SSH authentication using user and machine identities

Morten Linderud

$ whoami

- Morten Linderud
  - Foxboron
- F/OSS developer since ~2013
- Arch Linux Developer
- Reproducible Builds
- Usable security tools, supply chain security, Linux distro security
- `sbctl` and `go-uefi`

Combining machine identity with user identity claims

TPM Primer

TPM Primer

- Key shielding
- Hierarchies
  - Null
  - Storage
  - Endorsement
- Attestation

TPM Primer

- Slow devices
- API doesn't make sense
- Needs tooling
- Limited cryptography

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

```
$ 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.
```

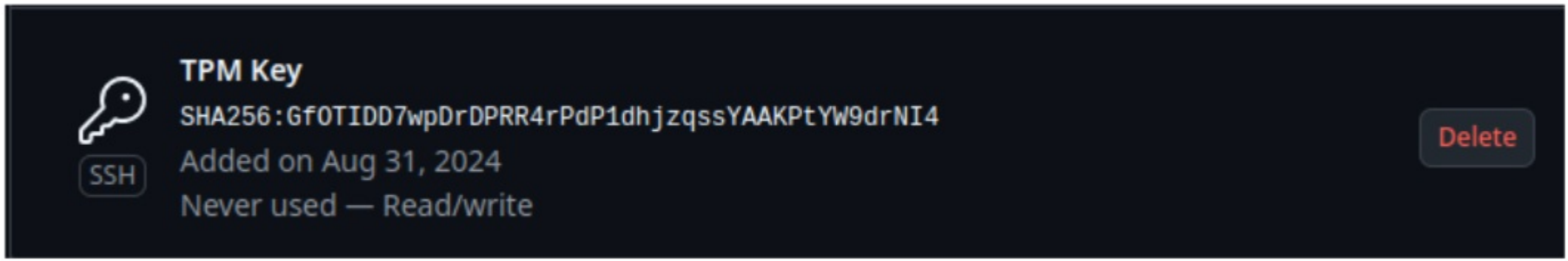
```
$ ssh-tpm-add /home/fox/.ssh/id_ecdsa.tpm
Identity added: id_ecdsa.tpm

$ 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.
```


TPM Key
SHA256:GfOTIDD7wpDrDPRR4rPdP1dhjzqssYAAKPtYW9drNI4
SSH
Added on Aug 31, 2024
Never used — Read/write
Delete

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

```
 1 func main() {
 2     tpm, _ := transport.OpenTPM()
 3     defer tpm.Close()
 4
 5     k, _ := keyfile.NewLoadableKey(tpm, tpm2.TPMAlgECC, 256, []byte{},
 6         keyfile.WithDescription("TPM Key"),
 7     )
 8
 9     os.Writefile("key.pem", k.Bytes(), 0640)
10
11     signer, _ := k.Signer(tpm, []byte(""), []byte(""))
12     sig, _ := signer.Sign((io.Reader)(nil), []byte{....}, crypto.SHA256)
13 }
```

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)
}
```

Go API

```
 1 func main() {
 2     tpm, _ := transport.OpenTPM()
 3     defer tpm.Close()
 4
 5     k, _ := keyfile.NewLoadableKey(tpm, tpm2.TPMAlgECC, 256, []byte{},
 6         keyfile.WithDescription("TPM Key"),
 7     )
 8
 9     os.Writefile("key.pem", k.Bytes(), 0640)
10
11     signer, _ := k.Signer(tpm, []byte(""), []byte(""))
12     sig, _ := signer.Sign((io.Reader)(nil), []byte{....}, crypto.SHA256)
13 }
```

NewTPMKey

```
1  func main(){
2      tpm, _ := transport.OpenTPM()
3      defer tpm.Close()
4
5      eccTemplate := tpm2.TPMTPublic{ ... }
6
7      eccKeyResponse, := tpm2.CreateLoaded{
8          ParentHandle: tpm2.AuthHandle{
9              Handle: primary.ObjectHandle,
10             Name:   primary.Name,
11             Auth:   tpm2.PasswordAuth([]byte(nil)),
12         },
13         InPublic: tpm2.New2BTemplate(&eccTemplate),
14     }.Execute(tpm)
15
```

NewTPMKey

