LASIGE

VIRTUAL AND DISTRIBUTED HARDWARE SECURITY MODULE FOR SECURE KEY MANAGEMENT

Presented by: **Diogo Novo**

CONTEXT
WHAT IS AN HSM?

A Hardware Security Module (HSM) is a dedicated physical device that is specialized in **safeguarding the protection of cryptographic keys** during their whole life-cycle and performing major cryptographic operations.

HSMs offer a **trusted environment** that is impenetrable by malware, viruses, exploits, and unauthorized accesses. Having internationally recognized certifications that vouch for their security guarantees.

PROBLEM

WHAT IS THE PROBLEM WITH HSMS?

Costly and often impractical for smaller companies

Difficult to deploy and manage

Difficult to secure at large scale

"...this type of hardware would typically cost at least $20K to deploy, $40K to achieve high availability, and multiple times more for a typical enterprise deployment..."

"...deployment costs for real-world use cases would start at around $250K..."

- Fortanix

SOLUTION

A VIRTUAL AND DISTRIBUTED HSM!

Robust and efficient implementation of a **software-only HSM** - Virtual HSM.

Implementation of the Virtual HSM in a **distributed manner** to achieve the same security level of a physical HSM, and also high levels of availability, integrity and confidentiality.

Employment of a **Byzantine Fault-Tolerant (BFT) State Machine Replication (SMR) system** to make the solution realistic and practical.

Leverage efficient protocols from the field of **threshold cryptography** to perform the main cryptographic operations of an HSM in a *secure* and *distributed* manner.

BACKGROUND THRESHOLD CRYPTOGRAPHY

- **Threshold cryptography** corresponds to cryptographic algorithms where **multiple parties are needed to perform an encryption or signature**, employing a secure protocol that allows the required secrets to be used collectively, revealing only the output of the cryptographic operation.

- In contrast with physical HSMs, this alternative **requires that a certain *threshold* of devices be compromised** for an attacker to recover the secrets or violate the protocol's security, which makes the attacks more difficult to succeed.

- It can be divided into three main branches:

Secret Sharing | **Threshold Signatures** | **Threshold Symmetric Encryption**

RELATED WORK

SoftHSM

- \+ PKCS#11 Specification
- \+ Open-Source solution
- \- Local implementation
- \- Non-distributed solution
- \- No security guarantees

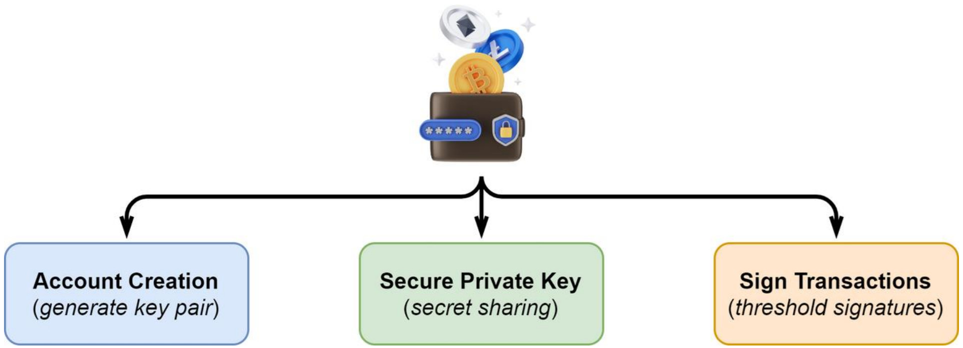
Poor Man's HSM

- \+ Threshold Cryptography
- \- Semi-distributed solution
- \- Single point-of-failure
- \- Synchronous communication
- \- Non fault-tolerant

