Replaced **`telnet`**, **`rsh`**, **`rlogin`** and **`rexec`**

De-facto standard for remote administration and automation

Default transport protocol in Ansible for connecting to managed hosts

More info: https://www.ssh.com/academy/ssh

SSH: Secure Shell



Remote shell operated securely over insecure network

Replaced `telnet`, `rsh`, `rlogin` and `rexec`

De-facto standard for remote administration and automation

Default transport protocol in Ansible for connecting to managed hosts

More info: https://www.ssh.com/academy/ssh

CIA -- but the other one…

Confidentiality

- Data is readable only by intended parties

Integrity

- Data is protected from tampering in transit

Authentication

- Identity of both endpoints is verified

Encryption

Symmetric encryption: DES (obsolete), AES, ChaCha20 etc.

- Same key for encryption and decryption -- shared secret

Asymmetric (public key) encryption: DSA (obsolete), RSA, ECDSA etc.

- Public key (shared) + private key (kept secret)
- Message is encrypted with one key from the pair
  can only be decrypted with the other key from the same pair

Encryption

Symmetric encryption:

Needs a secure channel to distribute the shared secret :(

Asymmetric (public key) encryption:

Computationally expensive for large data :(

Let's use a **public key** to derive the **shared key**, then encrypt with it. Call it key exchange?

That would work!

SSH session initialization

Client opens TCP connection to the server and starts the SSH protocol

Client and server both:

- agree algorithms for key exchange, symmetric and public key encryption
- generate shared session key using Diffie-Hellman method (SSH v2)

Client authenticates to the server

If all good, encrypted session is established

SSH client authentication

Common methods:

- User password
- User public key: RSA, Ed25519 etc.
- Host public key
- Interactive -- for one time passwords
- GSSAPI -- for external authentication services (example: Kerberos)

SSH client authentication

Common methods:

- User password ← avoid at all costs!
- User public key: RSA, Ed25519 etc. ← we only use this on this course
- Host public key
- Interactive -- for one time passwords
- GSSAPI -- for external authentication services (example: Kerberos)

Details: https://datatracker.ietf.org/doc/html/rfc4252#section-8

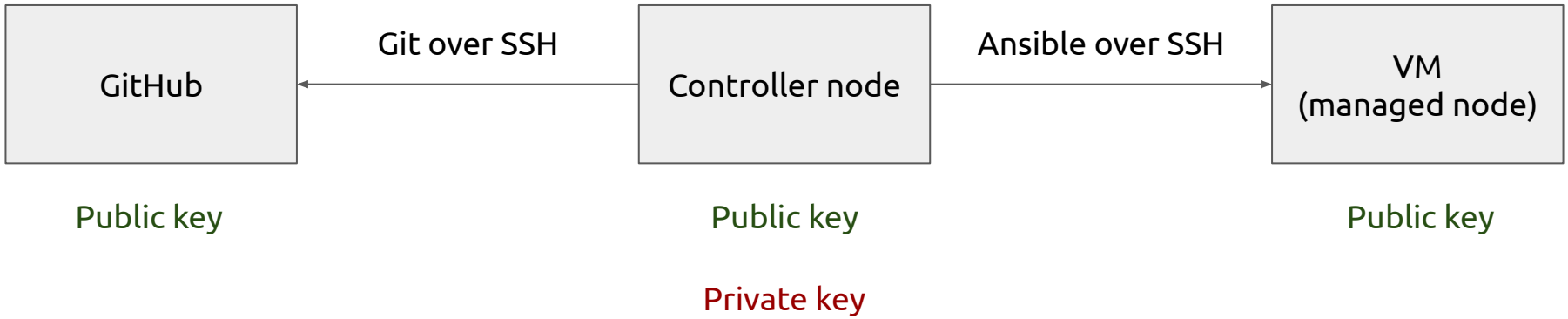
SSH public key authentication

In very simple terms:

- Client signs specific data with its **private** key
- Client sends its **public** key and the signature to the server
- Server checks whether the public key is authorized for the user account
- Server verifies the signature using the public key
- If all the checks pass, the client is authenticated

Details: https://datatracker.ietf.org/doc/html/rfc4252#section-7

SSH in this course



Your SSH keys in this course

Your public key:

`~/.ssh/id_ed25519.pub` file on Controller node (connecting from)

`~/.ssh/authorized_keys` file on your VMs (connecting to)

In your GitHub account: https://github.com/<username>.keys (lab 1)

Your private key:

`~/.ssh/id_ed25519` file on Controller node: must not leave your machine!

Public key may also be computed from the private key file, but not vice versa!

Your SSH keys in this course (if using RSA keys)

Your public key:

- **`~/.ssh/id_rsa.pub`** file on Controller node (connecting from)
- **`~/.ssh/authorized_keys`** file on your VMs (connecting to)
- In your GitHub account: https://github.com/<username>.keys (lab 1)

Your private key:

- **`~/.ssh/id_rsa`** file on Controller node: must not leave your machine!

Public key may also be computed from the private key file, but not vice versa!

Important!

If your private key is lost or compromised,

1. Delete the corresponding public key from your GitHub account immediately!
2. Generate a new key pair (see lab 1)
3. Contact the teachers so they can remove the old key from your VMs
4. Rotate the keys on any other systems that use the same key pair

Questions?