```
1  func main(){
2      tpm, _ := transport.OpenTPM()
3      defer tpm.Close()
4
5      eccTemplate := tpm2.TPMTPublic{ ... }
6
7      eccKeyResponse, := tpm2.CreateLoaded{
8          ParentHandle: tpm2.AuthHandle{
9              Handle: primary.ObjectHandle,
10             Name:   primary.Name,
11             Auth:   tpm2.PasswordAuth([]byte(nil)),
12         },
13         InPublic: tpm2.New2BTemplate(&eccTemplate),
14     }.Execute(tpm)
15
```

NewTPMKey

```
 4
 5      eccTemplate := tpm2.TPMTPublic{ ... }
 6
 7      eccKeyResponse, := tpm2.CreateLoaded{
 8          ParentHandle: tpm2.AuthHandle{
 9              Handle: primary.ObjectHandle,
10              Name:   primary.Name,
11              Auth:   tpm2.PasswordAuth([]byte(nil)),
12          },
13          InPublic: tpm2.New2BTemplate(&eccTemplate),
14      }.Execute(tpm)
15
16      k := keyfile.NewTPMKey(
17          keyfile.OIDOldLoadableKey
```

NewTPMKey

```
10              Name:   primary.Name,
11              Auth:   tpm2.PasswordAuth([]byte(nil)),
12          },
13          InPublic: tpm2.New2BTemplate(&eccTemplate),
14      }.Execute(tpm)
15
16      k := keyfile.NewTPMKey(
17          keyfile.OIDOldLoadableKey
18          eccKeyResponse.OutPublic,
19          eccKeyResponse.OutPrivate,
20          keyfile.WithDescription("This is a TPM Key"),
21      )
22
23      os.Writefile("key.pem", k.Bytes(), 0640)
24  }
```

NewTPMKey

```
10              Name:   primary.Name,
11              Auth:   tpm2.PasswordAuth([]byte(nil)),
12          },
13          InPublic: tpm2.New2BTemplate(&eccTemplate),
14      }.Execute(tpm)
15
16      k := keyfile.NewTPMKey(
17          keyfile.OIDOldLoadableKey
18          eccKeyResponse.OutPublic,
19          eccKeyResponse.OutPrivate,
20          keyfile.WithDescription("This is a TPM Key"),
21      )
22
23      os.Writefile("key.pem", k.Bytes(), 0640)
24  }
```

NewTPMKey

```
1  func main(){
2      tpm, _ := transport.OpenTPM()
3      defer tpm.Close()
4
5      eccTemplate := tpm2.TPMTPublic{ ... }
6
7      eccKeyResponse, := tpm2.CreateLoaded{
8          ParentHandle: tpm2.AuthHandle{
9              Handle: primary.ObjectHandle,
10             Name:   primary.Name,
11             Auth:   tpm2.PasswordAuth([]byte(nil)),
12         },
13         InPublic: tpm2.New2BTemplate(&eccTemplate),
14     }.Execute(tpm)
15
```

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=
```

Importing keys with wrapping

- ImportableKeys OID
- Allows to pass around encrypted keys
- Only the machin can decrypt

Create parent on client

```
tpm2_createprimary -C o -G ecc -g sha256 -c prim.ctx \
  -a 'restricted|decrypt|fixedtpm|fixedparent|
      sensitivedataorigin|userwithauth|noda' \
  -f pem -o srk.pem
```

Create key on remote

```
ssh-keygen -t ecdsa -b 256 -N "" -f ./ecdsa.key
# or
openssl genpkey -algorithm EC \
    -pkeyopt ec_paramgen_curve:prime256v1 \
    -out ecdsa.key
```

Wrap key with parent

```
1 ssh-tpm-keygen --wrap-with srk.pub --wrap ecdsa.key -f wrapped_id_ecdsa
```

Import wrapped key

```
1 ssh-tpm-keygen --import ./wrapped_id_ecdsa.tpm --output id_ecdsa.tpm
2 ssh-tpm-add id_ecdsa.tpm
```

Identities and Device attestation

Machine Identity with TPM

- 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
- Chain back to EK.

