SSH authentication using user and machine identities

Morten Linderud

$ whoami

- Morten Linderud
  - Foxboron
- F/OSS developer since ~2013
- Arch Linux Developer since ~2016
- Usable security tools
- Hackeriet
- Devops at NRK

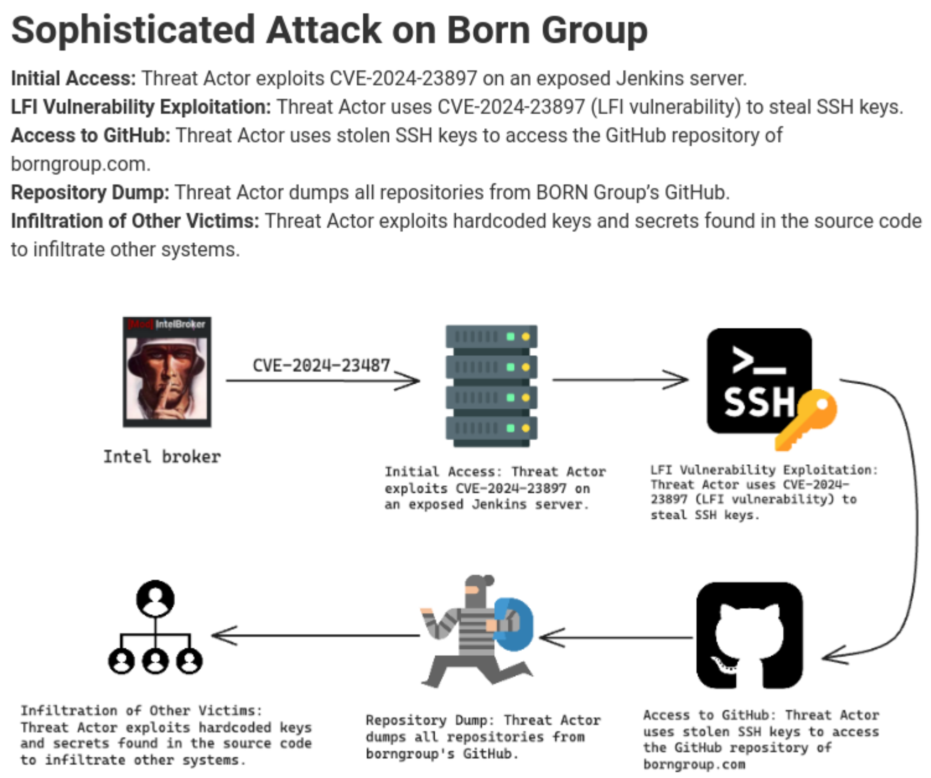
Lets make an ssh key!

Key Compromises

Key Compromises

- Not limited to SSH keys
- Impersonation
- Priviledge escalation

Born Group breach (2024)



Sophisticated Attack on Born Group

Initial Access: Threat Actor exploits CVE-2024-23897 on an exposed Jenkins server.
LFI Vulnerability Exploitation: Threat Actor uses CVE-2024-23897 (LFI vulnerability) to steal SSH keys.
Access to GitHub: Threat Actor uses stolen SSH keys to access the GitHub repository of borngroup.com.
Repository Dump: Threat Actor dumps all repositories from BORN Group's GitHub.
Infiltration of Other Victims: Threat Actor exploits hardcoded keys and secrets found in the source code to infiltrate other systems.

Initial Access: Threat Actor exploits CVE-2024-23897 on an exposed Jenkins server.

LFI Vulnerability Exploitation: Threat Actor uses CVE-2024-23897 (LFI vulnerability) to steal SSH keys.

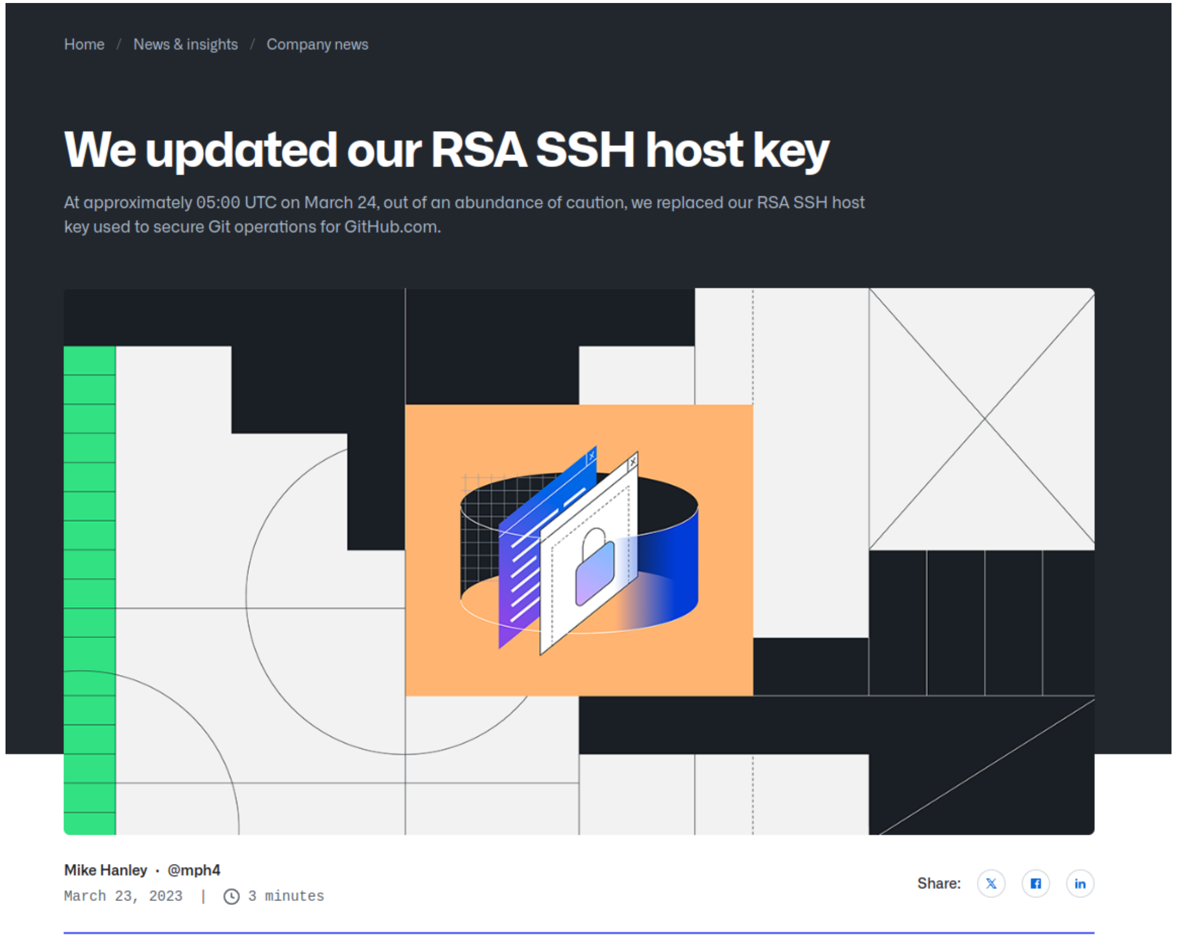
Access to GitHub: Threat Actor uses stolen SSH keys to access the GitHub repository of borngroup.com

Repository Dump: Threat Actor dumps all repositories from borngroup's GitHub.

Infiltration of Other Victims: Threat Actor exploits hardcoded keys and secrets found in the source code to infiltrate other systems.

BORN Group Supply Chain Breach: In-Depth Analysis of Intelbroker's Jenkins Exploitation

Github RSA ssh host key leak (2023)



Home / News & insights / Company news

We updated our RSA SSH host key

At approximately 05:00 UTC on March 24, out of an abundance of caution, we replaced our RSA SSH host key used to secure Git operations for GitHub.com.

Mike Hanley · @mph4

March 23, 2023 | 3 minutes

Github Blog - We updated our RSA SSH host key

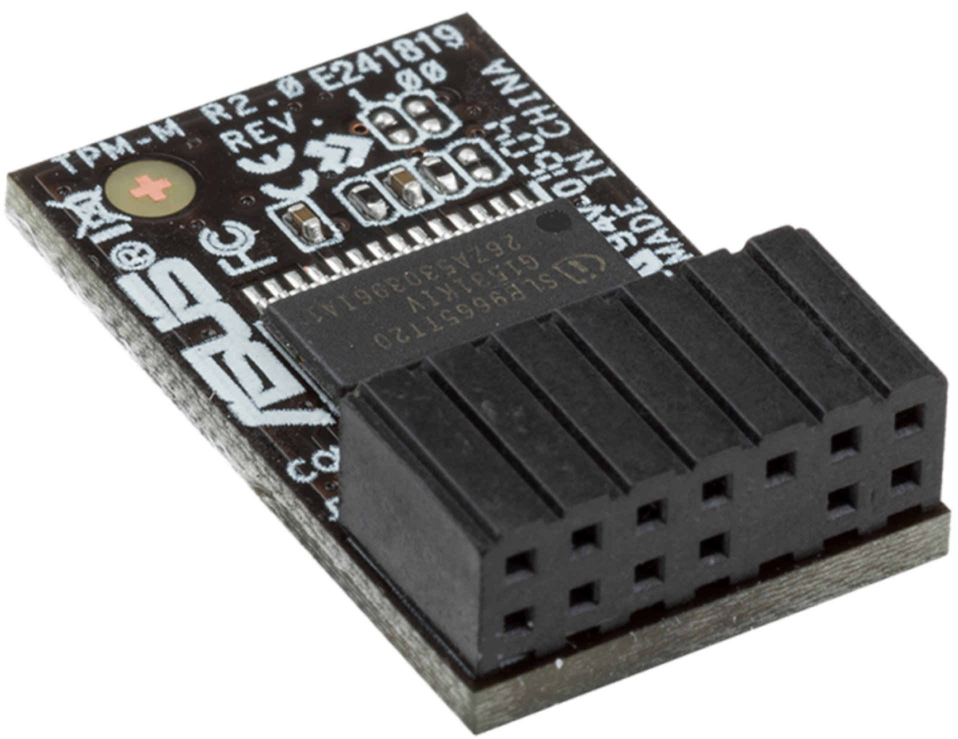
Goals

- Device bound keys
- Identity claims
- Machine identity

Goals

- Device bound keys
  - Prevents key extraction
- Identity claims
  - Ensures the user has access to the correct identity.
- Machine identity
  - Ensures the user has access to the correct machine.

Trusted Platform Module (TPM)



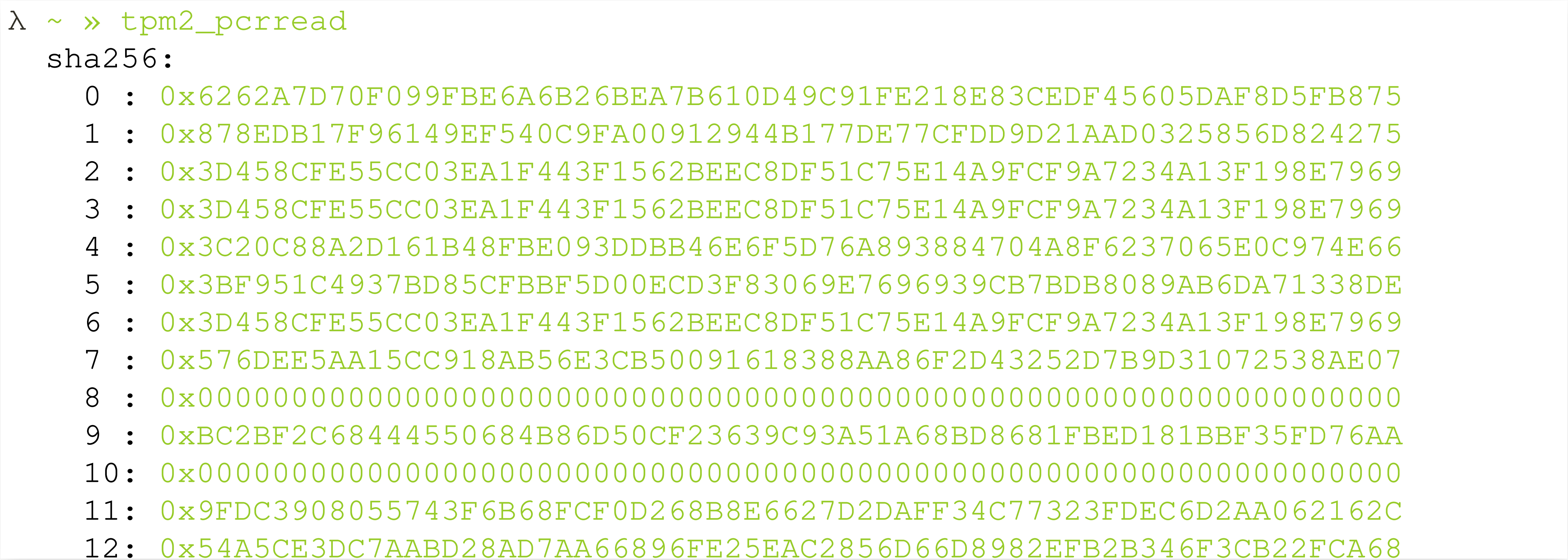
What are they?

- Secure crypto processor
- Boot integrity
- Implementation:
    - Discrete TPM (dTPM)
    - Firmware TPM (fTPM)

Features

- Platform Integrity
- Attestation (later in the talk)
- Hierarchies and keys

Platform Integrity



```
λ ~ » tpm2_pcrread
  sha256:
    0 : 0x6262A7D70F099FBE6A6B26BEA7B610D49C91FE218E83CEDF45605DAF8D5FB875
    1 : 0x878EDB17F96149EF540C9FA00912944B177DE77CFDD9D21AAD0325856D824275
    2 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
    3 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
    4 : 0x3C20C88A2D161B48FBE093DDBB46E6F5D76A893884704A8F6237065E0C974E66
    5 : 0x3BF951C4937BD85CFBBF5D00ECD3F83069E7696939CB7BDB8089AB6DA71338DE
    6 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
    7 : 0x576DEE5AA15CC918AB56E3CB5009161838 8AA86F2D43252D7B9D31072538AE07
    8 : 0x0000000000000000000000000000000000000000000000000000000000000000
    9 : 0xBC2BF2C68444550684B86D50CF23639C93A51A68BD8681FBED181BBF35FD76AA
    10: 0x0000000000000000000000000000000000000000000000000000000000000000
    11: 0x9FDC3908055743F6B68FCF0D268B8E6627D2DAFF34C77323FDEC6D2AA062162C
    12: 0x54A5CE3DC7AABD28AD7AA66896FE25EAC2856D66D8982EFB2B346F3CB22FCA68
```

UAPI Group Specification - 🔏 Linux TPM PCR Registry 🗒️

Hierarchies and keys

- Hierarchies:
    - Endorsement
    - Owner
    - Null
- Chains back to manufacturer
- Shielded keys

Key creation and signing

```
$ tpm2_createprimary -C e -c primary.ctx
$ tpm2_create -G rsa -u rsa.pub -r rsa.priv -C primary.ctx
$ tpm2_load -C primary.ctx -u rsa.pub -r rsa.priv -c rsa.ctx
$ echo "my message" > message.dat
$ tpm2_sign -c rsa.ctx -g sha256 -o sig.rssa message.dat
$ tpm2_verifysignature -c rsa.ctx -g sha256 -s sig.rssa -m message.dat
```

Key creation and signing

```
$ tpm2_createprimary -C e -c primary.ctx
$ tpm2_create -G rsa -u rsa.pub -r rsa.priv -C primary.ctx
$ tpm2_load -C primary.ctx -u rsa.pub -r rsa.priv -c rsa.ctx
$ echo "my message" > message.dat
$ tpm2_sign -c rsa.ctx -g sha256 -o sig.rssa message.dat
$ tpm2_verifysignature -c rsa.ctx -g sha256 -s sig.rssa -m message.dat
```

Key creation and signing

```
$ tpm2_createprimary -C e -c primary.ctx
$ tpm2_create -G rsa -u rsa.pub -r rsa.priv -C primary.ctx
$ tpm2_load -C primary.ctx -u rsa.pub -r rsa.priv -c rsa.ctx
$ echo "my message" > message.dat
$ tpm2_sign -c rsa.ctx -g sha256 -o sig.rssa message.dat
$ tpm2_verifysignature -c rsa.ctx -g sha256 -s sig.rssa -m message.dat
```

Key creation and signing

```
$ tpm2_createprimary -C e -c primary.ctx
$ tpm2_create -G rsa -u rsa.pub -r rsa.priv -C primary.ctx
$ tpm2_load -C primary.ctx -u rsa.pub -r rsa.priv -c rsa.ctx
$ echo "my message" > message.dat
$ tpm2_sign -c rsa.ctx -g sha256 -o sig.rssa message.dat
$ tpm2_verifysignature -c rsa.ctx -g sha256 -s sig.rssa -m message.dat
```

Key creation and signing

```
$ tpm2_createprimary -C e -c primary.ctx
$ tpm2_create -G rsa -u rsa.pub -r rsa.priv -C primary.ctx
$ tpm2_load -C primary.ctx -u rsa.pub -r rsa.priv -c rsa.ctx
$ echo "my message" > message.dat
$ tpm2_sign -c rsa.ctx -g sha256 -o sig.rssa message.dat
$ tpm2_verifysignature -c rsa.ctx -g sha256 -s sig.rssa -m message.dat
```

Key creation and signing

```
$ tpm2_createprimary -C e -c primary.ctx
$ tpm2_create -G rsa -u rsa.pub -r rsa.priv -C primary.ctx
$ tpm2_load -C primary.ctx -u rsa.pub -r rsa.priv -c rsa.ctx
$ echo "my message" > message.dat
$ tpm2_sign -c rsa.ctx -g sha256 -o sig.rssa message.dat
$ tpm2_verifysignature -c rsa.ctx -g sha256 -s sig.rssa -m message.dat
```

