First Steps with AWS CloudHSM

A Minimal Signing Workflow Example

Gregor Ratajc

About me

Principal Product Security Engineer @ Unchained

Before that:

- Principal Security Engineer @ Bitstamp
- Infrastructure Security Lead @ Bitstamp
- Product Security Lead @ Bitstamp

https://si.linkedin.com/in/gregor-ratajc

About this talk

(1)
Infrastructure setup and CloudHSM provisioning

(2)
HSM user and quorum setup

(3)
Cryptographic material setup

(4)
Execution

About HSMs

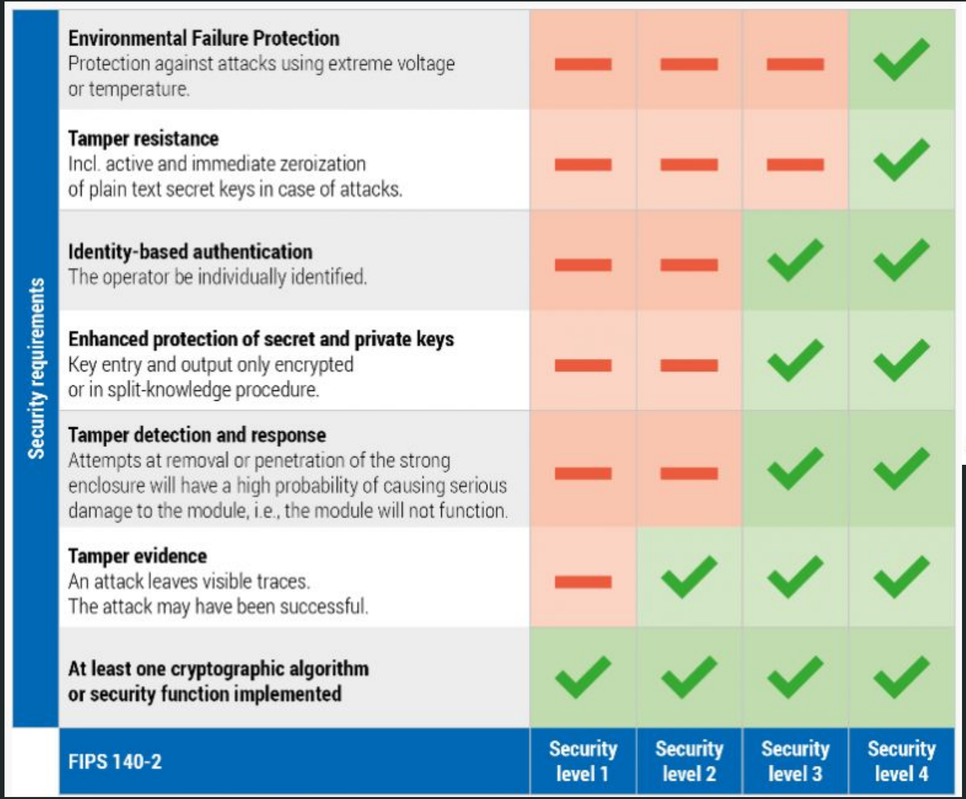
Main properties of HSMs:

- Secure key storage: keys never leave the HSM unencrypted
- Tamper-resistant & tamper-evident hardware
- Hardware-enforced isolation from host systems
- Cryptographic acceleration: offloads operations from applications
- Auditable and compliant with regulatory standards

Why use HSMs?

- Protect cryptographic keys from theft or misuse
- Meet strict security & compliance requirements (e.g., FIPS 140-2 Level 3)
- Enforce separation of duties and strong access control
- Enable trusted operations like digital signing, encryption, and key generation

About HSMs



Security requirements	**Environmental Failure Protection** Protection against attacks using extreme voltage or temperature.	—	—	—	✓
	Tamper resistance Incl. active and immediate zeroization of plain text secret keys in case of attacks.	—	—	—	✓
	Identity-based authentication The operator be individually identified.	—	—	✓	✓
	Enhanced protection of secret and private keys Key entry and output only encrypted or in split-knowledge procedure.	—	—	✓	✓
	Tamper detection and response Attempts at removal or penetration of the strong enclosure will have a high probability of causing serious damage to the module, i.e., the module will not function.	—	—	✓	✓
	Tamper evidence An attack leaves visible traces. The attack may have been successful.	—	✓	✓	✓
	At least one cryptographic algorithm or security function implemented	✓	✓	✓	✓
	FIPS 140-2	**Security level 1**	**Security level 2**	**Security level 3**	**Security level 4**