Virtual HSM Thesis

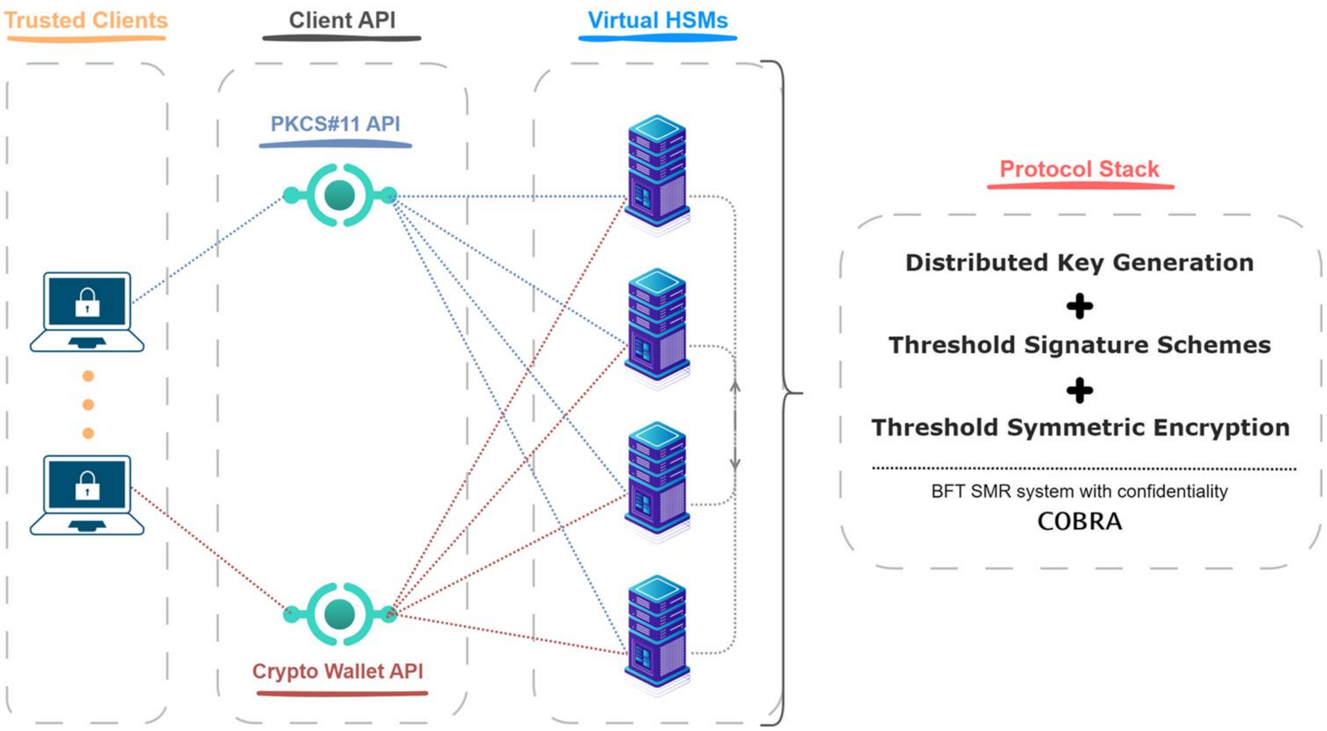
- \+ Privacy and high availability
- \+ Cloud-of-clouds replication
- \- Hardware-backed solution
- \- Intel SGX as the TEE
- \- Non fault-tolerant

RELATED WORK CRYPTOCURRENCY WALLETS



- Just like HSMs, cryptocurrency wallets are primarily responsible for **performing** cryptographic operations, particularly **signatures** and **storing and protecting private keys**.
- This union is mentioned in some papers; however, they use physical HSMs to secure private keys and to verify the user's identity through a Personal Identification Number (PIN).
- The **synergy between these technologies** can contribute to **improving the security of these wallets**, as the theft of these digital assets is always happening.

ARCHITECTURE



IMPLEMENTATION DISTRIBUTED KEY GENERATION (DKG)

- The COBRA *distributed polynomial generation protocol* was the foundation for implementing our distributed key generation. This protocol allows a group of servers to collectively create a random polynomial *P* of degree *t* with an *encoded point* in a fully distributed manner using *Byzantine Consensus*. **The encoded point corresponds to the secret or private key**, and the replicas receive only their corresponding *share* of the generated secret.