Key creation and signing

```
$ tpm2_createprimary -C e -c primary.ctx
$ tpm2_create -G rsa -u rsa.pub -r rsa.priv -C primary.ctx
$ tpm2_load -C primary.ctx -u rsa.pub -r rsa.priv -c rsa.ctx
$ echo "my message" > message.dat
$ tpm2_sign -c rsa.ctx -g sha256 -o sig.rssa message.dat
$ tpm2_verifysignature -c rsa.ctx -g sha256 -s sig.rssa -m message.dat
```

Key creation and signing

```
$ tpm2_createprimary -C e -c primary.ctx
$ tpm2_create -G rsa -u rsa.pub -r rsa.priv -C primary.ctx
$ tpm2_load -C primary.ctx -u rsa.pub -r rsa.priv -c rsa.ctx
$ echo "my message" > message.dat
$ tpm2_sign -c rsa.ctx -g sha256 -o sig.rssa message.dat
$ tpm2_verifysignature -c rsa.ctx -g sha256 -s sig.rssa -m message.dat
```

Policies

- Restrict usage
- Include system state
- Signed policies

Caveats

- Not an HSM(!)
  - Slow devices
- Limited cryptography
  - RSA2048
  - NIST P-256/384
  - SHA256/SHA384
- Needs user-friendly tooling
- Not supported by openssh

ssh-agent

ssh-agent

- Hold private keys for ssh
- Communicated over a UNIX socket
- Caches passwords
- Can offload key operations

ssh-agent

```
$ eval $(ssh-agent)

$ ssh-add .ssh/id_ed25519
Identity added: .ssh/id_ed25519 (localhost)

$ ssh-add -l
256 SHA256:zCgUHuvA2vSr06RulqTNSk7z2eCAMXqf6LuzYihrB+k localhost (ED25519)
```

ssh-agent

```
$ cat ~/.ssh/config
Host localhost
    IdentityFile ~/.ssh/id_ed25519.pub

$ ssh -i ~/.ssh/id_ed25519.pub root@localhost
```

ssh-agent

`golang.org/x/crypto/ssh/agent`

ssh-tpm-agent

https://github.com/foxboron/ssh-tpm-agent

ssh-tpm-agent

- ssh-agent supporting TPM keys
- Support key creation
    - RSA 2048
    - NIST P256/P358
- openssh key import
- `github.com/google/go-tpm`

```
$ export SSH_AUTH_SOCK="/run/user/1000/ssh-tpm-agent.sock"
$ ssh-tpm-agent &
$ ssh-tpm-keygen
Generating a sealed public/private ecdsa key pair.
Enter file in which to save the key (/home/fox/.ssh/id_ecdsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/fox/.ssh/id_ecdsa.tpm
Your public key has been saved in /home/fox/.ssh/id_ecdsa.pub
The key fingerprint is:
SHA256:NCMJJ2La+q5tGcngQUQvEOJP3gPH8bMP98wJOEMV564
The key's randomart image is the color of television, tuned to a dead channel.
```

```
$ ssh-tpm-add /home/fox/.ssh/id_ecdsa.tpm
Identity added: id_ecdsa.tpm

$ ssh-add -l
256 SHA256:bHnFOGJ/vJetVxa1ncwBu6yoX6Kpj/WgmGu/cP8ZCH0  (ECDSA)
```

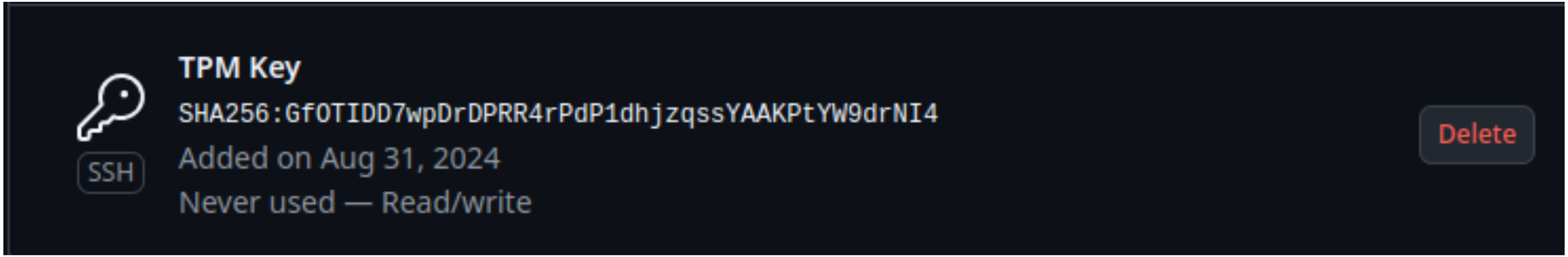
Key import

```
$ ssh-keygen -t ecdsa -f id_ecdsa
[...]

$ ssh-tpm-keygen --import id_ecdsa
Sealing an existing public/private ecdsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_ecdsa.tpm
The key fingerprint is:
SHA256:bDn2EpX6XRX5ADXQSuTq+uUyia/eV3Z6MW+UtxjnXvU
The key's randomart image is the color of television, tuned to a dead channel.
```

ssh-tpm-agent

- Not covered today:
- ssh-agent proxy support
- Hostkey support



TPM Key

`SHA256:GfOTIDD7wpDrDPRR4rPdP1dhjzqssYAAKPtYW9drNI4`

Added on Aug 31, 2024

Never used — Read/write

Github ssh key

```
ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBLXM/
KDMRNt84G78CE0I0TBws2gfF65fA94YBmB57kYs0ZxiHQxykSoxEE6zaPyfgw5IegpkqPz9jOdEH
qqt/bg= fox@framework
```

PASSWORD: 1234

TPM 2.0 Key Files

https://www.hansenpartnership.com/draft-bottomley-tpm2-keys.html

TPM 2.0 Key Files

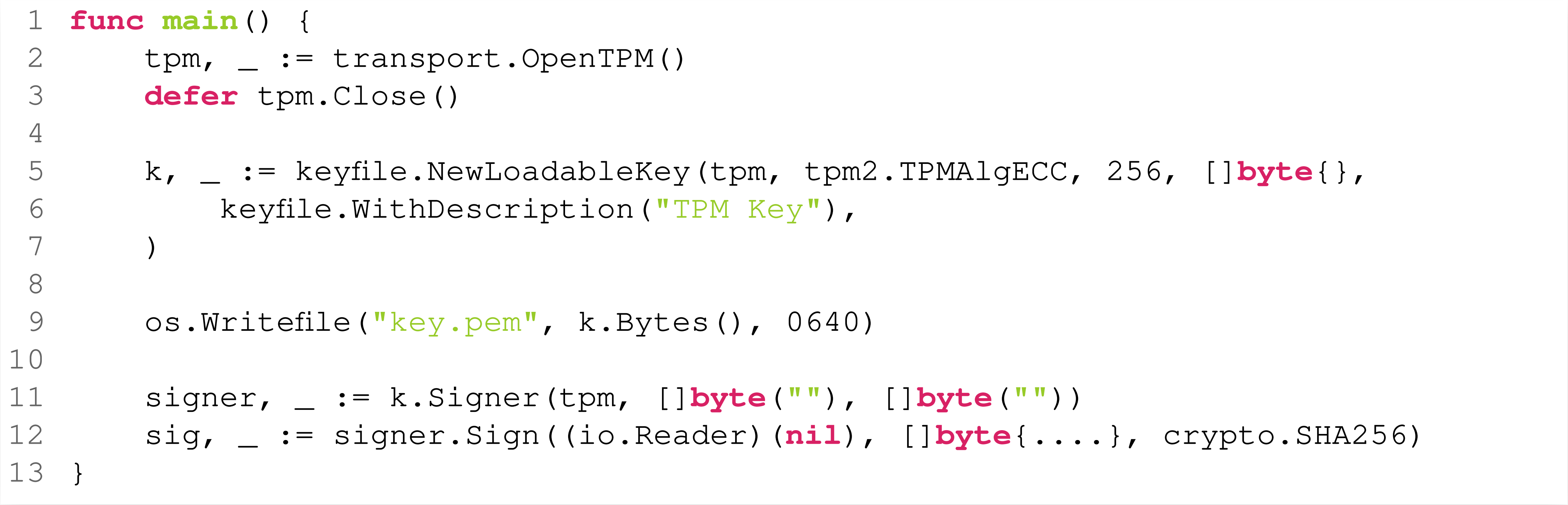
- ASN.1 format for TPM keys
- openssl `tpm2` provider
- Linux keyring support

TPM 2.0 Key Files

- Support different key types
  - Loadable Keys
  - Importable Keys
  - Sealed Keys

https://github.com/Foxboron/go-tpm-keyfiles

Go API



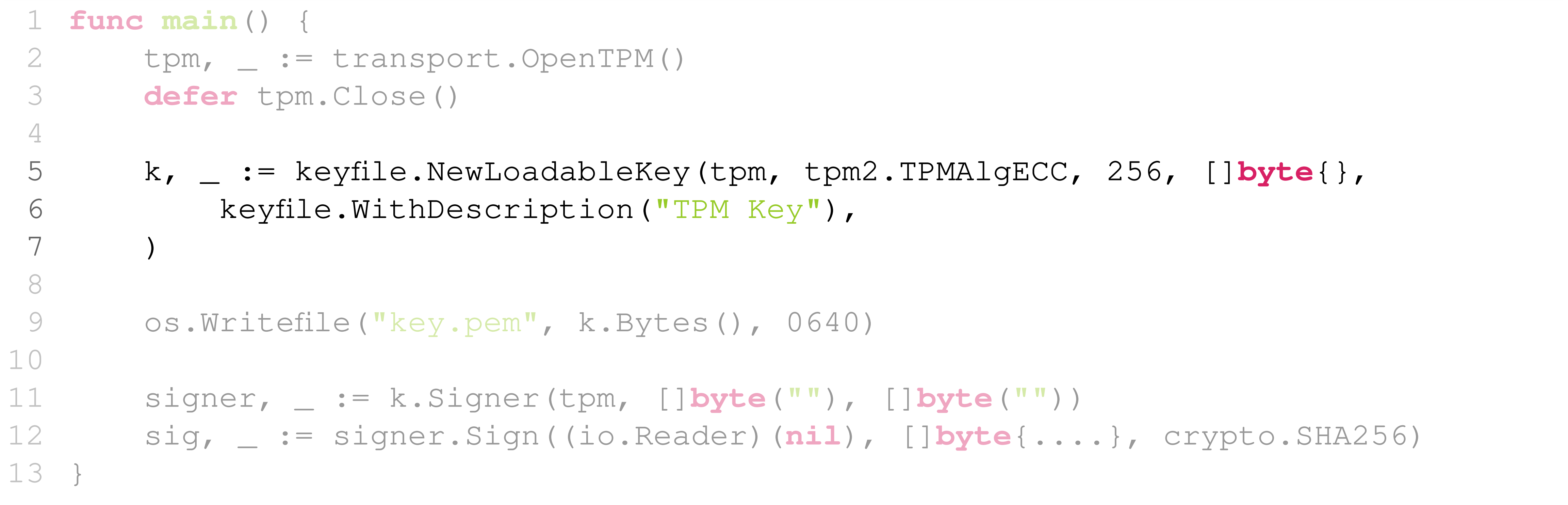
```go
func main() {
    tpm, _ := transport.OpenTPM()
    defer tpm.Close()

    k, _ := keyfile.NewLoadableKey(tpm, tpm2.TPMAlgECC, 256, []byte{},
        keyfile.WithDescription("TPM Key"),
    )

    os.Writefile("key.pem", k.Bytes(), 0640)

    signer, _ := k.Signer(tpm, []byte(""), []byte(""))
    sig, _ := signer.Sign((io.Reader)(nil), []byte{....}, crypto.SHA256)
}
```

Go API



```
func main() {
    tpm, _ := transport.OpenTPM()
    defer tpm.Close()

    k, _ := keyfile.NewLoadableKey(tpm, tpm2.TPMAlgECC, 256, []byte{},
        keyfile.WithDescription("TPM Key"),
    )

    os.Writefile("key.pem", k.Bytes(), 0640)

    signer, _ := k.Signer(tpm, []byte(""), []byte(""))
    sig, _ := signer.Sign((io.Reader)(nil), []byte{....}, crypto.SHA256)
}
```

Go API

```go
func main() {
    tpm, _ := transport.OpenTPM()
    defer tpm.Close()

    k, _ := keyfile.NewLoadableKey(tpm, tpm2.TPMAlgECC, 256, []byte{},
        keyfile.WithDescription("TPM Key"),
    )

    os.Writefile("key.pem", k.Bytes(), 0640)

    signer, _ := k.Signer(tpm, []byte(""), []byte(""))
    sig, _ := signer.Sign((io.Reader)(nil), []byte{....}, crypto.SHA256)
}
```

Go API

```go
func main() {
    tpm, _ := transport.OpenTPM()
    defer tpm.Close()

    k, _ := keyfile.NewLoadableKey(tpm, tpm2.TPMAlgECC, 256, []byte{},
        keyfile.WithDescription("TPM Key"),
    )

    os.Writefile("key.pem", k.Bytes(), 0640)

    signer, _ := k.Signer(tpm, []byte(""), []byte(""))
    sig, _ := signer.Sign((io.Reader)(nil), []byte{....}, crypto.SHA256)
}
```

Go API

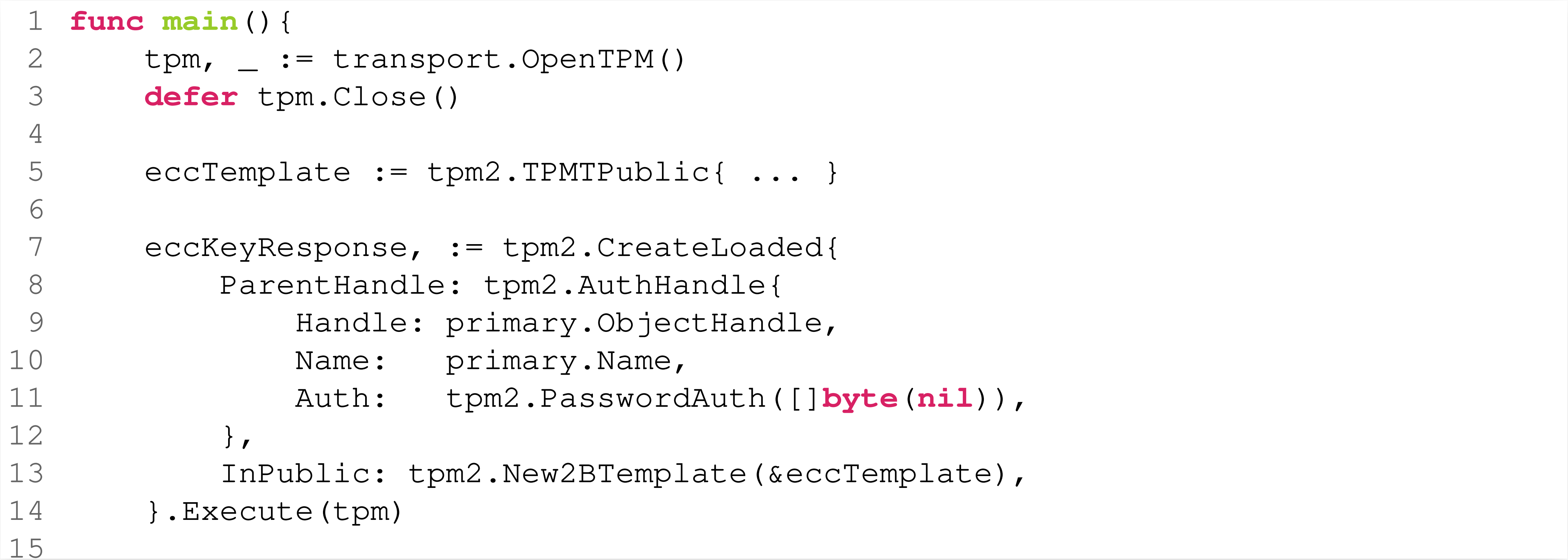
```go
func main() {
    tpm, _ := transport.OpenTPM()
    defer tpm.Close()

    k, _ := keyfile.NewLoadableKey(tpm, tpm2.TPMAlgECC, 256, []byte{},
        keyfile.WithDescription("TPM Key"),
    )

    os.Writefile("key.pem", k.Bytes(), 0640)

    signer, _ := k.Signer(tpm, []byte(""), []byte(""))
    sig, _ := signer.Sign((io.Reader)(nil), []byte{....}, crypto.SHA256)
}
```