Credential Protection

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

Credential Protection

```
// pubkey -> Public EK Cert
// ephKey -> Ephemeral Key for ECDH

seed := KDFe(ekNameAlg, z, "IDENTITY", ephKey.X, pubKey.X, bits)

symKey := KDFa(ekNameAlg, seed, "STORAGE" Name, Null, bits)

encIdentity := CFB^ekSymAlg(symKey, 0, CV)

HMACKey := KDFa(ekNameAlg, seed "INTEGRITY", NULL, NULL, bits)

outerHMAC := HMAC^ekNameAlg(HMACKey, encIdentity || Name)

blob := outerHMAC || encIdentity

```

Credential Protection

```
// pubkey -> Public EK Cert
// ephKey -> Ephemeral Key for ECDH

seed := KDFe(ekNameAlg, z, "IDENTITY", ephKey.X, pubKey.X, bits)

symKey := KDFa(ekNameAlg, seed, "STORAGE" Name, Null, bits)

encIdentity := CFB^ekSymAlg(symKey, 0, CV)

HMACKey := KDFa(ekNameAlg, seed "INTEGRITY", NULL, NULL, bits)

outerHMAC := HMAC^ekNameAlg(HMACKey, encIdentity || Name)

blob := outerHMAC || encIdentity

```

Credential Protection

```
// pubkey -> Public EK Cert
// ephKey -> Ephemeral Key for ECDH

seed := KDFe(ekNameAlg, z, "IDENTITY", ephKey.X, pubKey.X, bits)

symKey := KDFa(ekNameAlg, seed, "STORAGE" Name, Null, bits)

encIdentity := CFB^ekSymAlg(symKey, 0, CV)

HMACKey := KDFa(ekNameAlg, seed "INTEGRITY", NULL, NULL, bits)

outerHMAC := HMAC^ekNameAlg(HMACKey, encIdentity || Name)

blob := outerHMAC || encIdentity

```

Credential Protection

```
1  // pubkey -> Public EK Cert
2  // ephKey -> Ephemeral Key for ECDH
3
4  seed := KDFe(ekNameAlg, z, "IDENTITY", ephKey.X, pubKey.X, bits)
5
6  symKey := KDFa(ekNameAlg, seed, "STORAGE" Name, Null, bits)
7
8  encIdentity := CFB^ekSymAlg(symKey, 0, CV)
9
10 HMACKey := KDFa(ekNameAlg, seed "INTEGRITY", NULL, NULL, bits)
11
12 outerHMAC := HMAC^ekNameAlg(HMACKey, encIdentity || Name)
13
14 blob := outerHMAC || encIdentity
15
```

Credential Protection

```
1  // pubkey -> Public EK Cert
2  // ephKey -> Ephemeral Key for ECDH
3
4  seed := KDFe(ekNameAlg, z, "IDENTITY", ephKey.X, pubKey.X, bits)
5
6  symKey := KDFa(ekNameAlg, seed, "STORAGE" Name, Null, bits)
7
8  encIdentity := CFB^ekSymAlg(symKey, 0, CV)
9
10 HMACKey := KDFa(ekNameAlg, seed "INTEGRITY", NULL, NULL, bits)
11
12 outerHMAC := HMAC^ekNameAlg(HMACKey, encIdentity || Name)
13
14 blob := outerHMAC || encIdentity
15
```