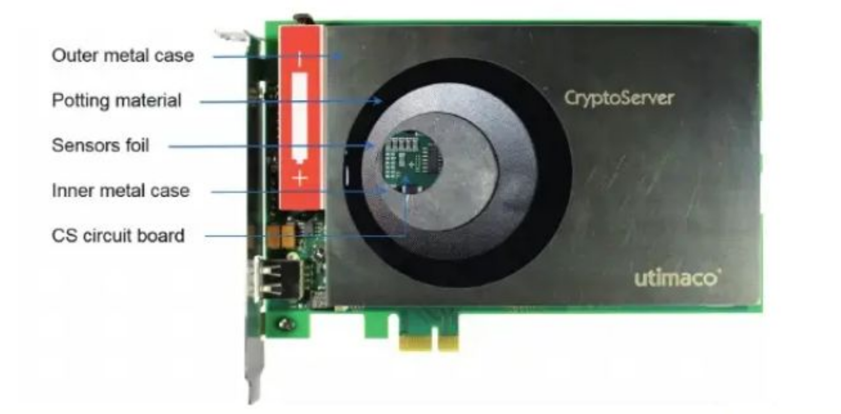


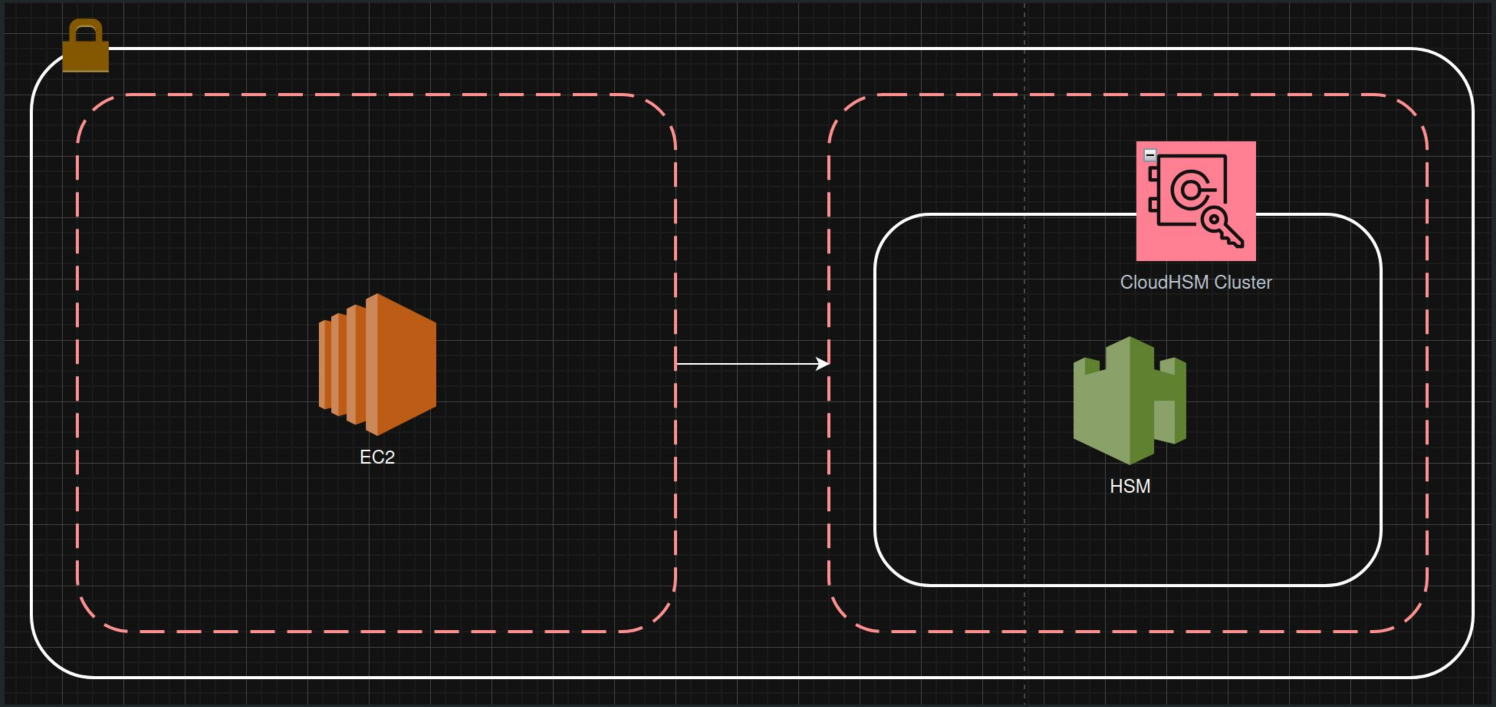
Figure 2.3: Cross-section of an HSM with tamper-responsive technology