NewTPMKey



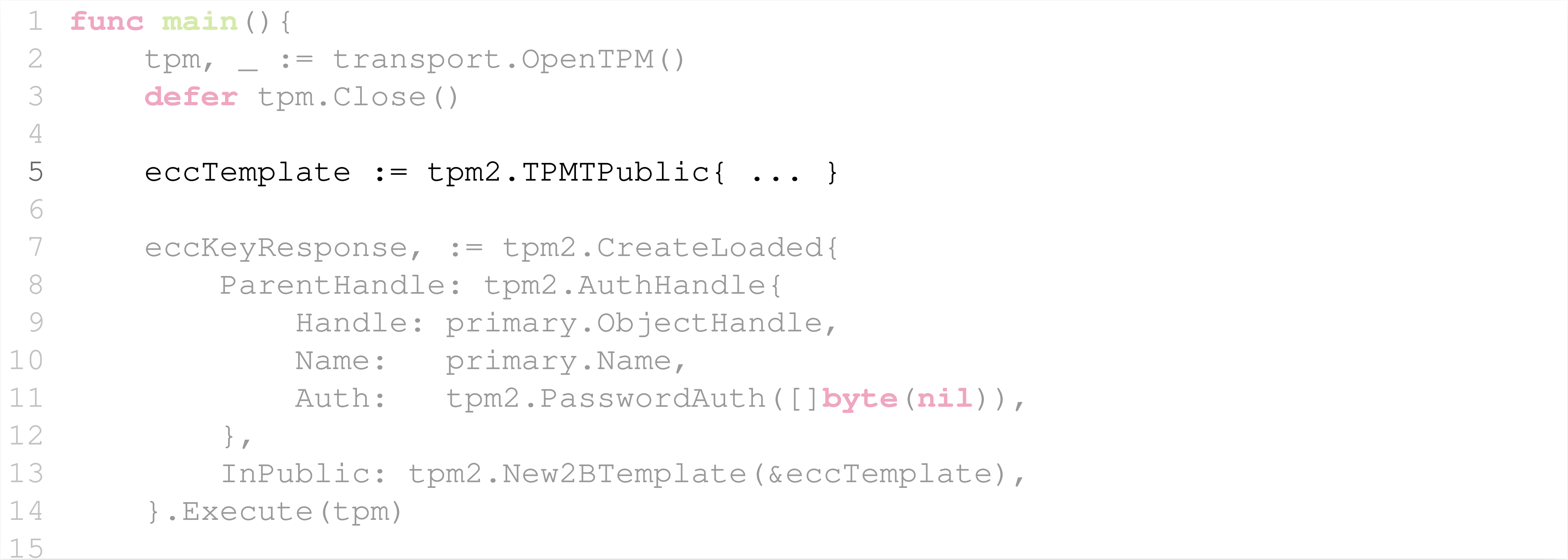
```go
func main(){
    tpm, _ := transport.OpenTPM()
    defer tpm.Close()

    eccTemplate := tpm2.TPMTPublic{ ... }

    eccKeyResponse, := tpm2.CreateLoaded{
        ParentHandle: tpm2.AuthHandle{
            Handle: primary.ObjectHandle,
            Name:   primary.Name,
            Auth:   tpm2.PasswordAuth([]byte(nil)),
        },
        InPublic: tpm2.New2BTemplate(&eccTemplate),
    }.Execute(tpm)

```

NewTPMKey



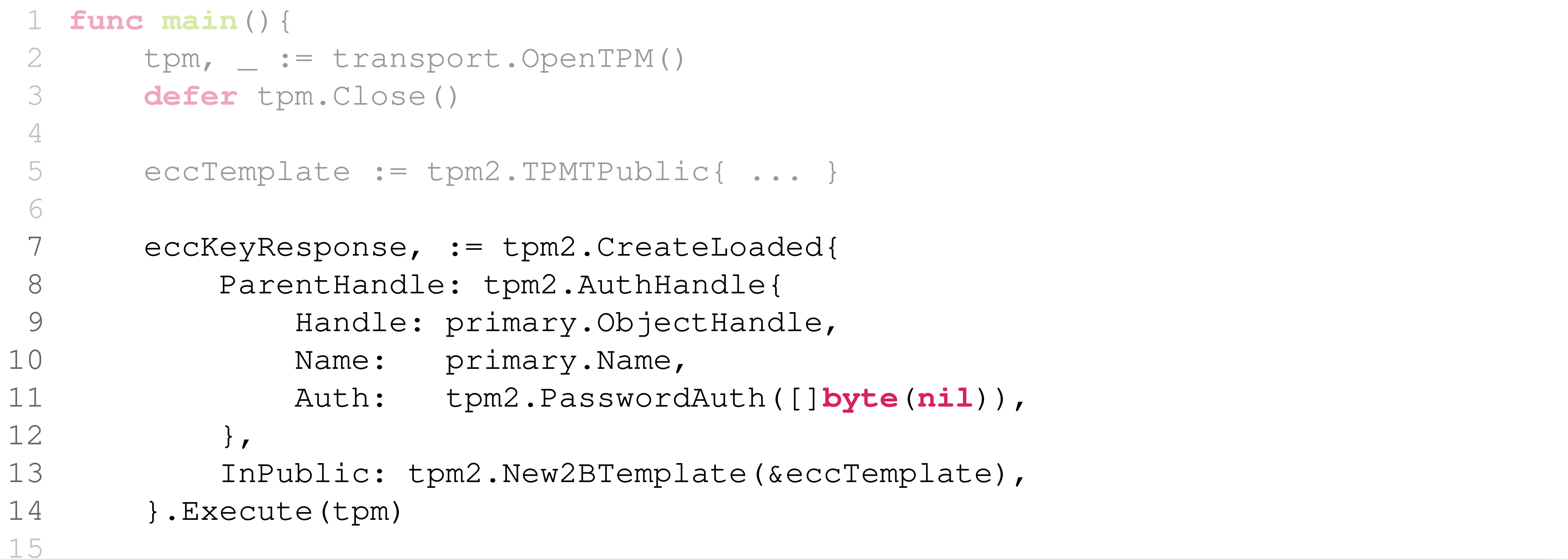
```go
func main(){
    tpm, _ := transport.OpenTPM()
    defer tpm.Close()

    eccTemplate := tpm2.TPMTPublic{ ... }

    eccKeyResponse, := tpm2.CreateLoaded{
        ParentHandle: tpm2.AuthHandle{
            Handle: primary.ObjectHandle,
            Name:   primary.Name,
            Auth:   tpm2.PasswordAuth([]byte(nil)),
        },
        InPublic: tpm2.New2BTemplate(&eccTemplate),
    }.Execute(tpm)

```

NewTPMKey



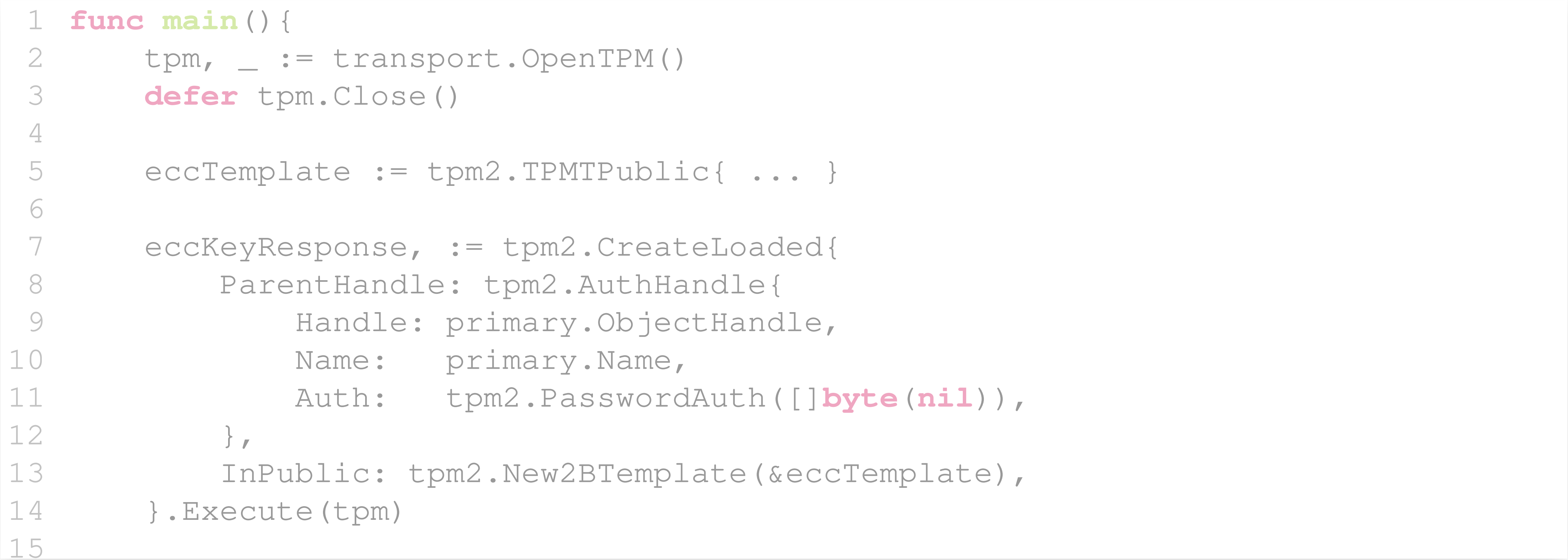
```
func main(){
    tpm, _ := transport.OpenTPM()
    defer tpm.Close()

    eccTemplate := tpm2.TPMTPublic{ ... }

    eccKeyResponse, := tpm2.CreateLoaded{
        ParentHandle: tpm2.AuthHandle{
            Handle: primary.ObjectHandle,
            Name:   primary.Name,
            Auth:   tpm2.PasswordAuth([]byte(nil)),
        },
        InPublic: tpm2.New2BTemplate(&eccTemplate),
    }.Execute(tpm)

```

NewTPMKey



```go
func main(){
    tpm, _ := transport.OpenTPM()
    defer tpm.Close()

    eccTemplate := tpm2.TPMTPublic{ ... }

    eccKeyResponse, := tpm2.CreateLoaded{
        ParentHandle: tpm2.AuthHandle{
            Handle: primary.ObjectHandle,
            Name:   primary.Name,
            Auth:   tpm2.PasswordAuth([]byte(nil)),
        },
        InPublic: tpm2.New2BTemplate(&eccTemplate),
    }.Execute(tpm)

```

NewTPMKey

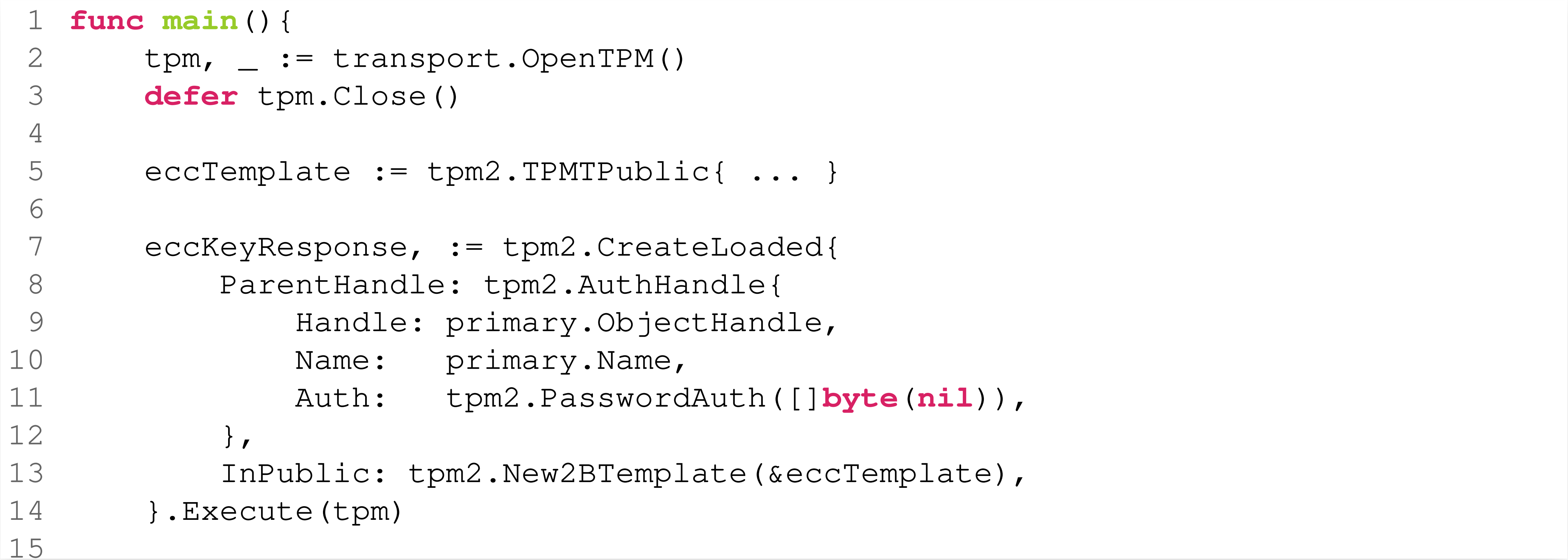
```go
func main(){
    tpm, _ := transport.OpenTPM()
    defer tpm.Close()

    eccTemplate := tpm2.TPMTPublic{ ... }

    eccKeyResponse, := tpm2.CreateLoaded{
        ParentHandle: tpm2.AuthHandle{
            Handle: primary.ObjectHandle,
            Name:   primary.Name,
            Auth:   tpm2.PasswordAuth([]byte(nil)),
        },
        InPublic: tpm2.New2BTemplate(&eccTemplate),
    }.Execute(tpm)

```

NewTPMKey



```go
func main(){
    tpm, _ := transport.OpenTPM()
    defer tpm.Close()

    eccTemplate := tpm2.TPMTPublic{ ... }

    eccKeyResponse, := tpm2.CreateLoaded{
        ParentHandle: tpm2.AuthHandle{
            Handle: primary.ObjectHandle,
            Name:   primary.Name,
            Auth:   tpm2.PasswordAuth([]byte(nil)),
        },
        InPublic: tpm2.New2BTemplate(&eccTemplate),
    }.Execute(tpm)

```

openssl key creation with ssh-tpm-agent

```
$ openssl genpkey -provider tpm2 \
    -algorithm EC -pkeyopt group:P-256 \
    -pkeyopt user-auth:1234 \
    -out id_ecdsa.tpm

$ ssh-tpm-keygen --print-pubkey ./id_ecdsa.tpm
ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHA[...]Tkw=
```

openssl key creation with ssh-tpm-agent

```
$ openssl genpkey -provider tpm2 \
    -algorithm EC -pkeyopt group:P-256 \
    -pkeyopt user-auth:1234 \
    -out id_ecdsa.tpm

$ ssh-tpm-keygen --print-pubkey ./id_ecdsa.tpm
ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHA[...]Tkw=
```

openssl key creation with ssh-tpm-agent

```
$ openssl genpkey -provider tpm2 \
    -algorithm EC -pkeyopt group:P-256 \
    -pkeyopt user-auth:1234 \
    -out id_ecdsa.tpm

$ ssh-tpm-keygen --print-pubkey ./id_ecdsa.tpm
ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHA[...]Tkw=
```

openssl key creation with ssh-tpm-agent

```
$ openssl genpkey -provider tpm2 \
    -algorithm EC -pkeyopt group:P-256 \
    -pkeyopt user-auth:1234 \
    -out id_ecdsa.tpm

$ ssh-tpm-keygen --print-pubkey ./id_ecdsa.tpm
ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHA[...]Tkw=
```

Goals

- ✅ Device bound keys
- Identity claims
- Machine identity

SSH Certificate Authority

SSH certificates

SSH certificates

- Principals - users
- Capabilities
- Time limit/lifetime

SSH certificate - Go example

```go
after := time.Now()
before := after.Add(time.Minute * 5)

sshkey, _ := ssh.NewPublicKey(...)

certificate := ssh.Certificate{
    Key:             sshkey,
    CertType:        ssh.UserCert,
    ValidPrincipals: []string{"Foxboron"},
    KeyId:           "TPM Key",
    ValidAfter:      uint64(after.Unix()),
    ValidBefore:     uint64(before.Unix()),
    Permissions: ssh.Permissions{
        Extensions: map[string]string{
            "permit-agent-forwarding": "",
```

SSH certificate - Go example

```go
after := time.Now()
before := after.Add(time.Minute * 5)

sshkey, _ := ssh.NewPublicKey(...)

certificate := ssh.Certificate{
    Key:             sshkey,
    CertType:        ssh.UserCert,
    ValidPrincipals: []string{"Foxboron"},
    KeyId:           "TPM Key",
    ValidAfter:      uint64(after.Unix()),
    ValidBefore:     uint64(before.Unix()),
    Permissions: ssh.Permissions{
        Extensions: map[string]string{
            "permit-agent-forwarding": "",
```

SSH certificate - Go example

```go
after := time.Now()
before := after.Add(time.Minute * 5)

sshkey, _ := ssh.NewPublicKey(...)

certificate := ssh.Certificate{
    Key:             sshkey,
    CertType:        ssh.UserCert,
    ValidPrincipals: []string{"Foxboron"},
    KeyId:           "TPM Key",
    ValidAfter:      uint64(after.Unix()),
    ValidBefore:     uint64(before.Unix()),
    Permissions: ssh.Permissions{
        Extensions: map[string]string{
            "permit-agent-forwarding": "",
```

SSH certificate - Go example

```go
after := time.Now()
before := after.Add(time.Minute * 5)

sshkey, _ := ssh.NewPublicKey(...)

certificate := ssh.Certificate{
    Key:             sshkey,
    CertType:        ssh.UserCert,
    ValidPrincipals: []string{"Foxboron"},
    KeyId:           "TPM Key",
    ValidAfter:      uint64(after.Unix()),
    ValidBefore:     uint64(before.Unix()),
    Permissions: ssh.Permissions{
        Extensions: map[string]string{
            "permit-agent-forwarding": "",
```

SSH certificate - Go example

```go
after := time.Now()
before := after.Add(time.Minute * 5)

sshkey, _ := ssh.NewPublicKey(...)

certificate := ssh.Certificate{
    Key:             sshkey,
    CertType:        ssh.UserCert,
    ValidPrincipals: []string{"Foxboron"},
    KeyId:           "TPM Key",
    ValidAfter:      uint64(after.Unix()),
    ValidBefore:     uint64(before.Unix()),
    Permissions: ssh.Permissions{
        Extensions: map[string]string{
            "permit-agent-forwarding": "",
```