Credential Protection

```
 2 // ephKey <- Ephemeral Key for ECDH
 3
 4 seed := KDFe(ekNameAlg, z, "IDENTITY", ephKey.X, pubKey.X, bits)
 5
 6 symKey := KDFa(ekNameAlg, seed, "STORAGE" Name, Null, bits)
 7
 8 encIdentity := CFB^ekSymAlg(symKey, 0, CV)
 9
10 HMACKey := KDFa(ekNameAlg, seed "INTEGRITY", NULL, NULL, bits)
11
12 outerHMAC := HMAC^ekNameAlg(HMACKey, encIdentity || Name)
13
14 blob := outerHMAC || encIdentity
15
16 // Send back public part of ephKey, and blob to TPM
```

Credential Protection

```
2  // ephKey <- Ephemeral Key for ECDH
3
4  seed := KDFe(ekNameAlg, z, "IDENTITY", ephKey.X, pubKey.X, bits)
5
6  symKey := KDFa(ekNameAlg, seed, "STORAGE" Name, Null, bits)
7
8  encIdentity := CFB^ekSymAlg(symKey, 0, CV)
9
10 HMACKey := KDFa(ekNameAlg, seed "INTEGRITY", NULL, NULL, bits)
11
12 outerHMAC := HMAC^ekNameAlg(HMACKey, encIdentity || Name)
13
14 blob := outerHMAC || encIdentity
15
16 // Send back public part of ephKey, and blob to TPM
```

Credential Protection

```
2 // ephKey <- EphemeralKey for ECDH
3
4 seed := KDFe(ekNameAlg, z, "IDENTITY", ephKey.X, pubKey.X, bits)
5
6 symKey := KDFa(ekNameAlg, seed, "STORAGE" Name, Null, bits)
7
8 encIdentity := CFB^ekSymAlg(symKey, 0, CV)
9
10 HMACKey := KDFa(ekNameAlg, seed "INTEGRITY", NULL, NULL, bits)
11
12 outerHMAC := HMAC^ekNameAlg(HMACKey, encIdentity || Name)
13
14 blob := outerHMAC || encIdentity
15
16 // Send back public part of ephKey, and blob to TPM
```

Credential Protection

```
 2 // ephKey <- EphemeralKey for ECDH
 3
 4 seed := KDFe(ekNameAlg, z, "IDENTITY", ephKey.X, pubKey.X, bits)
 5
 6 symKey := KDFa(ekNameAlg, seed, "STORAGE" Name, Null, bits)
 7
 8 encIdentity := CFB^ekSymAlg(symKey, 0, CV)
 9
10 HMACKey := KDFa(ekNameAlg, seed "INTEGRITY", NULL, NULL, bits)
11
12 outerHMAC := HMAC^ekNameAlg(HMACKey, encIdentity || Name)
13
14 blob := outerHMAC || encIdentity
15
16 // Send back public part of ephKey, and blob to TPM
```

Credential Protection

```
// pubkey -> Public EK Cert
// ephKey -> Ephemeral Key for ECDH

seed := KDFe(ekNameAlg, z, "IDENTITY", ephKey.X, pubKey.X, bits)

symKey := KDFa(ekNameAlg, seed, "STORAGE" Name, Null, bits)

encIdentity := CFB^ekSymAlg(symKey, 0, CV)

HMACKey := KDFa(ekNameAlg, seed "INTEGRITY", NULL, NULL, bits)

outerHMAC := HMAC^ekNameAlg(HMACKey, encIdentity || Name)

blob := outerHMAC || encIdentity

```

Open ID Connect

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


```
{
  "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 Example

```
12     "at_hash": "NkMjXQVRUX48RX8hRQDNfQ",
13     "c_hash": "bJchbzUCdmu8OSMTHNs5ZQ",
14     "email": "morten@linderud.pw",
15     "email_verified": true,
16     "federated_claims": {
17       "connector_id": "https://github.com/login/oauth",
18       "user_id": "1042946"
19     }
20   }
21   RSASHA256(
22     base64UrlEncode(header) + "." +
23     base64UrlEncode(payload),
24     [privkey]
25     [pubkey],
26   )
```

Identity Provider (IdP)

Identity Provider

- Microsoft, Google, Keycloak
- 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