Cost

From the time I first checked the pricing a lot has changed. At the time I believe it was about 5000 EUR/month fixed. Nowadays it is priced per hour, about 1.5 EUR/hour. Not too bad for experimentation, BUT do not leave it running :)

(1) Infrastructure setup

Minimum POC Setup



(1) Infrastructure setup - initialize the cluster

1. Sign the Cluster Signing Request (CSR)
2. Verify Cluster Verification Certificates (manufacturer, AWS, certificate chain)
3. Initialize connection from EC2 to HSM - cloudhsm-cli (use supported OS)
4. Upload your issuing certificate to EC2 to the predefined location
5. Set cluster password (admin)

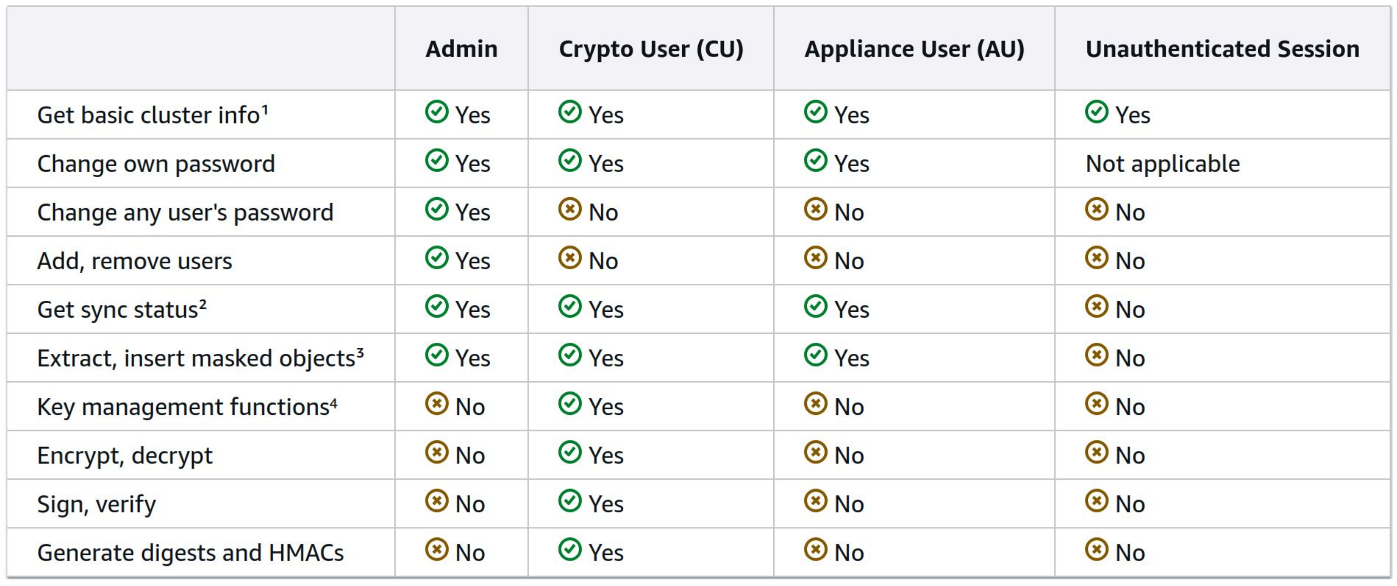
(2) Users and quorum setup

Admin
User management

Crypto User (CU)
Key management & crypto ops

Appliance User (AU)
Sync HSMs and clusters

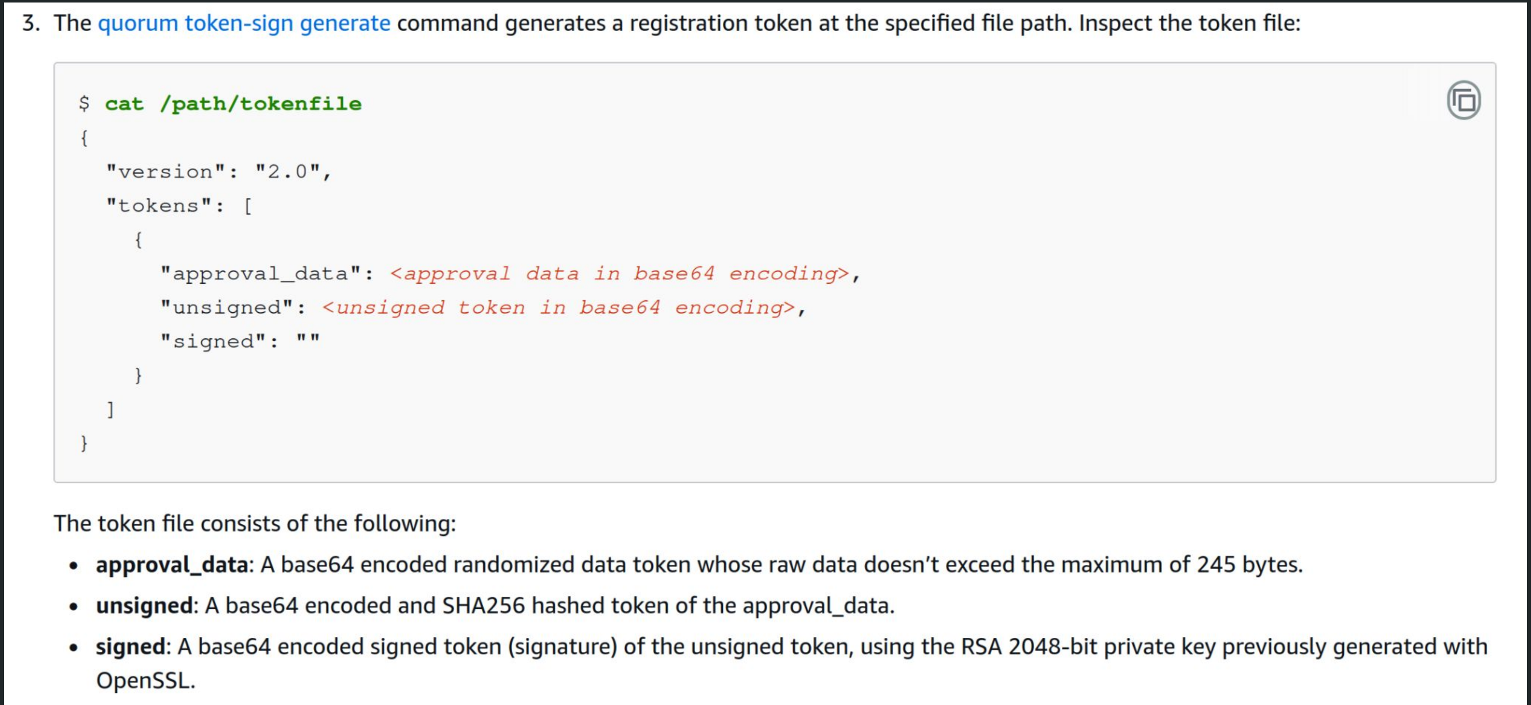
AWS
🚫



	Admin	Crypto User (CU)	Appliance User (AU)	Unauthenticated Session
Get basic cluster info[1]	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Change own password	✓ Yes	✓ Yes	✓ Yes	Not applicable
Change any user's password	✓ Yes	⊗ No	⊗ No	⊗ No