SSH certificate - Go example

```
after := time.Now()
before := after.Add(time.Minute * 5)

sshkey, _ := ssh.NewPublicKey(...)

certificate := ssh.Certificate{
    Key:             sshkey,
    CertType:        ssh.UserCert,
    ValidPrincipals: []string{"Foxboron"},
    KeyId:           "TPM Key",
    ValidAfter:      uint64(after.Unix()),
    ValidBefore:     uint64(before.Unix()),
    Permissions: ssh.Permissions{
        Extensions: map[string]string{
            "permit-agent-forwarding": "",
```

SSH certificate - Go example

```
after := time.Now()
before := after.Add(time.Minute * 5)

sshkey, _ := ssh.NewPublicKey(...)

certificate := ssh.Certificate{
    Key:             sshkey,
    CertType:        ssh.UserCert,
    ValidPrincipals: []string{"Foxboron"},
    KeyId:           "TPM Key",
    ValidAfter:      uint64(after.Unix()),
    ValidBefore:     uint64(before.Unix()),
    Permissions: ssh.Permissions{
        Extensions: map[string]string{
            "permit-agent-forwarding": "",
```

SSH certificate - Go example

```go
after := time.Now()
before := after.Add(time.Minute * 5)

sshkey, _ := ssh.NewPublicKey(...)

certificate := ssh.Certificate{
    Key:             sshkey,
    CertType:        ssh.UserCert,
    ValidPrincipals: []string{"Foxboron"},
    KeyId:           "TPM Key",
    ValidAfter:      uint64(after.Unix()),
    ValidBefore:     uint64(before.Unix()),
    Permissions: ssh.Permissions{
        Extensions: map[string]string{
            "permit-agent-forwarding": "",
```

SSH certificate - Go example

```go
after := time.Now()
before := after.Add(time.Minute * 5)

sshkey, _ := ssh.NewPublicKey(...)

certificate := ssh.Certificate{
    Key:             sshkey,
    CertType:        ssh.UserCert,
    ValidPrincipals: []string{"Foxboron"},
    KeyId:           "TPM Key",
    ValidAfter:      uint64(after.Unix()),
    ValidBefore:     uint64(before.Unix()),
    Permissions: ssh.Permissions{
        Extensions: map[string]string{
            "permit-agent-forwarding": "",
```

SSH certificates

```
TrustedUserCAKeys /etc/ssh/ca_user_key.pub
```

Identity claims

Identity claims

- Identity Provider (IdP)
- Information about a given user
- Signed
- Open ID Connect (OIDC)
- Proves access to a user

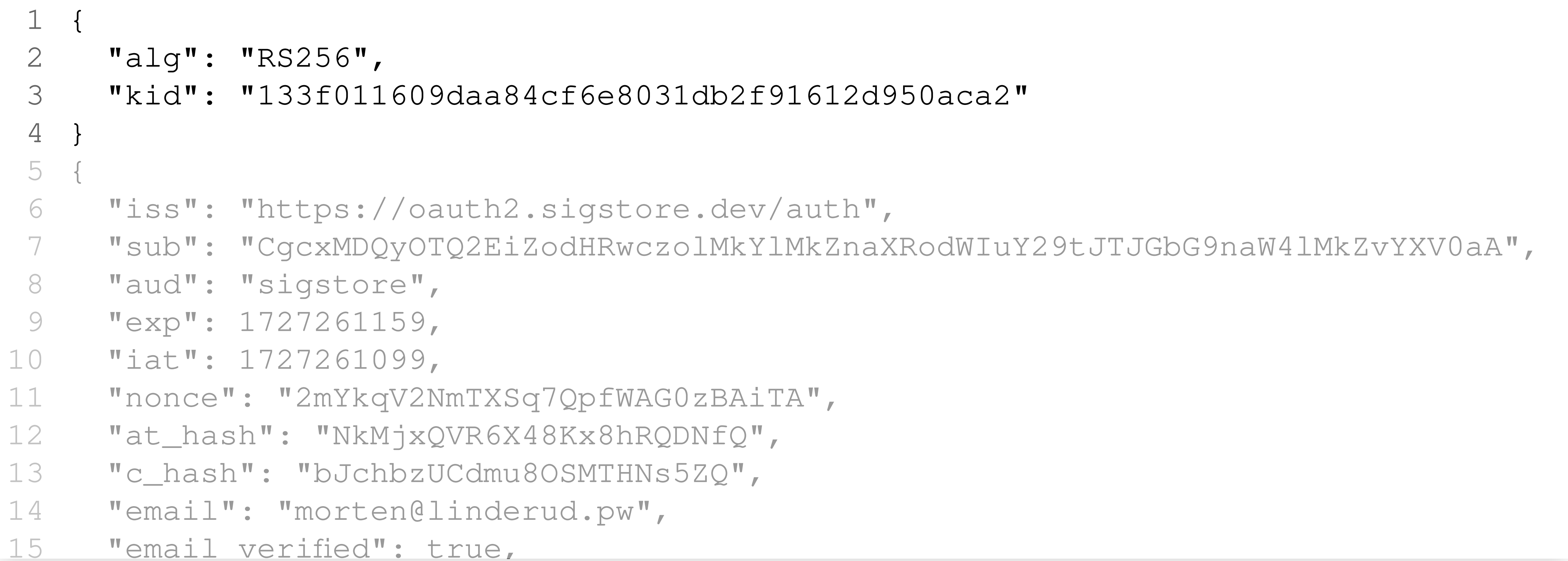
Open ID Connect

- JWTs
- Signed by IdP
- Can be validated remotely

JWT Example

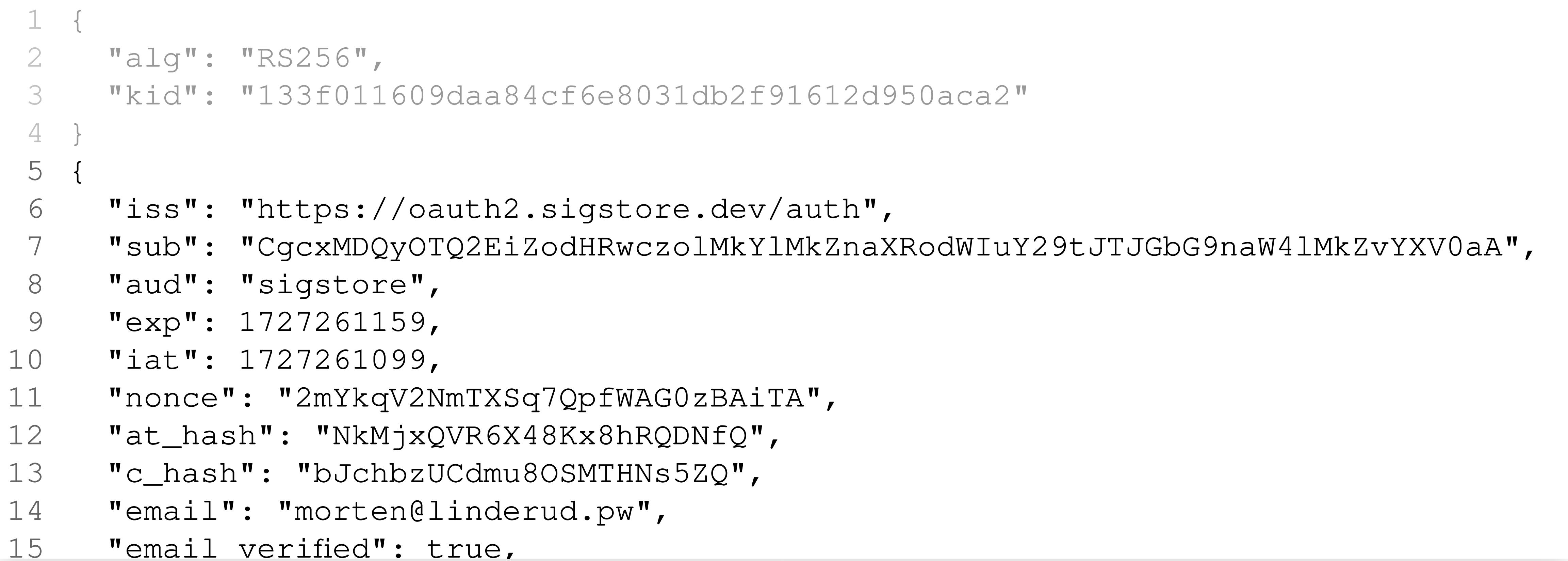
```
{
  "alg": "RS256",
  "kid": "133f011609daa84cf6e8031db2f91612d950aca2"
}
{
  "iss": "https://oauth2.sigstore.dev/auth",
  "sub": "CgcxMDQyOTQ2EiZodHRwczolMkYlMkZnaXRodWIuY29tJTJGbG9naW4lMkZvYXV0aA",
  "aud": "sigstore",
  "exp": 1727261159,
  "iat": 1727261099,
  "nonce": "2mYkqV2NmTXSq7QpfWAG0zBAiTA",
  "at_hash": "NkMjxQVR6X48Kx8hRQDNfQ",
  "c_hash": "bJchbzUCdmu8OSMTHNs5ZQ",
  "email": "morten@linderud.pw",
  "email_verified": true,
```

JWT Example



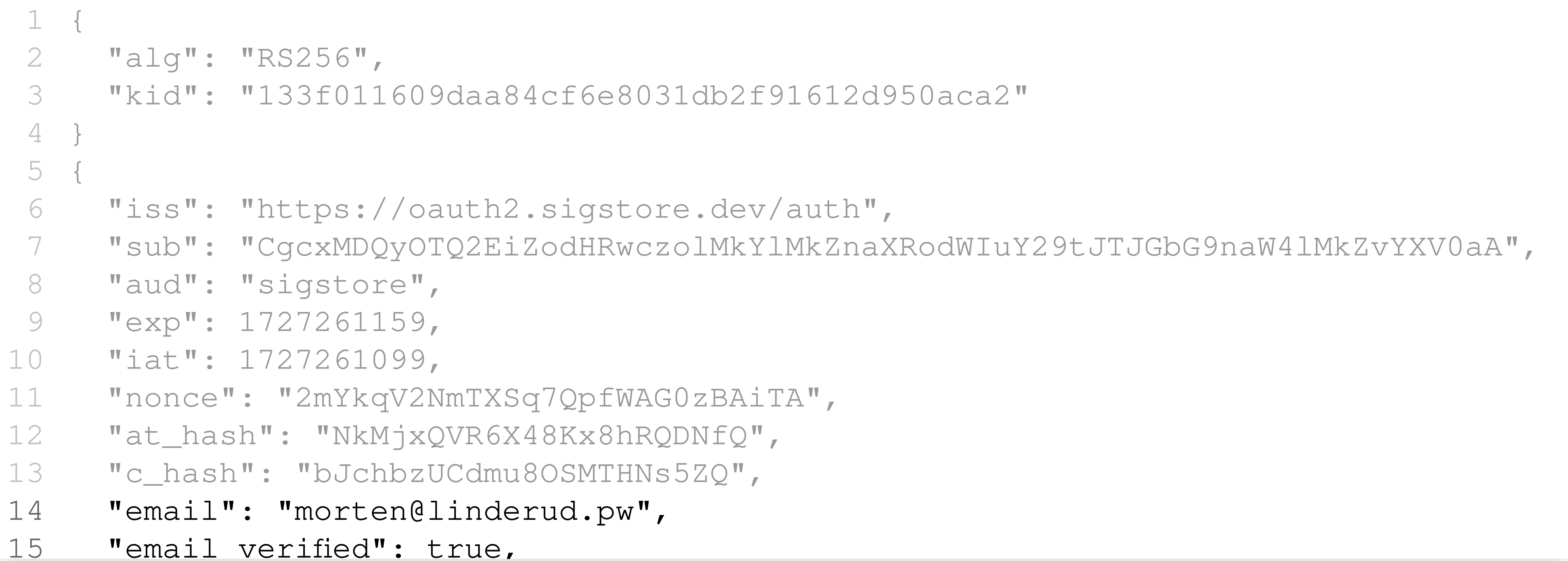
```
{
  "alg": "RS256",
  "kid": "133f011609daa84cf6e8031db2f91612d950aca2"
}
{
  "iss": "https://oauth2.sigstore.dev/auth",
  "sub": "CgcxMDQyOTQ2EiZodHRwczolMkYlMkZnaXRodWIuY29tJTJGbG9naW4lMkZvYXV0aA",
  "aud": "sigstore",
  "exp": 1727261159,
  "iat": 1727261099,
  "nonce": "2mYkqV2NmTXSq7QpfWAG0zBAiTA",
  "at_hash": "NkMjxQVR6X48Kx8hRQDNfQ",
  "c_hash": "bJchbzUCdmu8OSMTHNs5ZQ",
  "email": "morten@linderud.pw",
  "email_verified": true,
```

JWT Example



```
{
  "alg": "RS256",
  "kid": "133f011609daa84cf6e8031db2f91612d950aca2"
}
{
  "iss": "https://oauth2.sigstore.dev/auth",
  "sub": "CgcxMDQyOTQ2EiZodHRwczolMkYlMkZnaXRodWIuY29tJTJGbG9naW4lMkZvYXV0aA",
  "aud": "sigstore",
  "exp": 1727261159,
  "iat": 1727261099,
  "nonce": "2mYkqV2NmTXSq7QpfWAG0zBAiTA",
  "at_hash": "NkMjxQVR6X48Kx8hRQDNfQ",
  "c_hash": "bJchbzUCdmu8OSMTHNs5ZQ",
  "email": "morten@linderud.pw",
  "email_verified": true,
```

JWT Example



```
{
  "alg": "RS256",
  "kid": "133f011609daa84cf6e8031db2f91612d950aca2"
}
{
  "iss": "https://oauth2.sigstore.dev/auth",
  "sub": "CgcxMDQyOTQ2EiZodHRwczolMkYlMkZnaXRodWIuY29tJTJGbG9naW4lMkZvYXV0aA",
  "aud": "sigstore",
  "exp": 1727261159,
  "iat": 1727261099,
  "nonce": "2mYkqV2NmTXSq7QpfWAG0zBAiTA",
  "at_hash": "NkMjxQVR6X48Kx8hRQDNfQ",
  "c_hash": "bJchbzUCdmu8OSMTHNs5ZQ",
  "email": "morten@linderud.pw",
  "email_verified": true,
```

JWT Example

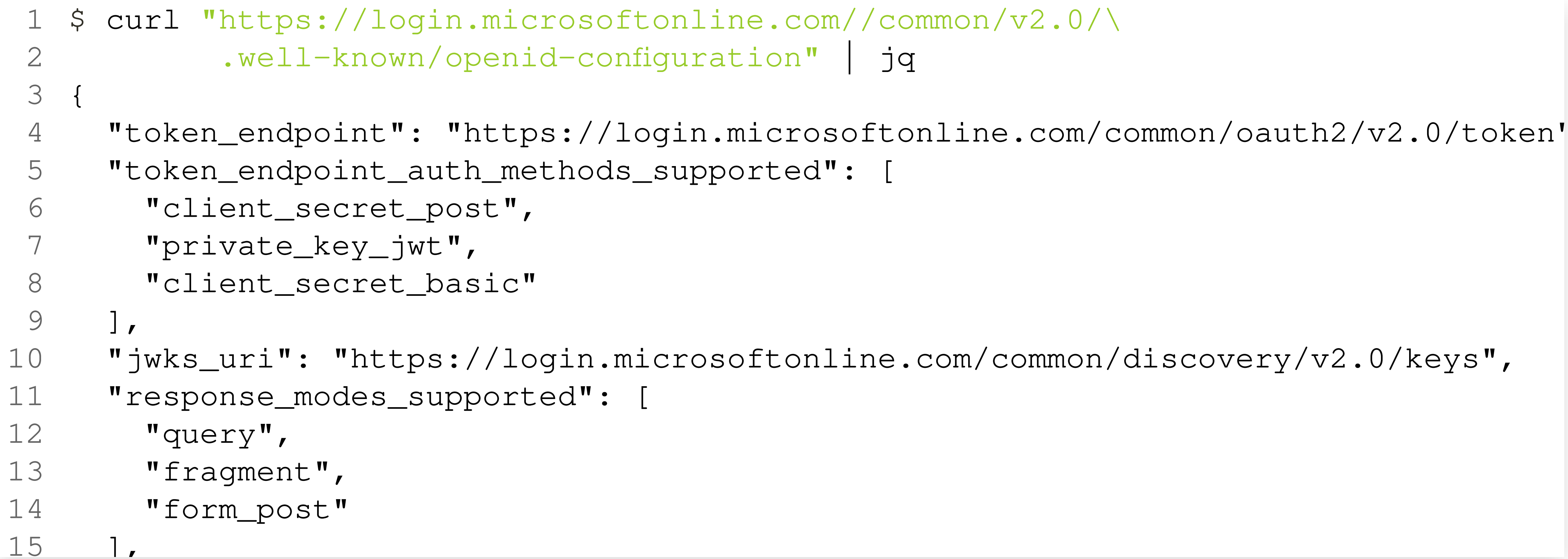
```
{
  "alg": "RS256",
  "kid": "133f011609daa84cf6e8031db2f91612d950aca2"
}
{
  "iss": "https://oauth2.sigstore.dev/auth",
  "sub": "CgcxMDQyOTQ2EiZodHRwczolMkYlMkZnaXRodWIuY29tJTJGbG9naW4lMkZvYXV0aA",
  "aud": "sigstore",
  "exp": 1727261159,
  "iat": 1727261099,
  "nonce": "2mYkqV2NmTXSq7QpfWAG0zBAiTA",
  "at_hash": "NkMjxQVR6X48Kx8hRQDNfQ",
  "c_hash": "bJchbzUCdmu8OSMTHNs5ZQ",
  "email": "morten@linderud.pw",
  "email_verified": true,
```

Identity Provider (IdP)