```
46      "cloud_instance_host_name",
47      "cloud_graph_host_name",
48      "msgraph_host",
49      "aud",
50      "exp",
51      "iat",
52      "auth_time",
53      "acr",
54      "nonce",
55      "preferred_username",
56      "name",
57      "tid",
58      "ver",
59      "at_hash",
60      "c_hash",
```

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(!)

SSH certificates

ssh certificates

- Principals - users
- Capabilities
- Time limit/lifetime

SSH Cert - 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 Cert - 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 Cert - 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 Cert - 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 Cert - 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 Cert - Go example

```
 3
 4 sshkey, _ := ssh.NewPublicKey(...)
 5
 6 certificate := ssh.Certificate{
 7     Key:             sshkey,
 8     CertType:        ssh.UserCert,
 9     ValidPrincipals: []string{"Foxboron"},
10     KeyId:           "TPM Key",
11     ValidAfter:      uint64(after.Unix()),
12     ValidBefore:     uint64(before.Unix()),
13     Permissions: ssh.Permissions{
14         Extensions: map[string]string{
15             "permit-agent-forwarding": "",
16             "permit-port-forwarding":  "",
17             "permit-pty":              "",
```

SSH Cert - Go example

```
 5
 6 certificate := ssh.Certificate{
 7     Key:             sshkey,
 8     CertType:        ssh.UserCert,
 9     ValidPrincipals: []string{"Foxboron"},
10     KeyId:           "TPM Key",
11     ValidAfter:      uint64(after.Unix()),
12     ValidBefore:     uint64(before.Unix()),
13     Permissions: ssh.Permissions{
14         Extensions: map[string]string{
15             "permit-agent-forwarding": "",
16             "permit-port-forwarding":  "",
17             "permit-pty":              "",
18             "permit-user-rc":          "",
```

SSH Cert - Go example

```
 7     Key:             sshkey,
 8     CertType:        ssh.UserCert,
 9     ValidPrincipals: []string{"Foxboron"},
10     KeyId:           "TPM Key",
11     ValidAfter:      uint64(after.Unix()),
12     ValidBefore:     uint64(before.Unix()),
13     Permissions: ssh.Permissions{
14         Extensions: map[string]string{
15             "permit-agent-forwarding": "",
16             "permit-port-forwarding":  "",
17             "permit-pty":              "",
18             "permit-user-rc":          "",
19         },
20     },
21 }
```

SSH Cert - 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": "",
```

Combine the Open ID Connect claims with TPM Attestation

```
{
  "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!

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

```
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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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/