Add, remove users	✓ Yes	⊗ No	⊗ No	⊗ No
Get sync status[2]	✓ Yes	✓ Yes	✓ Yes	⊗ No
Extract, insert masked objects[3]	✓ Yes	✓ Yes	✓ Yes	⊗ No
Key management functions[4]	⊗ No	✓ Yes	⊗ No	⊗ No
Encrypt, decrypt	⊗ No	✓ Yes	⊗ No	⊗ No
Sign, verify	⊗ No	✓ Yes	⊗ No	⊗ No
Generate digests and HMACs	⊗ No	✓ Yes	⊗ No	⊗ No

(2) Quorum setup - m-of-n to sign

Crypto User (CU) - each user has a key pair that they use to sign quorum tokens



3. The quorum token-sign generate command generates a registration token at the specified file path. Inspect the token file:

```
$ cat /path/tokenfile
{
  "version": "2.0",
  "tokens": [
    {
      "approval_data": <approval data in base64 encoding>,
      "unsigned": <unsigned token in base64 encoding>,
      "signed": ""
    }
  ]
}
```

The token file consists of the following:

- **approval_data**: A base64 encoded randomized data token whose raw data doesn't exceed the maximum of 245 bytes.
- **unsigned**: A base64 encoded and SHA256 hashed token of the approval_data.
- **signed**: A base64 encoded signed token (signature) of the unsigned token, using the RSA 2048-bit private key previously generated with OpenSSL.