SSH session initialization

Client opens TCP connection to the server and starts the SSH protocol

Client and server both:

- agree algorithms for key exchange, symmetric and public key encryption
- generate shared session key using Diffie-Hellman method (SSH v2)

~~~ What step is missing here? ~~~

Client authenticates to the server

If all good, encrypted session is established
~~~

SSH session initialization

Client opens TCP connection to the server and starts the SSH protocol

Client and server both:

- agree algorithms for key exchange, symmetric and public key encryption
- generate shared session key using Diffie-Hellman method (SSH v2)

Client verifies the server identity

Client authenticates to the server

If all good, encrypted session is established

Host SSH keys

Host key:

- Identifies the host (server) the client is connecting to
- Verified by the client during session initialization

Host SSH keys

Host key:

- Identifies the host (server) the client is connecting to
- Verified by the client during session initialization

Client key (discussed earlier):

- Identifies the client connecting to the server
- Verified by the server during client authentication

Host SSH keys

Host key:

- Identifies the host (server) the client is connecting to
- Verified by the client during session initialization

Client key (discussed earlier):

- Identifies the client connecting to the server
- Verified by the server during client authentication

"Host key verification" ≠ "Host key based client authentication"

^-- The server checks the client's host key

^-- The client checks the server's host key

Host SSH keys: first connection

Host key is approved manually on the client's first connection to this host:

```
$ ssh -p9022 ubuntu@193.40.157.25
The authenticity of host '193.40.157.25 (193.40.157.25)' can't be established.
ECDSA key fingerprint is SHA256:{...key fingerprint...}.
Are you sure you want to continue connecting (yes/no)?
```

Host SSH key is changed

Client will discard SSH connection if the host key is different from previously approved value:

```
$ ssh -p9022 ubuntu@193.40.157.25
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
```

Host SSH key files

Host public key:

`/etc/ssh/ssh_host_ecdsa_key.pub` file on your VMs (connecting to)

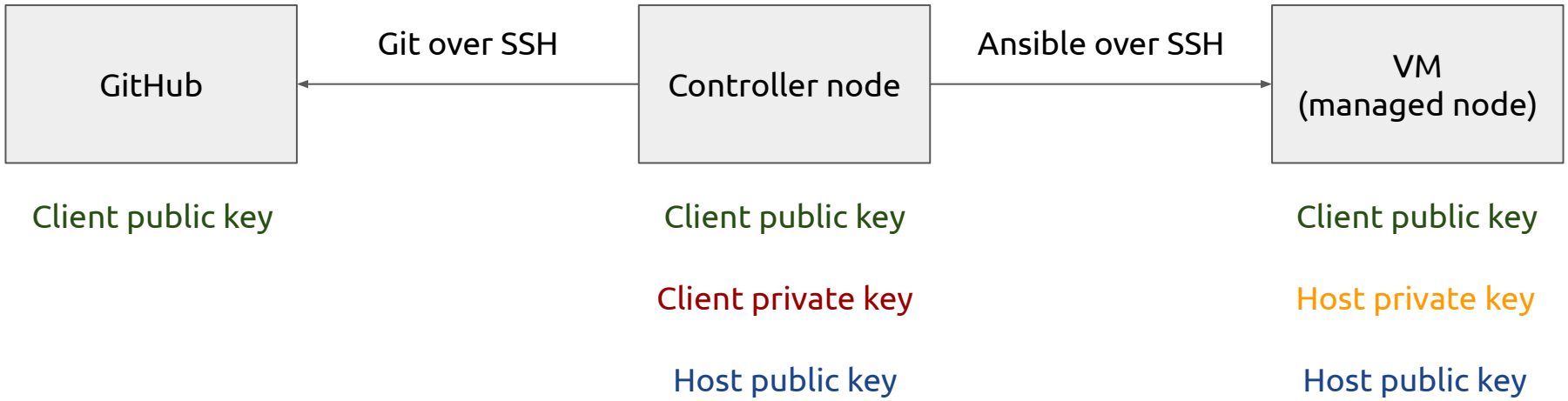
`~/.ssh/known_hosts` file on Controller node (connecting from)

Host private key:

`/etc/ssh/ssh_host_ecdsa_key` file on your VMs

On this course: do NOT touch host private keys!

SSH in this course



Questions?

What key is stored in this file?

Also where is the file located (client machine, server etc.)?

1. `~/.ssh/authorized_keys`
2. `~/.ssh/id_ed25519`
3. `~/.ssh/id_ed25519.pub`
4. `~/.ssh/known_hosts`
5. `/etc/ssh/ssh_host_ecdsa_key`
6. `/etc/ssh/ssh_host_ecdsa_key.pub`
7. `https://github.com/elvis.keys`

What key is stored in this file?

1. `~/.ssh/authorized_keys` ← client public key on the server
2. `~/.ssh/id_ed25519` ← client private key
3. `~/.ssh/id_ed25519.pub` ← client public key on client machine
4. `~/.ssh/known_hosts` ← server public key on client machine
5. `/etc/ssh/ssh_host_ecdsa_key` ← server private key
6. `/etc/ssh/ssh_host_ecdsa_key.pub` ← server public key on the server
7. `https://github.com/elvis.keys` ← client public key in GitHub