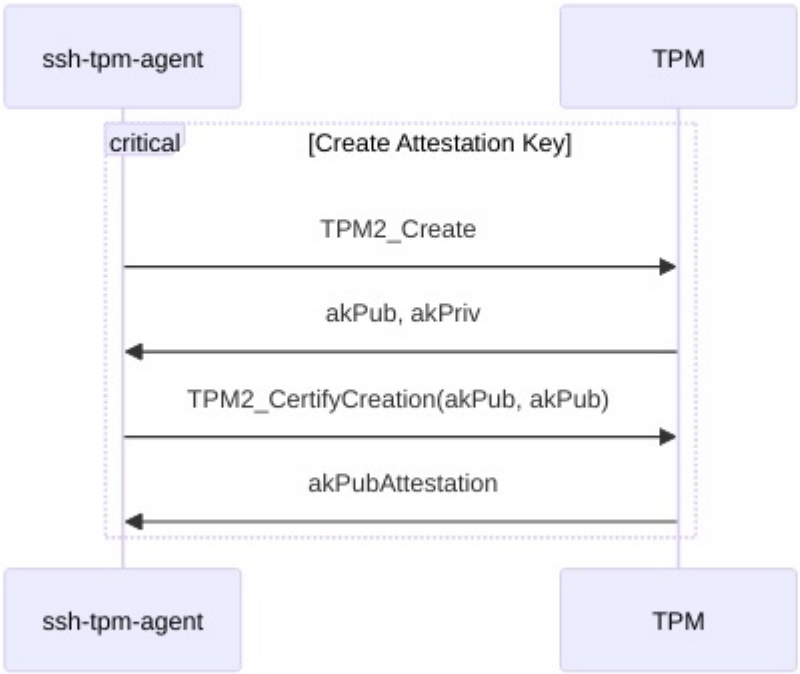
Attestation Protocol

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

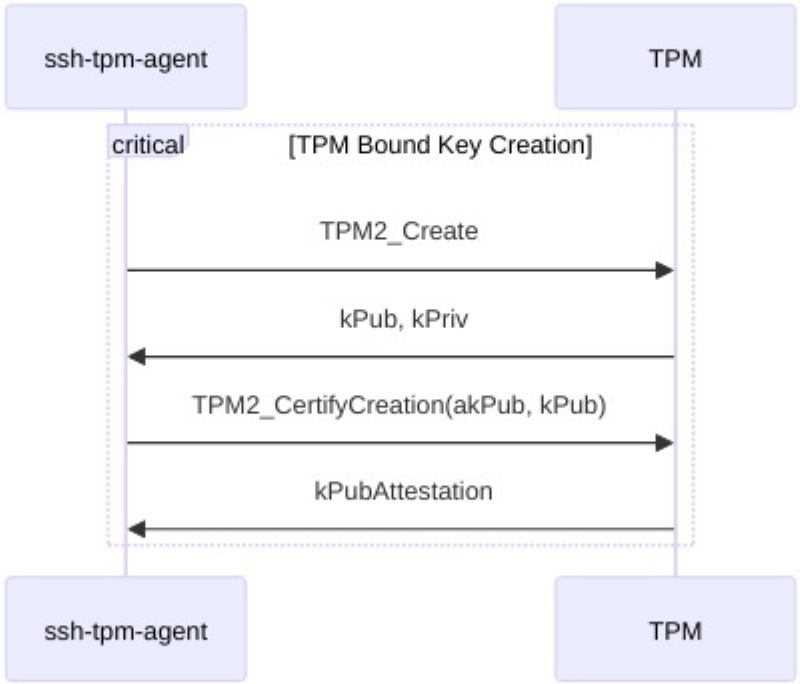
Attestation Protocol

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

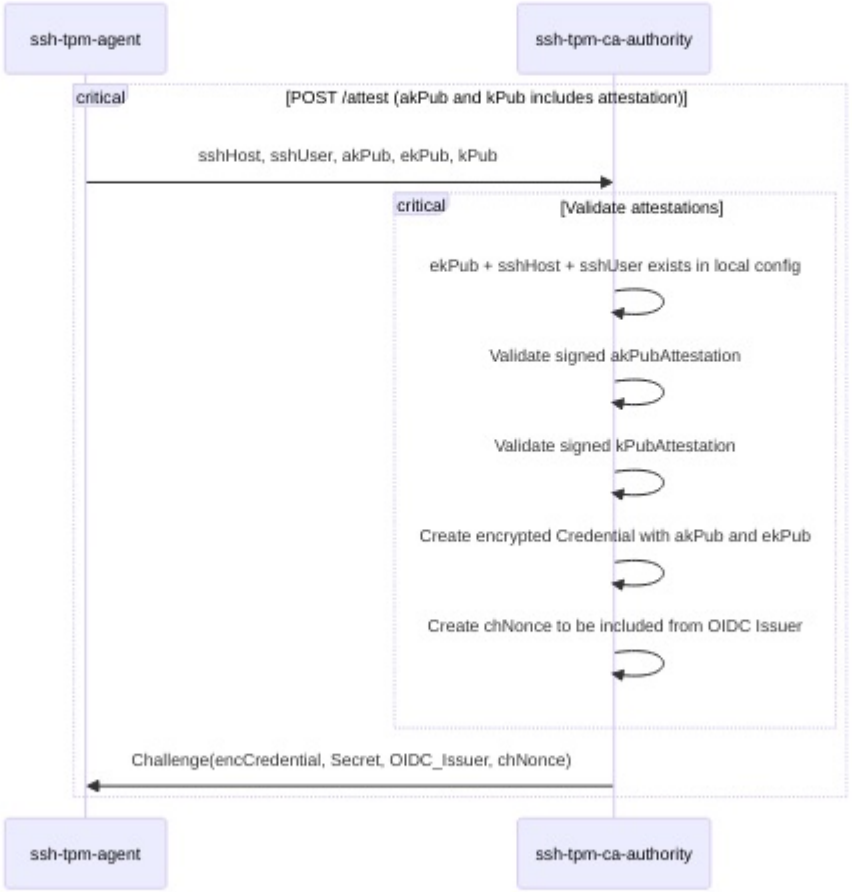
Key creation - Bound AK



Key creation - TPM Bound Key



Submit attestation



JSON Structs

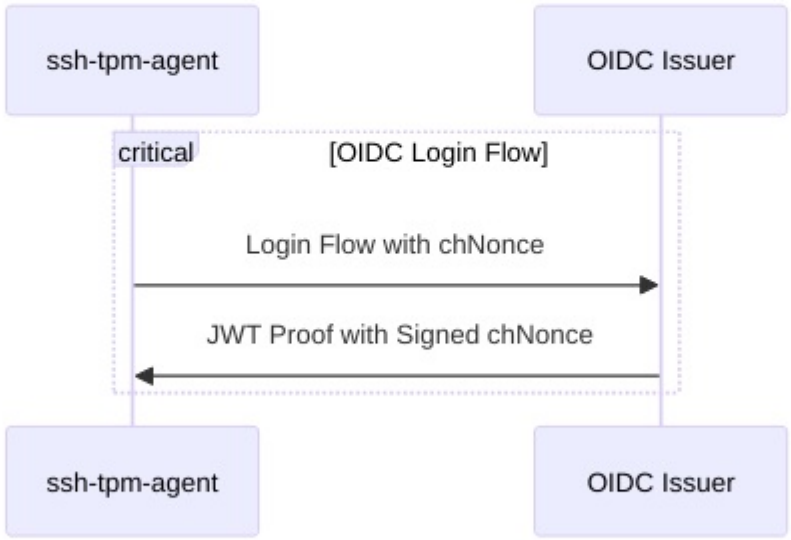
```
type Attestation struct {
    Public            *tpm2.TPMTPublic