Identity Provider

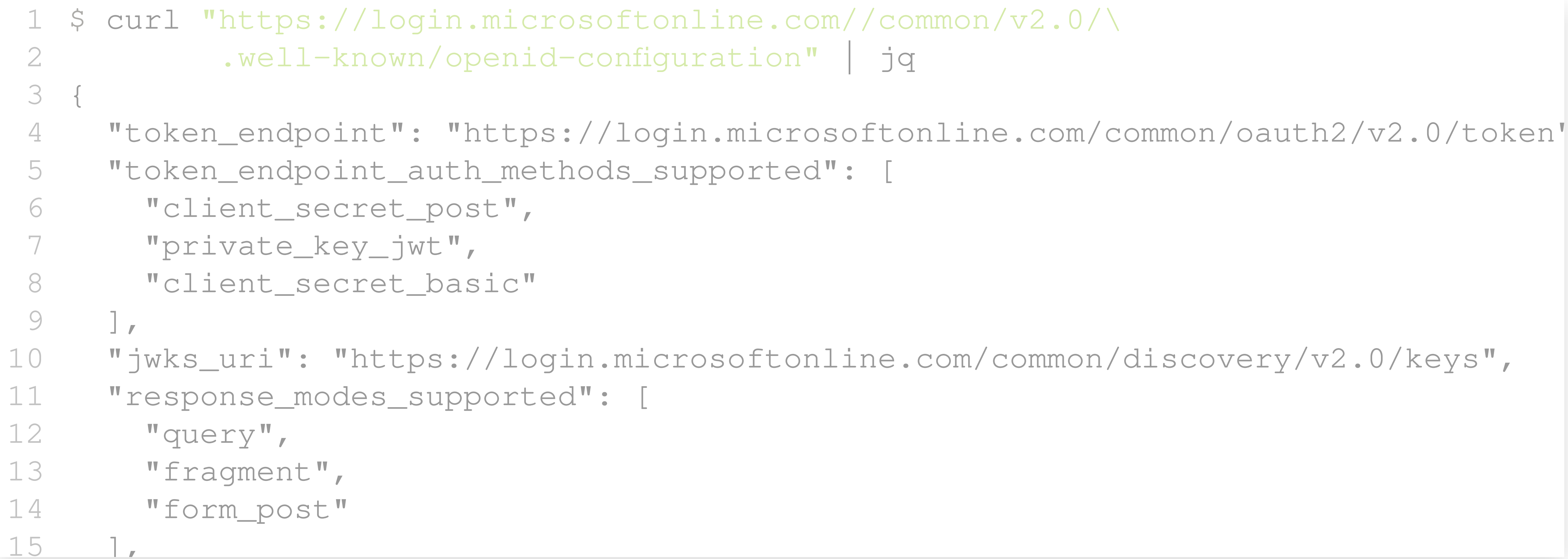
- Keycloak, Okta, Microsoft, Google
- Different claims pr provider
- Github

Identity Provider config



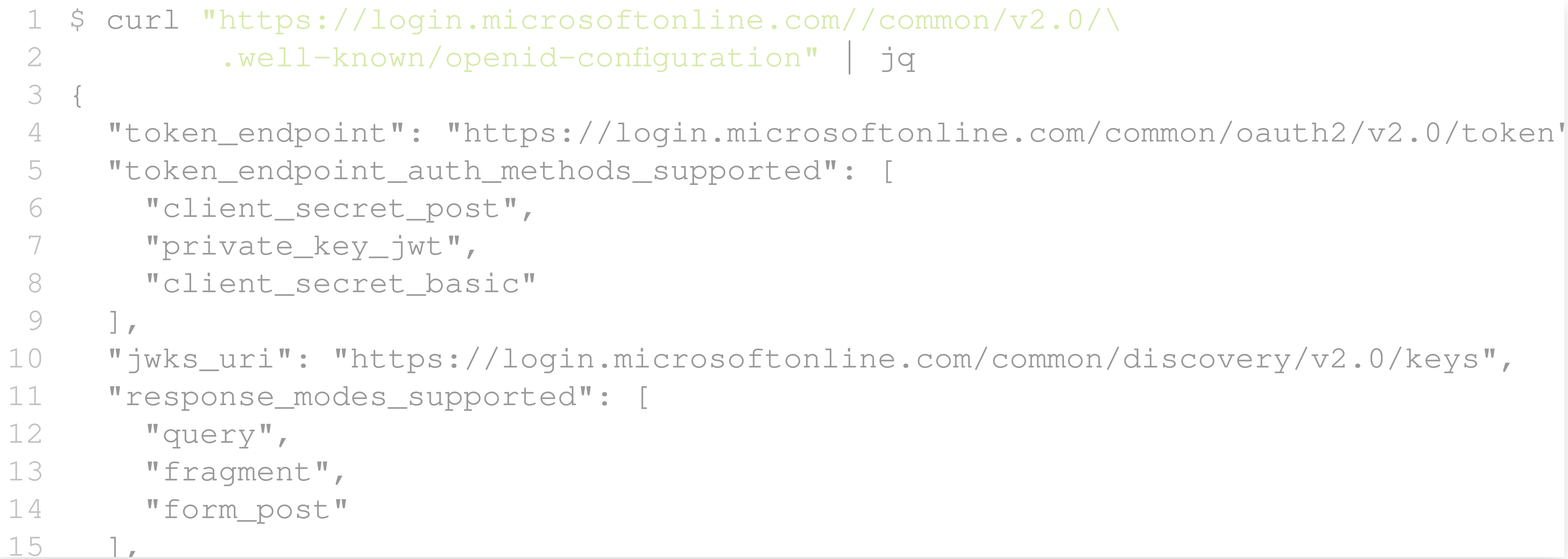
```
$ curl "https://login.microsoftonline.com//common/v2.0/\
        .well-known/openid-configuration" | jq
{
  "token_endpoint": "https://login.microsoftonline.com/common/oauth2/v2.0/token"
  "token_endpoint_auth_methods_supported": [
    "client_secret_post",
    "private_key_jwt",
    "client_secret_basic"
  ],
  "jwks_uri": "https://login.microsoftonline.com/common/discovery/v2.0/keys",
  "response_modes_supported": [
    "query",
    "fragment",
    "form_post"
  ],
```

Identity Provider config



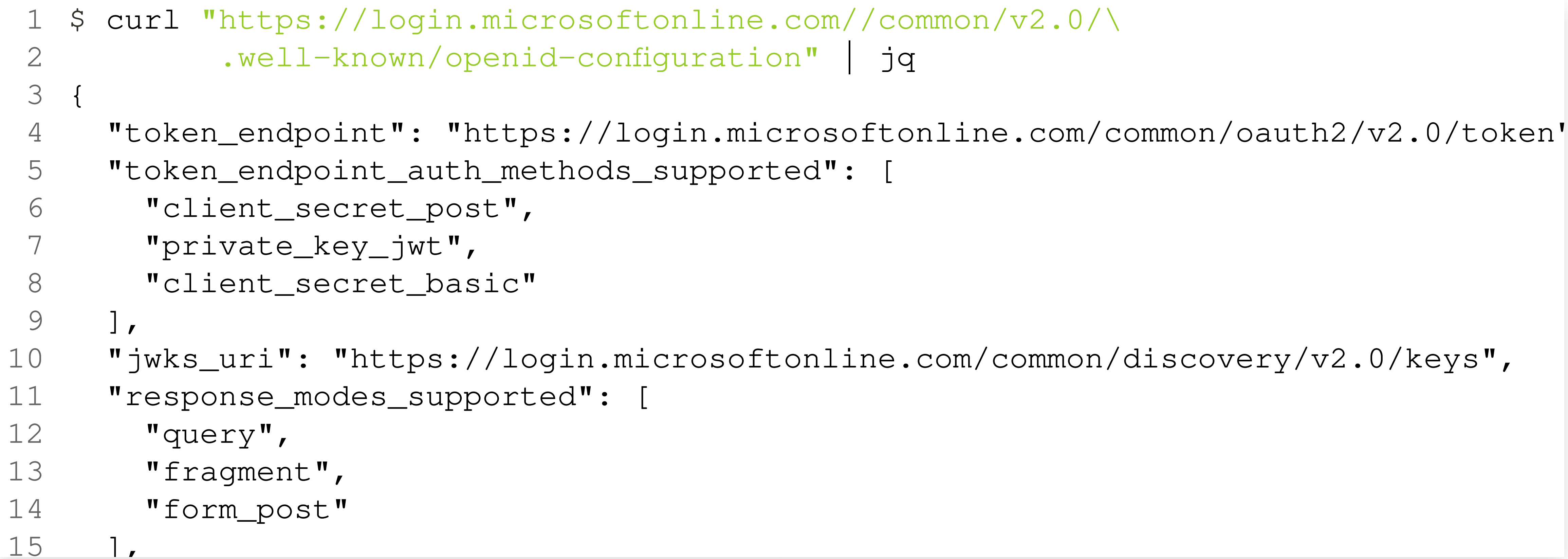
```
$ curl "https://login.microsoftonline.com//common/v2.0/\
       .well-known/openid-configuration" | jq
{
  "token_endpoint": "https://login.microsoftonline.com/common/oauth2/v2.0/token"
  "token_endpoint_auth_methods_supported": [
    "client_secret_post",
    "private_key_jwt",
    "client_secret_basic"
  ],
  "jwks_uri": "https://login.microsoftonline.com/common/discovery/v2.0/keys",
  "response_modes_supported": [
    "query",
    "fragment",
    "form_post"
  ],
```

Identity Provider config



```
$ curl "https://login.microsoftonline.com//common/v2.0/\
        .well-known/openid-configuration" | jq
{
  "token_endpoint": "https://login.microsoftonline.com/common/oauth2/v2.0/token"
  "token_endpoint_auth_methods_supported": [
    "client_secret_post",
    "private_key_jwt",
    "client_secret_basic"
  ],
  "jwks_uri": "https://login.microsoftonline.com/common/discovery/v2.0/keys",
  "response_modes_supported": [
    "query",
    "fragment",
    "form_post"
  ],
```

Identity Provider config



```
$ curl "https://login.microsoftonline.com//common/v2.0/\
        .well-known/openid-configuration" | jq
{
  "token_endpoint": "https://login.microsoftonline.com/common/oauth2/v2.0/token"
  "token_endpoint_auth_methods_supported": [
    "client_secret_post",
    "private_key_jwt",
    "client_secret_basic"
  ],
  "jwks_uri": "https://login.microsoftonline.com/common/discovery/v2.0/keys",
  "response_modes_supported": [
    "query",
    "fragment",
    "form_post"
  ],
```

Identity Claims

- How can we know the IdP verifies identities?
- `acr` is not standardized
- `acr_values_supported` is optional
- Values `silver` and `bronze`

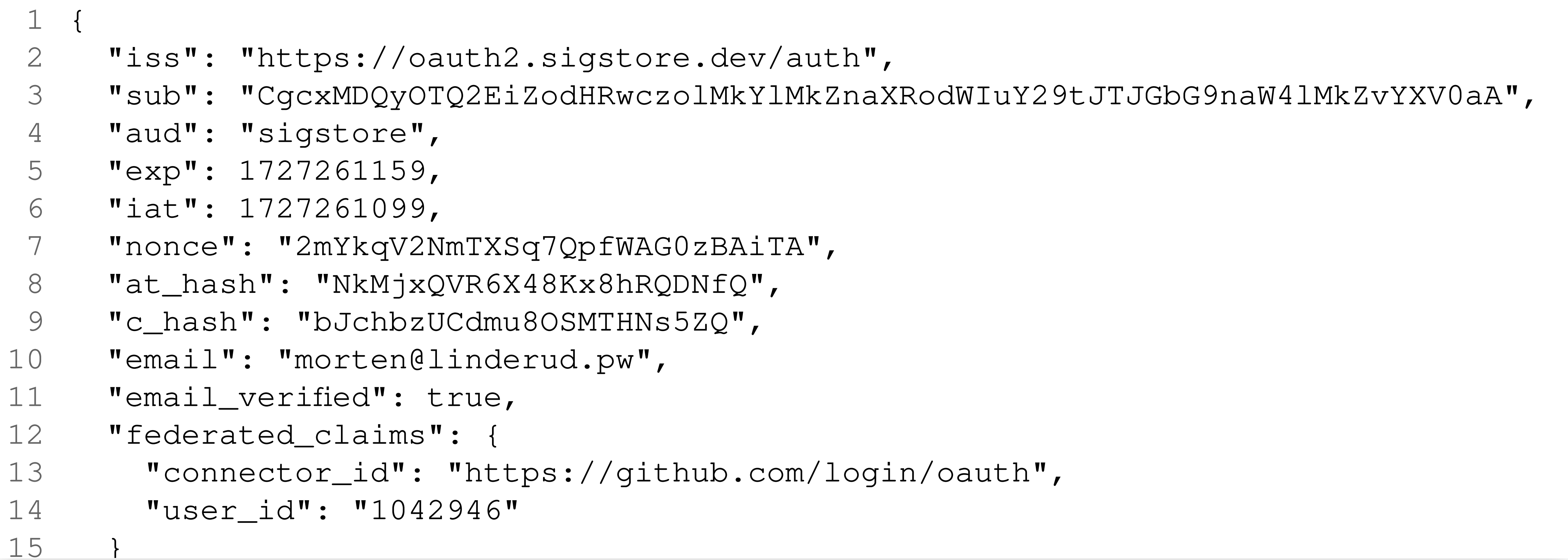
Sigstore

https://www.sigstore.dev/

DEX

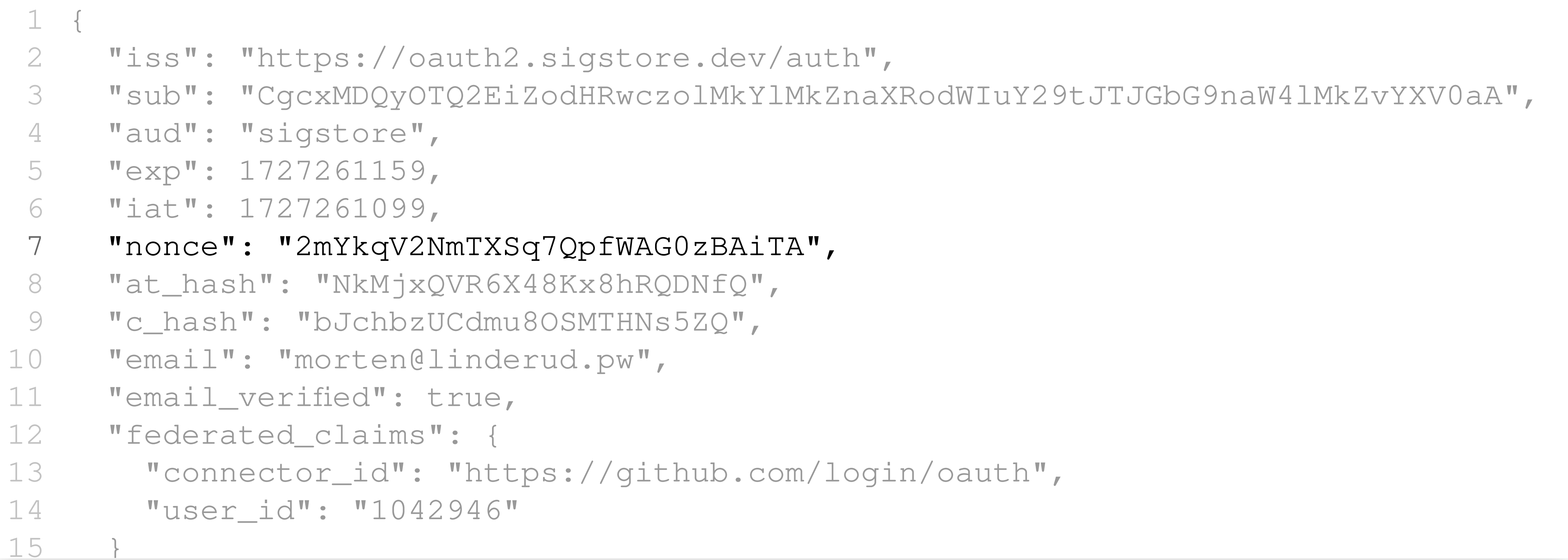
- Federated IdP
- Support Microsoft, Google
- Github(!)

How do we know the authentication is from the CA?



```
{
  "iss": "https://oauth2.sigstore.dev/auth",
  "sub": "CgcxMDQyOTQ2EiZodHRwczolMkYlMkZnaXRodWIuY29tJTJGbG9naW4lMkZvYXV0aA",
  "aud": "sigstore",
  "exp": 1727261159,
  "iat": 1727261099,
  "nonce": "2mYkqV2NmTXSq7QpfWAG0zBAiTA",
  "at_hash": "NkMjxQVR6X48Kx8hRQDNfQ",
  "c_hash": "bJchbzUCdmu8OSMTHNs5ZQ",
  "email": "morten@linderud.pw",
  "email_verified": true,
  "federated_claims": {
    "connector_id": "https://github.com/login/oauth",
    "user_id": "1042946"
  }
```

]



```
{
  "iss": "https://oauth2.sigstore.dev/auth",
  "sub": "CgcxMDQyOTQ2EiZodHRwczolMkYlMkZnaXRodWIuY29tJTJGbG9naW4lMkZvYXV0aA",
  "aud": "sigstore",
  "exp": 1727261159,
  "iat": 1727261099,
  "nonce": "2mYkqV2NmTXSq7QpfWAG0zBAiTA",
  "at_hash": "NkMjxQVR6X48Kx8hRQDNfQ",
  "c_hash": "bJchbzUCdmu8OSMTHNs5ZQ",
  "email": "morten@linderud.pw",
  "email_verified": true,
  "federated_claims": {
    "connector_id": "https://github.com/login/oauth",
    "user_id": "1042946"
  }
```

JWT nonce abuse!