(3) Cryptographic material and m-of-n

Example: RSA key pair which has key quorum values of two (2) set for both key-management and key-usage operations. Public keys do not have quorum values.

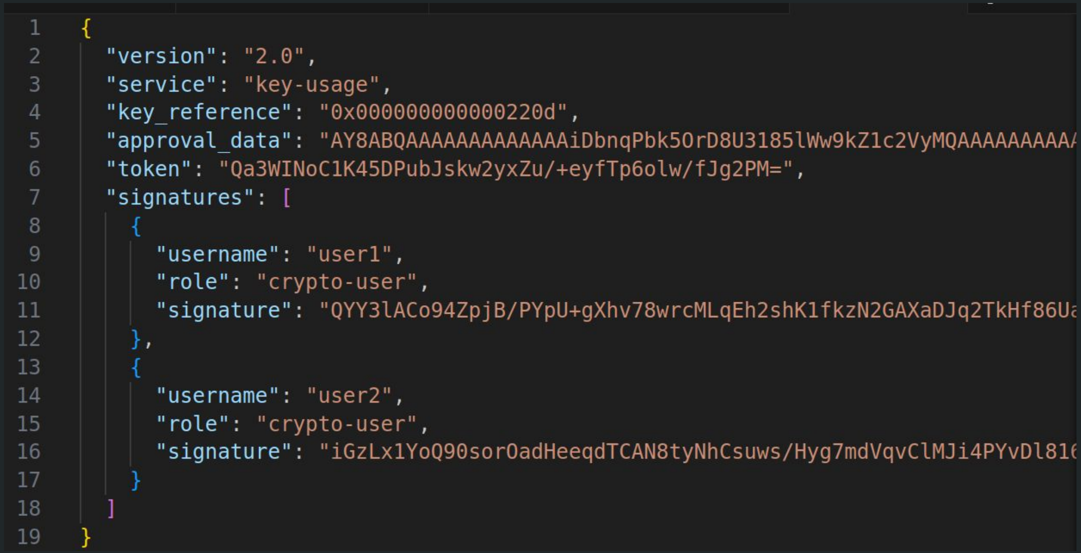
Max *n* is the number of crypto users.

```
> login --username user1 --role crypto-user

> key generate-asymmetric-pair rsa \
--public-exponent 65537 \
--modulus-size-bits 2048 \
--public-label rsa-public-key-example \
--private-label rsa-private-key-example \
--public-attributes verify=true \
--private-attributes sign=true
--share-crypto-users user2 user3 \       #n (user1 is the owner)
--manage-private-key-quorum-value 2 \    #m for key management
--use-private-key-quorum-value 2         #m for key usage
```

(4) Sign

1. Sign with user1 -> ❌ Quorum failed
2. Generate a quorum token
3. Get signatures from approving crypto-users
4. Approve the token on the CloudHSM cluster and execute an operation
5. The quorum token is one-time only - you cannot execute any operation with it after it has been used
6. Check logs



```
{
  "version": "2.0",
  "service": "key-usage",
  "key_reference": "0x000000000000220d",
  "approval_data": "AY8ABQAAAAAAAAAAAAAAiDbnqPbk5OrD8U3185lWw9kZ1c2VyMQAAAAAAAAAA
  "token": "Qa3WINoC1K45DPubJskw2yxZu/+eyfTp6olw/fJg2PM=",
  "signatures": [
    {
      "username": "user1",
      "role": "crypto-user",
      "signature": "QYY3lACo94ZpjB/PYpU+gXhv78wrcMLqEh2shK1fkzN2GAXaDJq2TkHf86Ua
    },
    {
      "username": "user2",
      "role": "crypto-user",
      "signature": "iGzLx1YoQ90sorOadHeeqdTCAN8tyNhCsuws/Hyg7mdVqvClMJi4PYvDl816
    }
  ]
}
```

Questions?

Gregor Ratajc
https://www.linkedin.com/in/gregor-ratajc/