    Signer            *tpm2.TPMTPublic
    CreateData        []byte
    CreateAttestation []byte
    CreateSignature   []byte
}

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

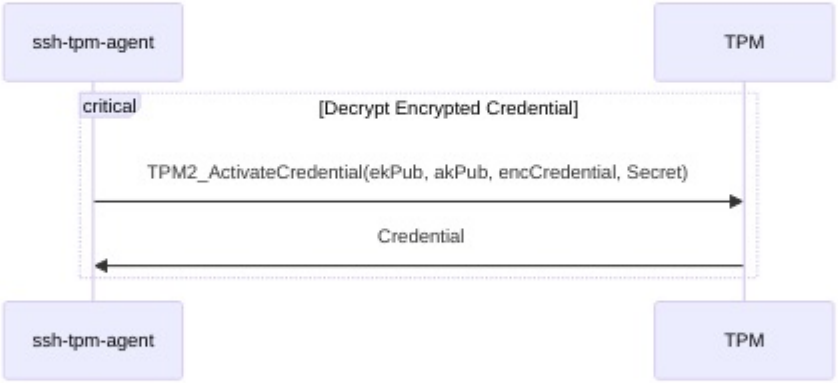
Challenge

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

OIDC Issuer challenge



Decrypt credential



Submit challenge

JSON Struct

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

Signed SSH Response

```
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