JWT nonce abuse

- Value from the CA
- Signed by the IdP
- Probably proves it comes from us

Goals

- ✅ Device bound keys
- ✅ Identity claims
- Machine identity

Machine identities

Machine identities

- Proves access to a given machine
- Doesn't circumvent MDM

Machine identities

- Endorsement keys can't sign things
- Attestation Key
- Credential Protection

TPM2_Certify and Attestation Keys

- Certifies the creation of a TPM object.
- Ensure we are dealing with an object created inside a TPM.
- Signs the creation of the attributes
- Chains back to EK.

Credential Protection

- Proves possession of a machine resident ky
- Shared secret
  - Elliptic Curve Diffie-Hellman (ECDH)

Credential Protection

- `TPM2_CredentialActivate`
  - CredentialBlob
  - Secret
- `TPM2_MakeCredential`
  - Parent Handle
  - ObjectName
  - Digest
- Can be done without a TPM.

Prove possession of machine

Prove possession of machine

- MakeCredential on CA

Prove possession of machine

- MakeCredential on CA
- Send blob to the client

Prove possession of machine

- MakeCredential on CA
- Send blob to the client
- Client decrypt the blob

Prove possession of machine

- MakeCredential on CA
- Send blob to the client
- Client decrypt the blob
- Send back to the CA

Goals

- ✅ Device bound keys
- ✅ Identity claims
- ✅ Machine identity

What do we have...

- An ssh agent for TPM keys
- A way to prove possession of:
  - An identity
  - An machine

ssh-tpm-ca-authority

https://github.com/Foxboron/ssh-tpm-ca-authority

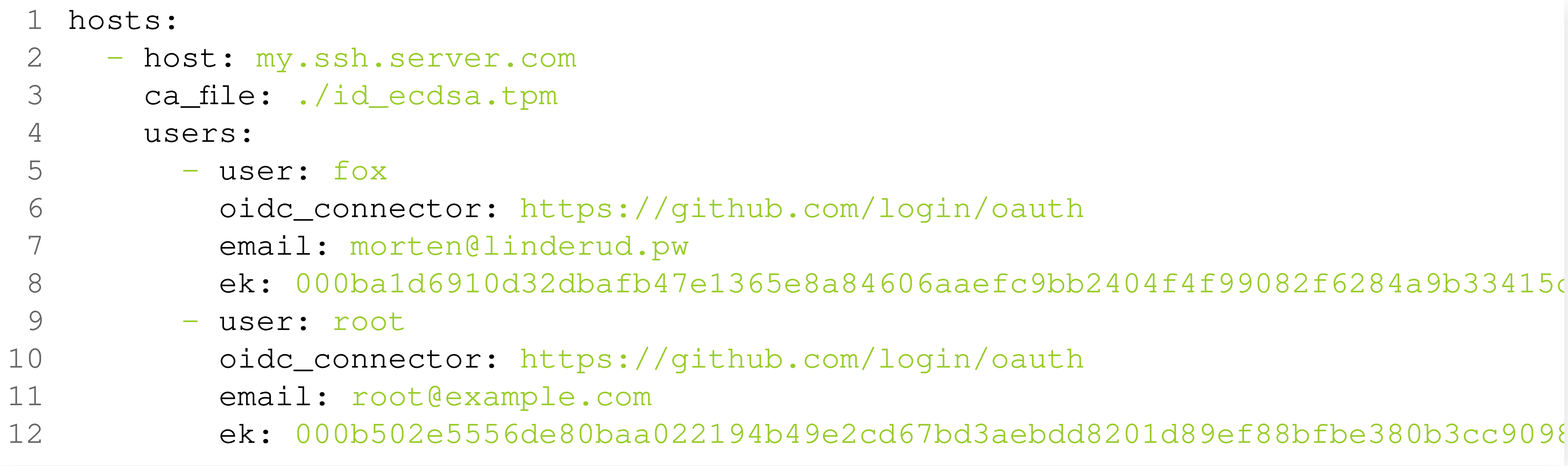
ssh-tpm-ca-authority

- Shorted lived certs
- User identities (OIDC)
- Machine identities - Credential Protection
- Uses standard openssh

Key creation

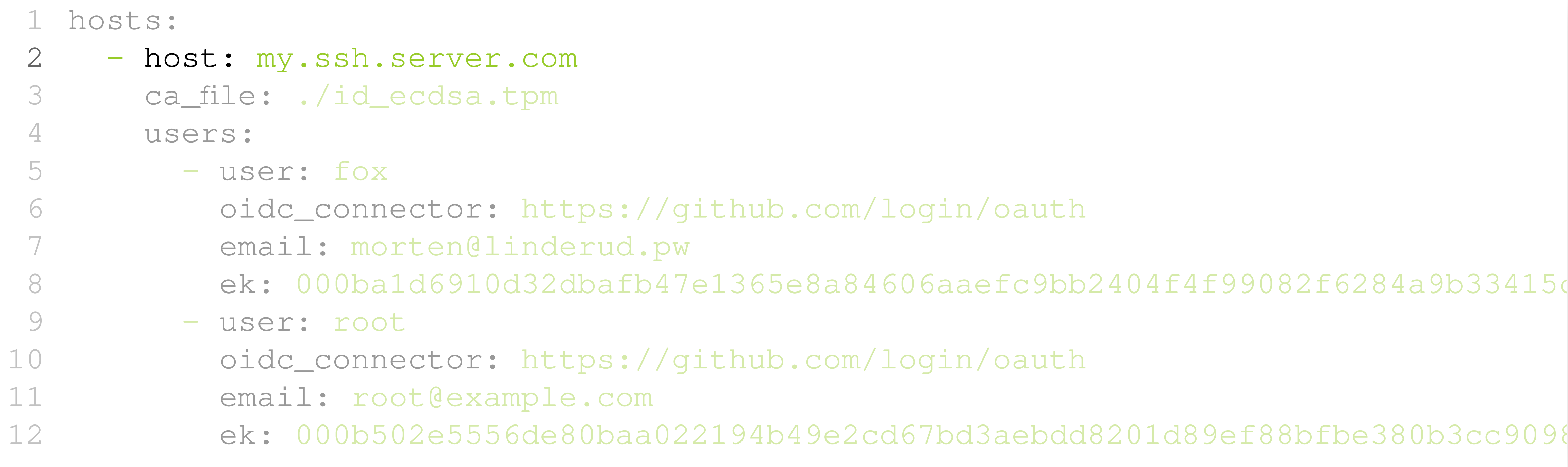
```
ssh-tpm-keygen -f ./id_ecdsa.tpm
```

Configuration



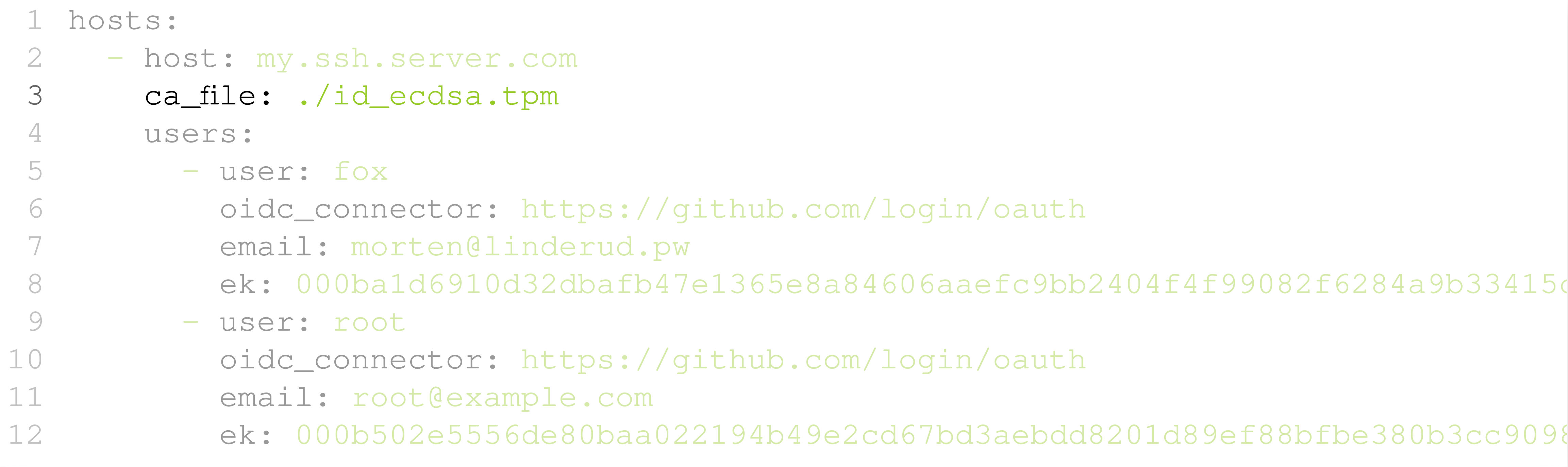
```yaml
hosts:
  - host: my.ssh.server.com
    ca_file: ./id_ecdsa.tpm
    users:
      - user: fox
        oidc_connector: https://github.com/login/oauth
        email: morten@linderud.pw
        ek: 000ba1d6910d32dbafb47e1365e8a84606aaefc9bb2404f4f99082f6284a9b33415c
      - user: root
        oidc_connector: https://github.com/login/oauth
        email: root@example.com
        ek: 000b502e5556de80baa022194b49e2cd67bd3aebdd8201d89ef88bfbe380b3cc9098
```

Configuration



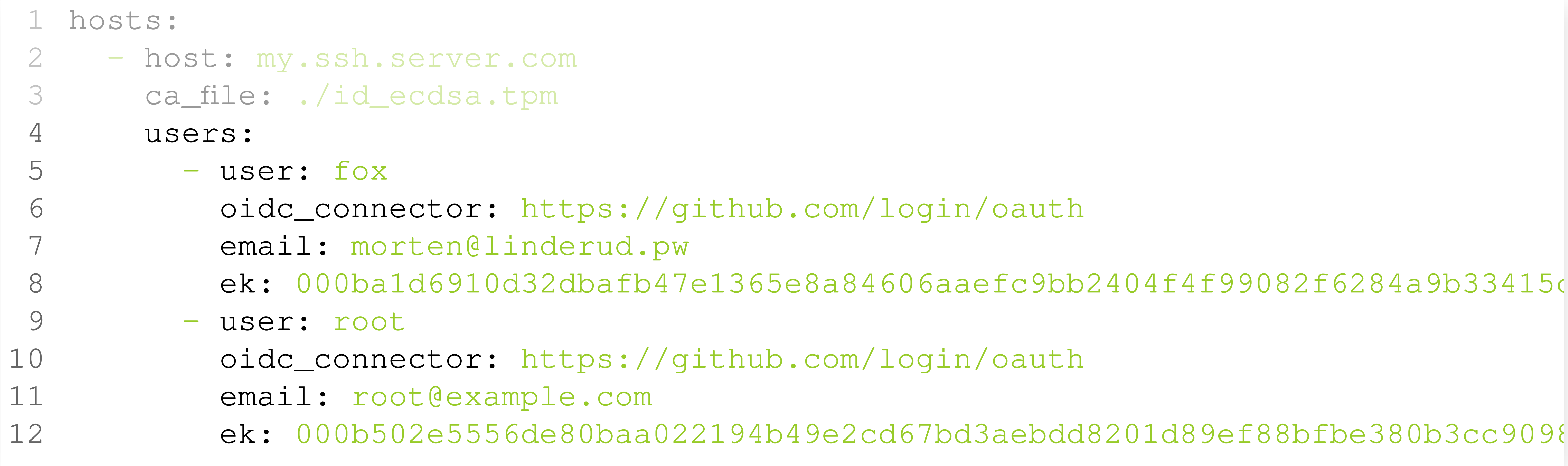
```yaml
hosts:
  - host: my.ssh.server.com
    ca_file: ./id_ecdsa.tpm
    users:
      - user: fox
        oidc_connector: https://github.com/login/oauth
        email: morten@linderud.pw
        ek: 000ba1d6910d32dbafb47e1365e8a84606aaefc9bb2404f4f99082f6284a9b33415
      - user: root
        oidc_connector: https://github.com/login/oauth
        email: root@example.com
        ek: 000b502e5556de80baa022194b49e2cd67bd3aebdd8201d89ef88bfbe380b3cc909
```

Configuration



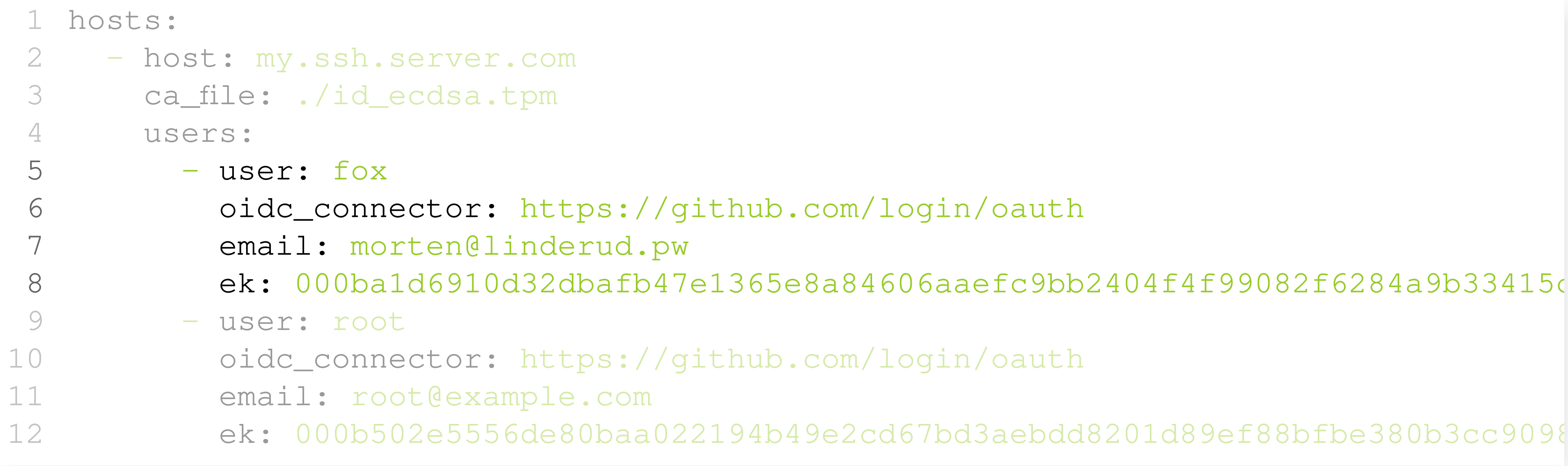
```yaml
hosts:
  - host: my.ssh.server.com
    ca_file: ./id_ecdsa.tpm
    users:
      - user: fox
        oidc_connector: https://github.com/login/oauth
        email: morten@linderud.pw
        ek: 000ba1d6910d32dbafb47e1365e8a84606aaefc9bb2404f4f99082f6284a9b33415
      - user: root
        oidc_connector: https://github.com/login/oauth
        email: root@example.com
        ek: 000b502e5556de80baa022194b49e2cd67bd3aebdd8201d89ef88bfbe380b3cc909
```

Configuration



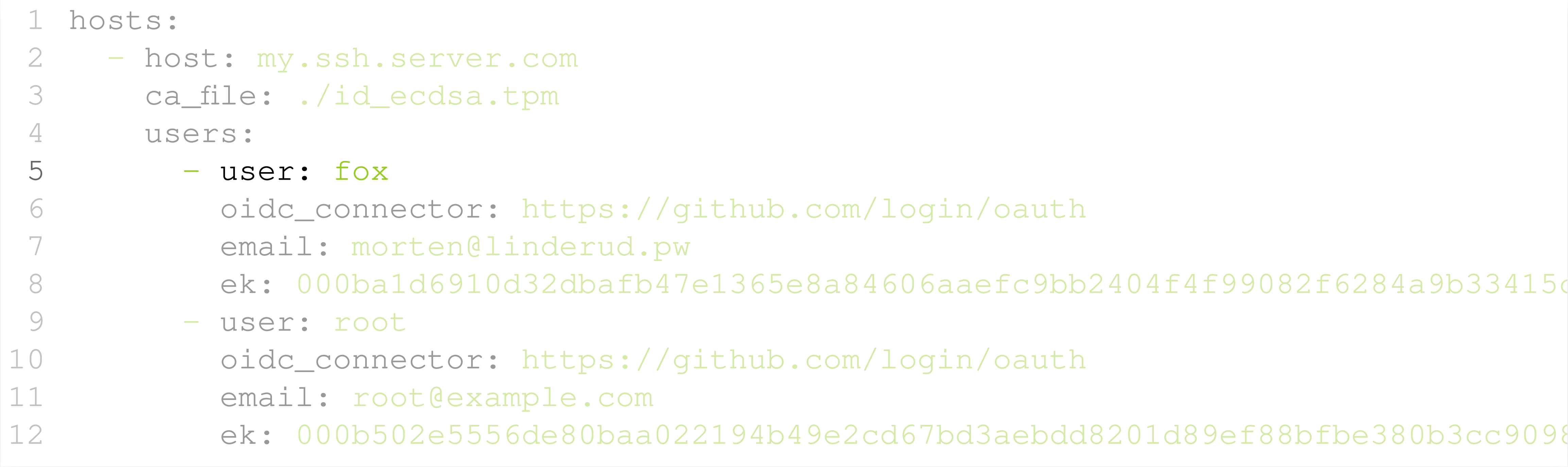
```yaml
hosts:
  - host: my.ssh.server.com
    ca_file: ./id_ecdsa.tpm
    users:
      - user: fox
        oidc_connector: https://github.com/login/oauth
        email: morten@linderud.pw
        ek: 000ba1d6910d32dbafb47e1365e8a84606aaefc9bb2404f4f99082f6284a9b33415c
      - user: root
        oidc_connector: https://github.com/login/oauth
        email: root@example.com
        ek: 000b502e5556de80baa022194b49e2cd67bd3aebdd8201d89ef88bfbe380b3cc9098
```

Configuration



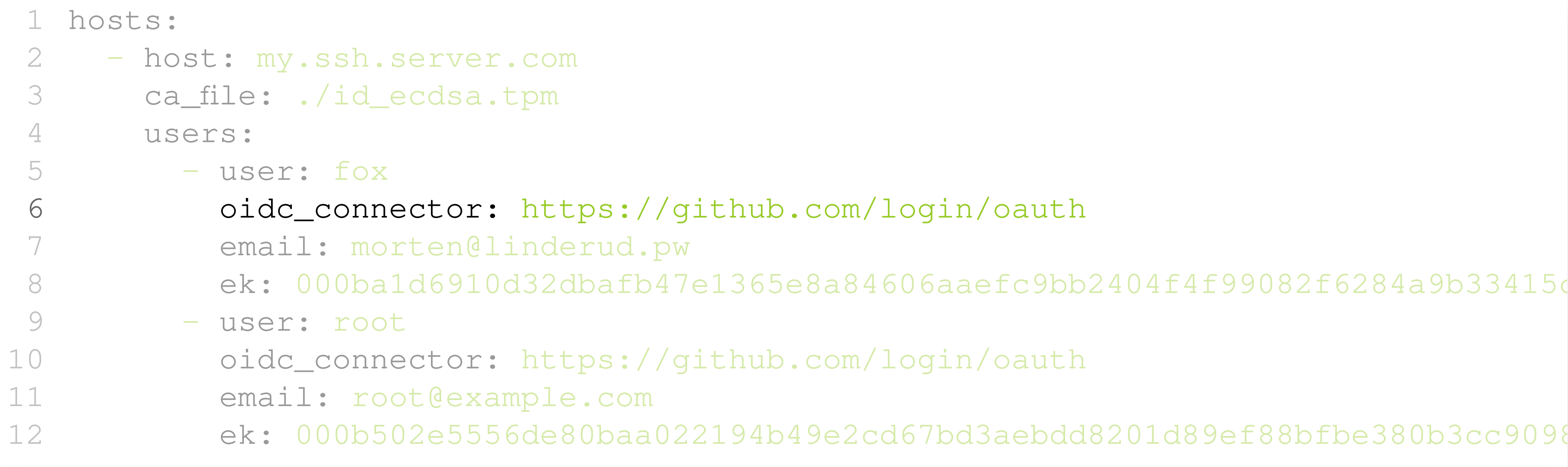
```yaml
hosts:
  - host: my.ssh.server.com
    ca_file: ./id_ecdsa.tpm
    users:
      - user: fox
        oidc_connector: https://github.com/login/oauth
        email: morten@linderud.pw
        ek: 000ba1d6910d32dbafb47e1365e8a84606aaefc9bb2404f4f99082f6284a9b33415
      - user: root
        oidc_connector: https://github.com/login/oauth
        email: root@example.com
        ek: 000b502e5556de80baa022194b49e2cd67bd3aebdd8201d89ef88bfbe380b3cc909
```

Configuration



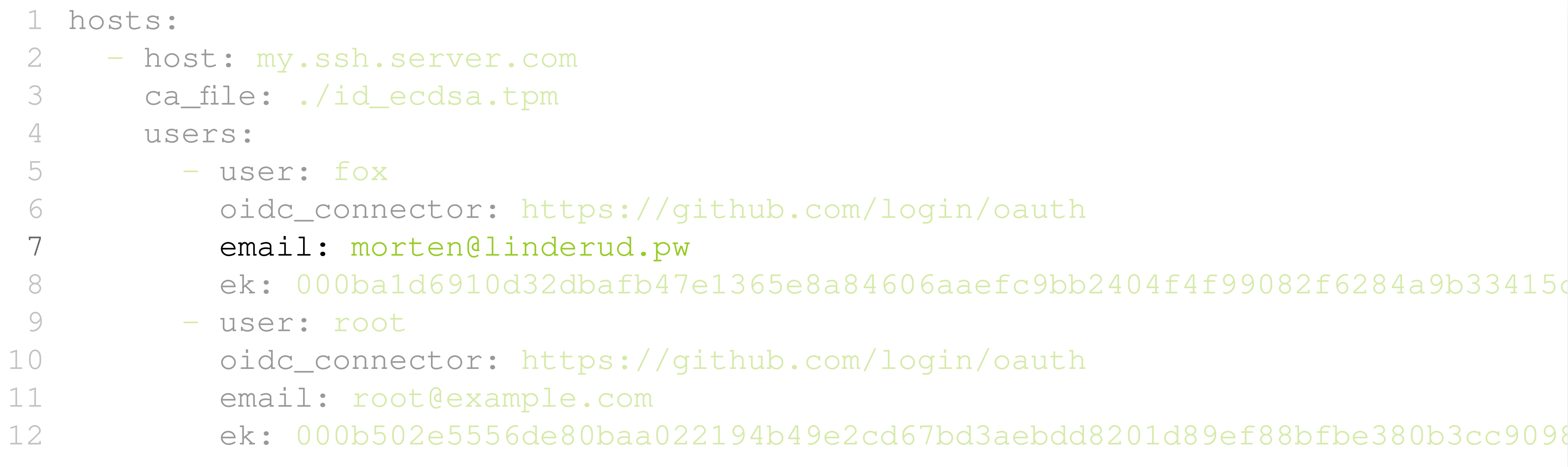
```yaml
hosts:
  - host: my.ssh.server.com
    ca_file: ./id_ecdsa.tpm
    users:
      - user: fox
        oidc_connector: https://github.com/login/oauth
        email: morten@linderud.pw
        ek: 000ba1d6910d32dbafb47e1365e8a84606aaefc9bb2404f4f99082f6284a9b33415
      - user: root
        oidc_connector: https://github.com/login/oauth
        email: root@example.com
        ek: 000b502e5556de80baa022194b49e2cd67bd3aebdd8201d89ef88bfbe380b3cc909
```

Configuration



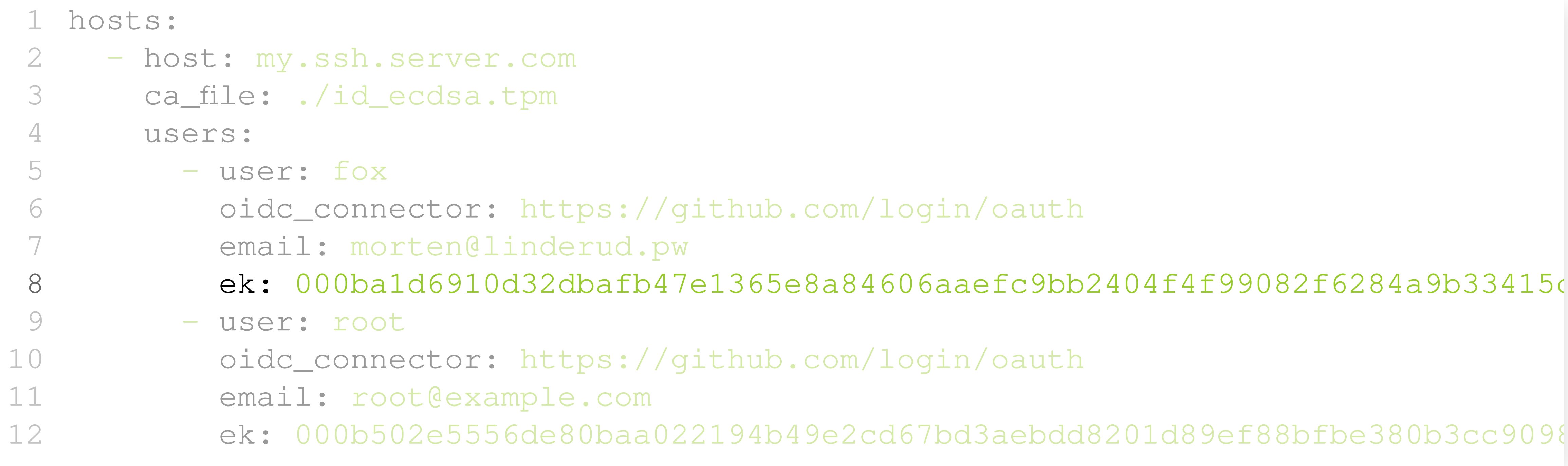
```yaml
hosts:
  - host: my.ssh.server.com
    ca_file: ./id_ecdsa.tpm
    users:
      - user: fox
        oidc_connector: https://github.com/login/oauth
        email: morten@linderud.pw
        ek: 000ba1d6910d32dbafb47e1365e8a84606aaefc9bb2404f4f99082f6284a9b33415
      - user: root
        oidc_connector: https://github.com/login/oauth
        email: root@example.com
        ek: 000b502e5556de80baa022194b49e2cd67bd3aebdd8201d89ef88bfbe380b3cc909
```

Configuration



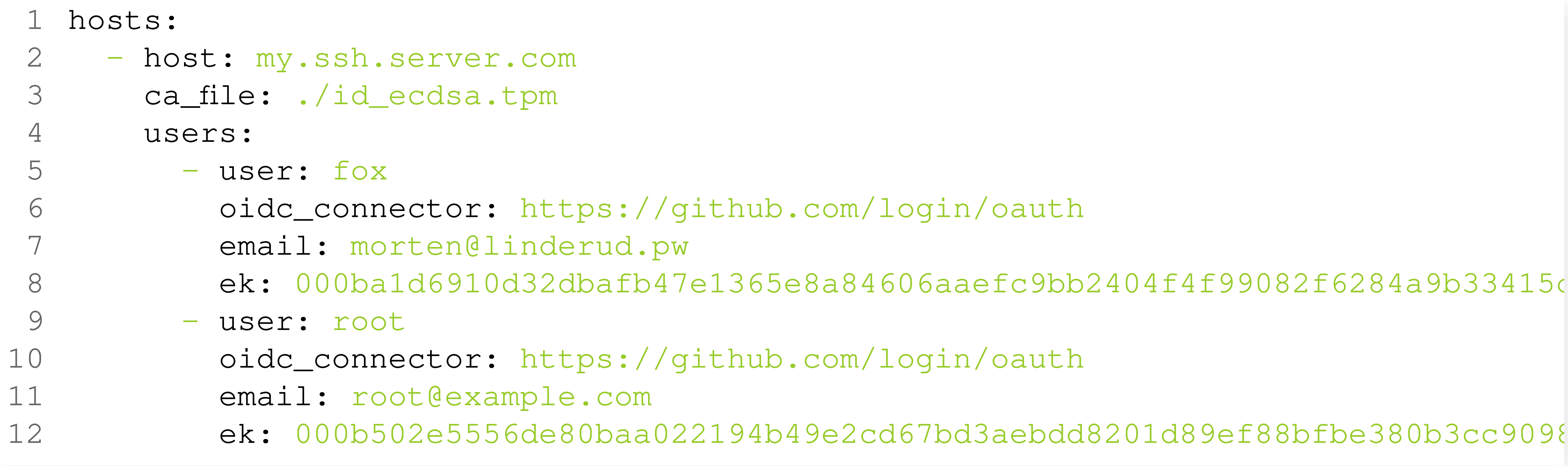
```yaml
hosts:
  - host: my.ssh.server.com
    ca_file: ./id_ecdsa.tpm
    users:
      - user: fox
        oidc_connector: https://github.com/login/oauth
        email: morten@linderud.pw
        ek: 000ba1d6910d32dbafb47e1365e8a84606aaefc9bb2404f4f99082f6284a9b33415
      - user: root
        oidc_connector: https://github.com/login/oauth
        email: root@example.com
        ek: 000b502e5556de80baa022194b49e2cd67bd3aebdd8201d89ef88bfbe380b3cc909
```

Configuration



```yaml
hosts:
  - host: my.ssh.server.com
    ca_file: ./id_ecdsa.tpm
    users:
      - user: fox
        oidc_connector: https://github.com/login/oauth
        email: morten@linderud.pw
        ek: 000ba1d6910d32dbafb47e1365e8a84606aaefc9bb2404f4f99082f6284a9b33415c
      - user: root
        oidc_connector: https://github.com/login/oauth
        email: root@example.com
        ek: 000b502e5556de80baa022194b49e2cd67bd3aebdd8201d89ef88bfbe380b3cc9098
```

Configuration

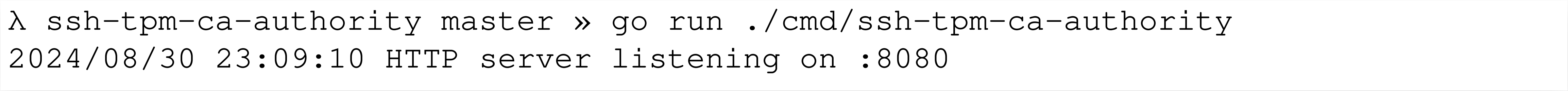


```
hosts:
  - host: my.ssh.server.com
    ca_file: ./id_ecdsa.tpm
    users:
      - user: fox
        oidc_connector: https://github.com/login/oauth
        email: morten@linderud.pw
        ek: 000ba1d6910d32dbafb47e1365e8a84606aaefc9bb2404f4f99082f6284a9b33415c
      - user: root
        oidc_connector: https://github.com/login/oauth
        email: root@example.com
        ek: 000b502e5556de80baa022194b49e2cd67bd3aebdd8201d89ef88bfbe380b3cc9098
```

Get EK certificate

```
λ ssh-tpm-ca-authority master » go run ./cmd/getek
000ba1d6910d32dbafb47e1365e8a84606aaefc9bb2404f4f99082f6284a9b33415c
```

Run the ssh ca server



```
λ ssh-tpm-ca-authority master » go run ./cmd/ssh-tpm-ca-authority
2024/08/30 23:09:10 HTTP server listening on :8080
```

Client Setup

```
Match host my.ssh.server.com  exec \
  "ssh-tpm-add --ca 'http://127.0.0.1:8080' --host '%h' --user '%r'"
```

Attestation Protocol

Attestation Protocol

SSH CA with device and identity attestation: ssh-tpm-ca-authority

https://linderud.dev/blog/ssh-ca-with-device-and-identity-attestation-ssh-tpm-ca-authority/

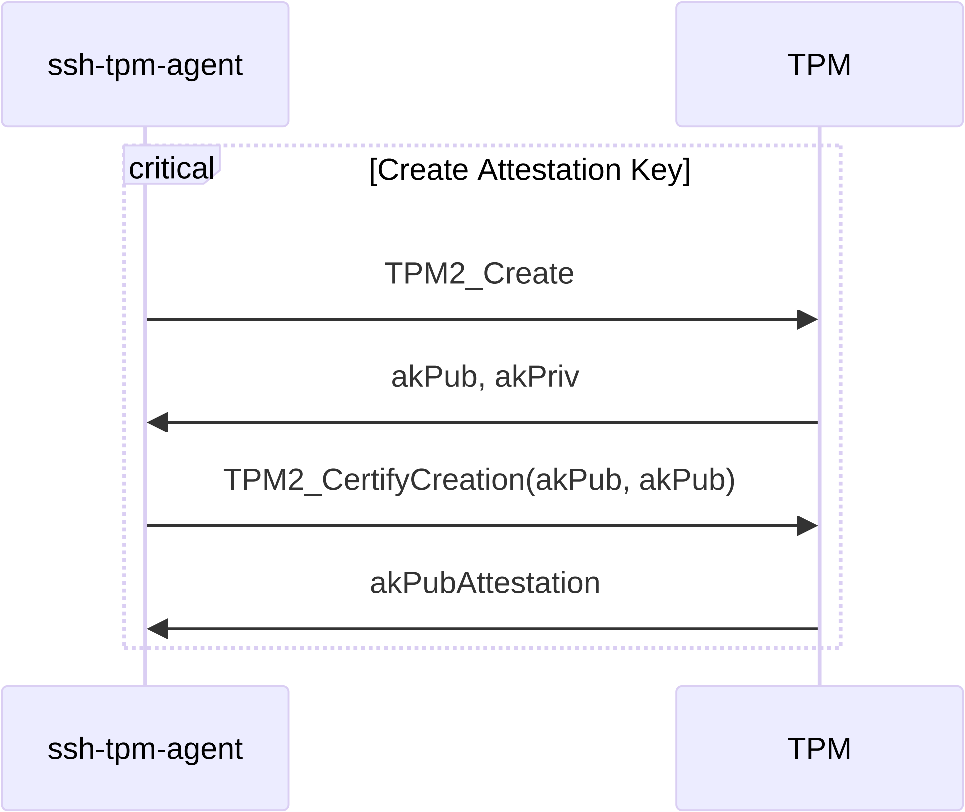
Attestation Protocol

- Server has two endpoints
- `/attest`
- `/submit`

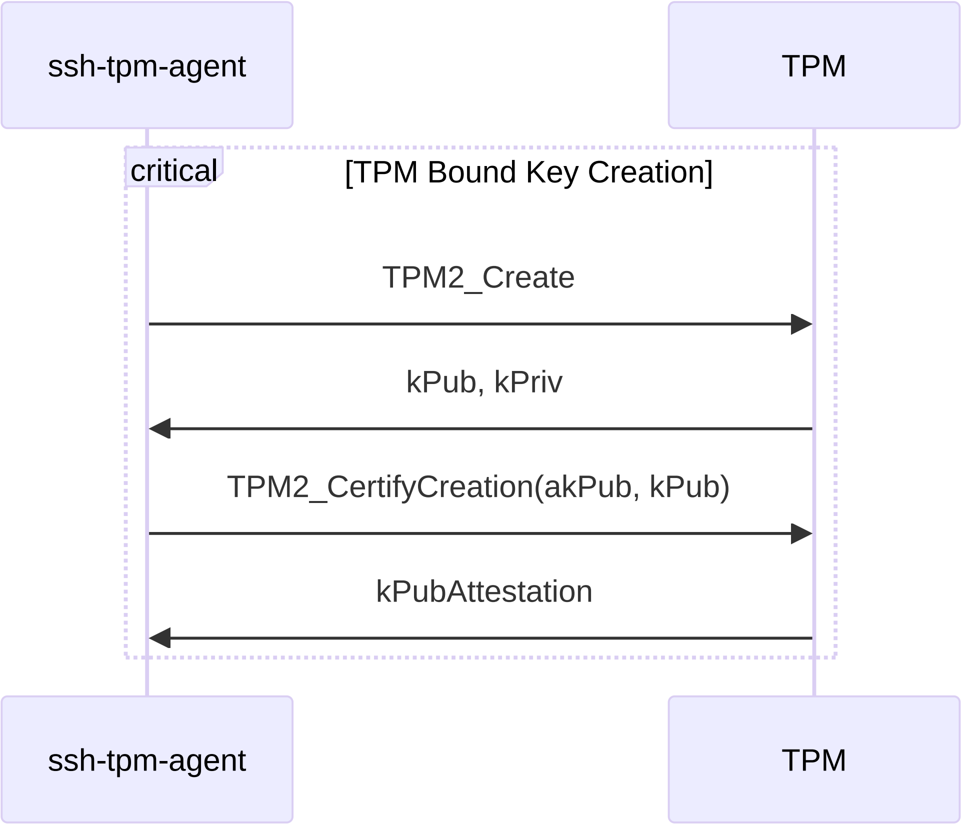
Attestation Protocol

- Key Creation
    - AK
    - TPM Bound Key
- Submit attestation
- OIDC Issuer Challenge
- Decrypt Credential
- Submit Challenge
- Signed ssh certificate

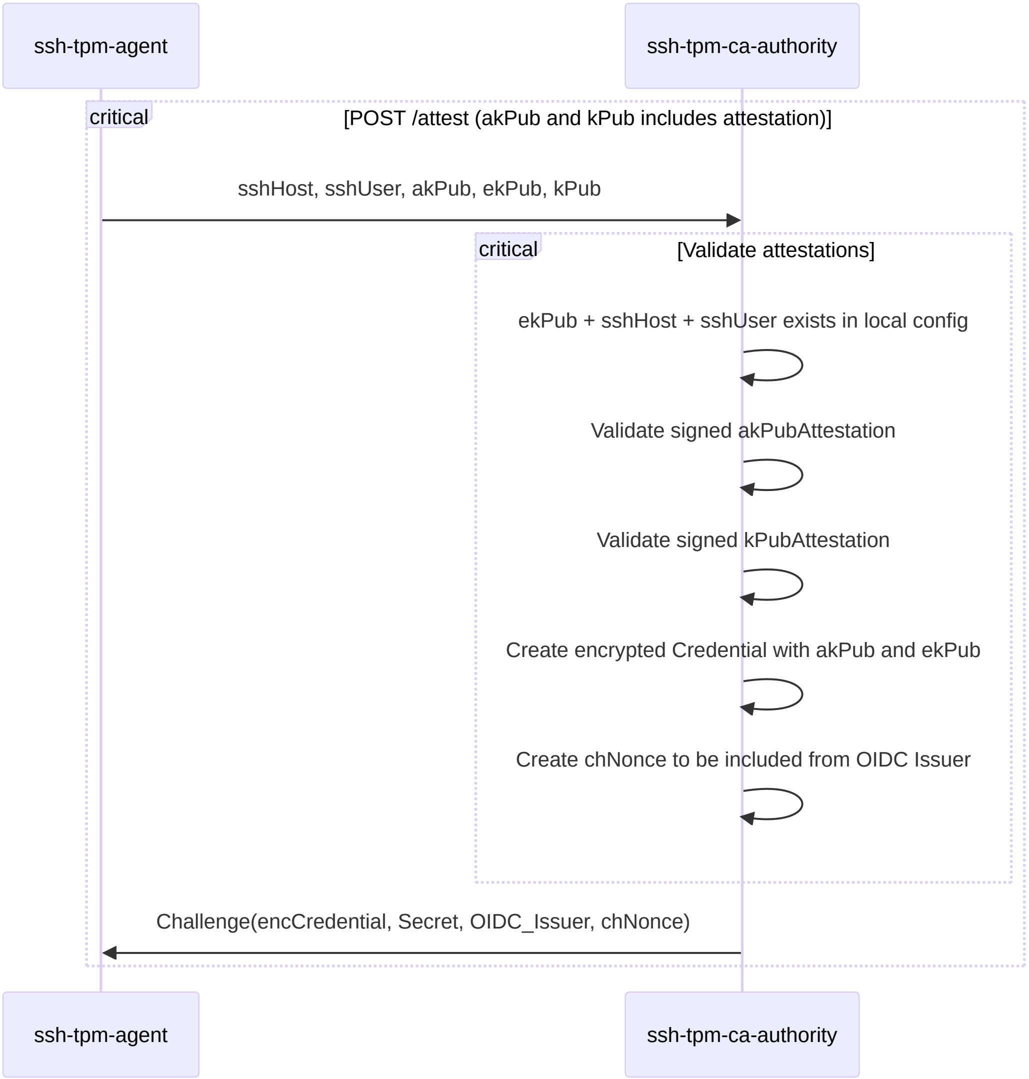
Key creation - Bound AK



Key creation - TPM Bound Key



Submit attestation



ssh-tpm-agent

ssh-tpm-ca-authority

critical [POST /attest (akPub and kPub includes attestation)]

sshHost, sshUser, akPub, ekPub, kPub

critical [Validate attestations]

ekPub + sshHost + sshUser exists in local config

Validate signed akPubAttestation

Validate signed kPubAttestation

Create encrypted Credential with akPub and ekPub

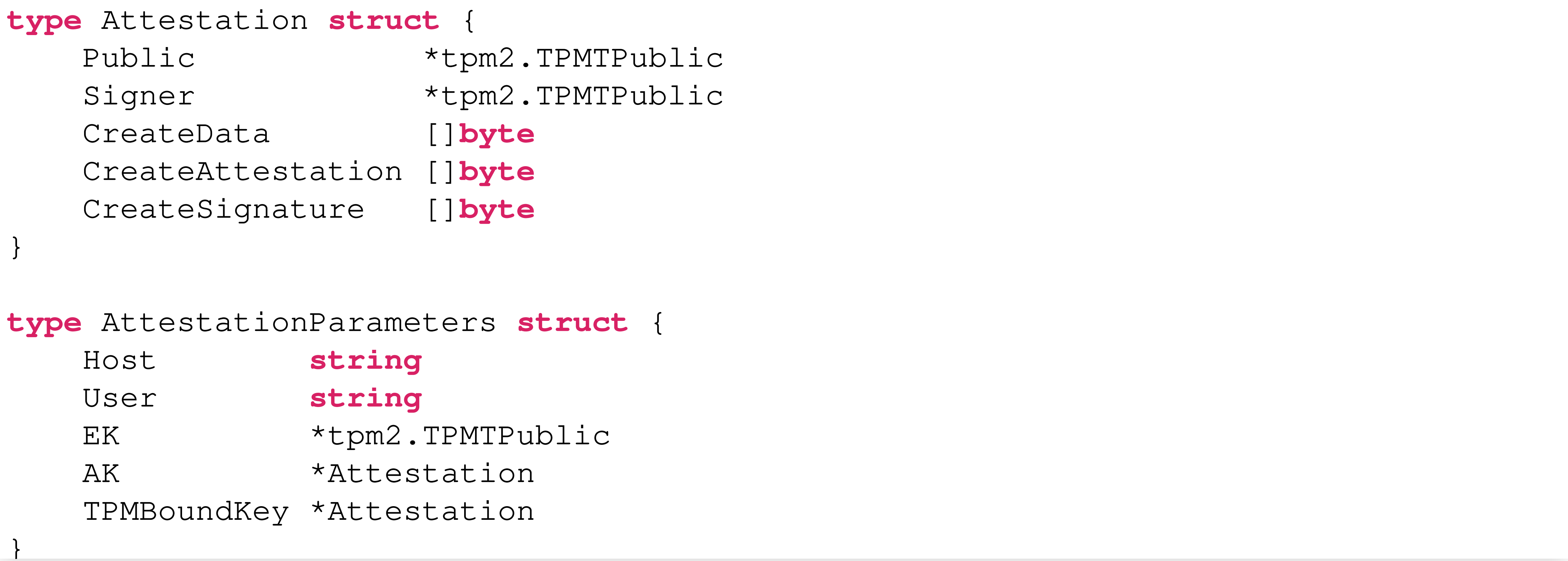
Create chNonce to be included from OIDC Issuer

Challenge(encCredential, Secret, OIDC_Issuer, chNonce)

ssh-tpm-agent

ssh-tpm-ca-authority

JSON Structs



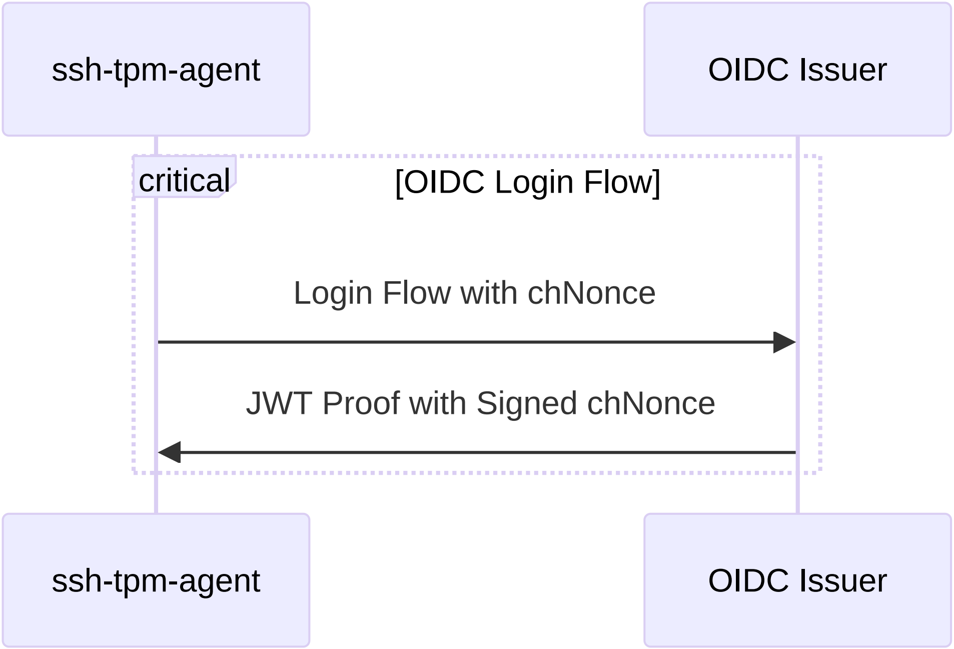
```
type Attestation struct {
    Public            *tpm2.TPMTPublic
    Signer            *tpm2.TPMTPublic
    CreateData        []byte
    CreateAttestation []byte
    CreateSignature   []byte
}

type AttestationParameters struct {
    Host        string
    User        string
    EK          *tpm2.TPMTPublic
    AK          *Attestation
    TPMBoundKey *Attestation
}
```

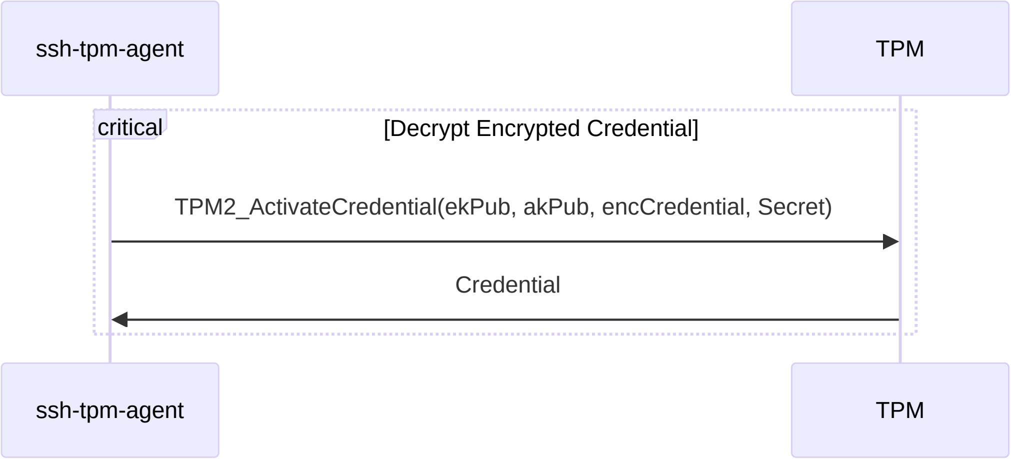
Challenge

```go
type EncryptedCredential struct {
    Credential []byte
    Secret     []byte
    OIDC       string
    Nonce      string
}
```

OIDC Issuer challenge



Decrypt credential



ssh-tpm-agent

TPM

critical

[Decrypt Encrypted Credential]

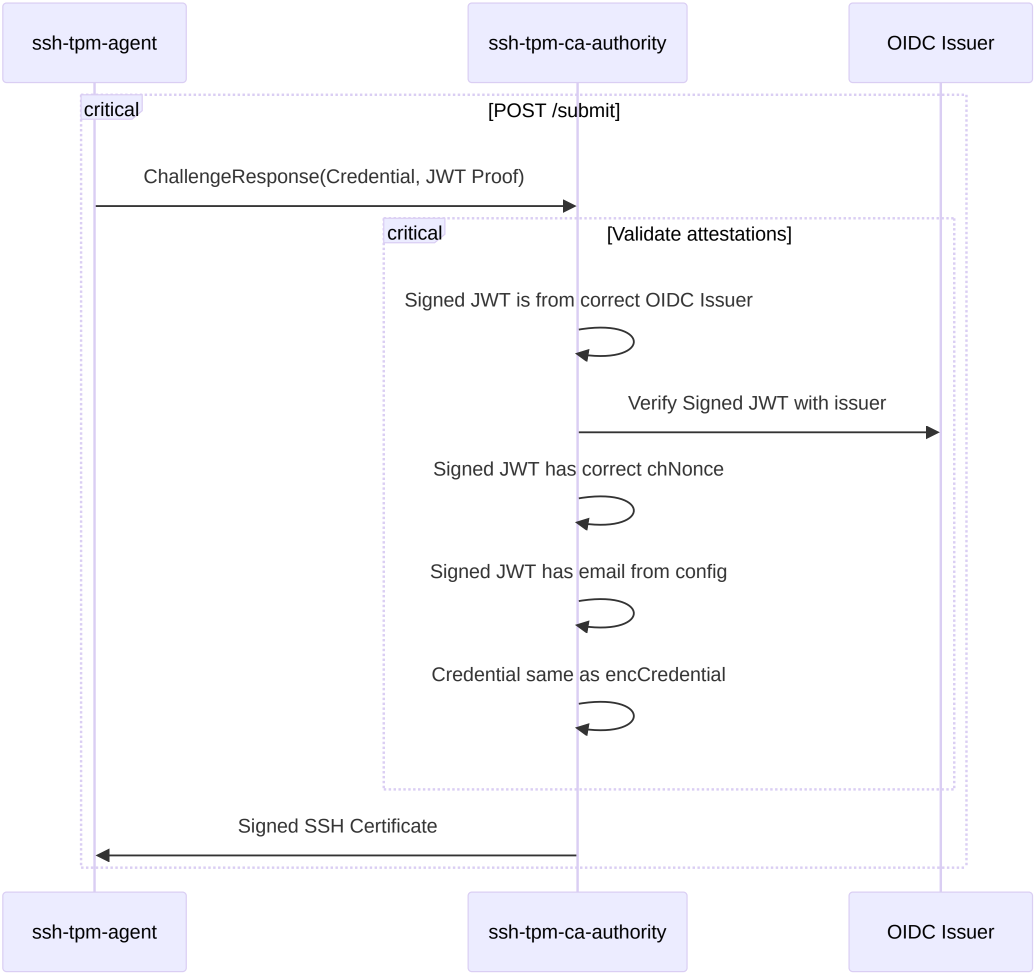
TPM2_ActivateCredential(ekPub, akPub, encCredential, Secret)

Credential

ssh-tpm-agent

TPM

Submit challenge


ssh-tpm-agent
ssh-tpm-ca-authority
OIDC Issuer
critical
[POST /submit]
ChallengeResponse(Credential, JWT Proof)
critical
[Validate attestations]
Signed JWT is from correct OIDC Issuer
Verify Signed JWT with issuer
Signed JWT has correct chNonce
Signed JWT has email from config
Credential same as encCredential
Signed SSH Certificate
ssh-tpm-agent
ssh-tpm-ca-authority
OIDC Issuer

JSON Struct

```
type ChallengeResponse struct {
    Secret []byte
    Jwt    string
}
```

Signed SSH Response

```go
type SignedCertResponse struct {
    SignedSSHCert []byte
}
```

ssh-agent

```
λ ~ » ssh-add -l
256 SHA256:+wa71L64hnLjYNOm4gintIfLc9GDUrwY22AIX40OliE  (ECDSA)
256 SHA256:+wa71L64hnLjYNOm4gintIfLc9GDUrwY22AIX40OliE  (ECDSA-CERT)
```

Improvements

- Is the protocol good enough?
- EK Certificates
    - Teleport does this

Improvements - openssh

- We should have a hostkey signed statement about the login?
- Agent should be aware of the ssh server?
- Specific agent for an extension?

Conclusion

Thanks!

- Github: Foxboron
- https://linderud.dev
- Email: morten@linderud.pw
- Mastodon: https://chaos.social/@Foxboron