- Based on secret sharing, a scheme that **protects the confidentiality of a stored *secret s* by splitting it into *n* pieces**, called shares. For the secret to be recovered, a portion of its shares ***t*** must be combined, and fewer than ***t*** shares do not reveal any information about the secret.

- COBRA uses the **Dynamic Proactive Verifiable Secret Sharing**, or DPSS, which aggregates several improvements made to the initial Shamir's proposal, including share integrity, share renewal, and shareholder reconfiguration.

IMPLEMENTATION
THRESHOLD SIGNATURES

- A threshold signature scheme (TSS) **enables a group of parties to collectively compute a signature** without disclosing any information about their private key.

- **Non-interactive protocols** are the most efficient since they need only **two communication rounds** to conclude their protocol.

- The **Schnorr and BLS signatures** were chosen to be implemented since these are non-interactive, have simpler threshold versions of their centralized algorithm, and are compatible with Bitcoin and Ethereum, respectively.

Schnorr

$$s_i = k_i + e \cdot sk_i$$

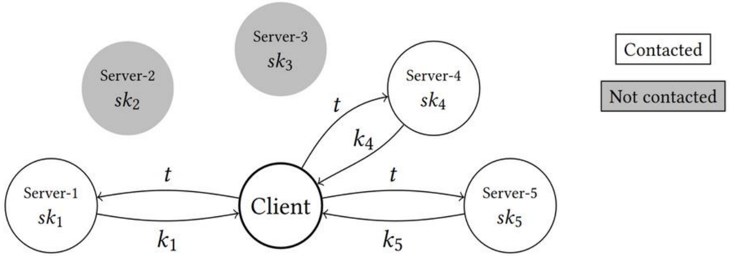
$$e = H(M||r)$$

BLS

$$s_i = H(M) \cdot sk_i$$

IMPLEMENTATION THRESHOLD SYMMETRIC ENCRYPTION

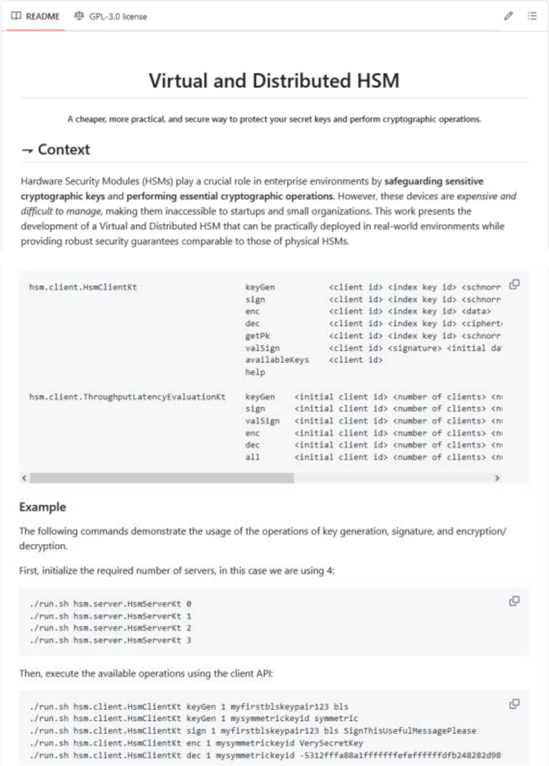
- As our encryption/decryption protocol, we implemented the proposal **Distributed Symmetric-key Encryption (DiSE)**, by Agrawal et al., which consists of a **generic construction of *threshold authenticated encryption* based on any *distributed pseudorandom function (DPRF)***.
- As the algorithm is based on DPRF, we **adapted the privately verifiable version** suggested in the article to allow the usage of *elliptic curves* and *commitments*.



DistEnc(m):

- $t \leftarrow$ commit(m)
- $k_1, k_4, k_5 \leftarrow$ query(t, {1,4,5})
- $k \leftarrow$ combine(k_1, k_4, k_5)
- $c \leftarrow$ Enc(k, m)

GITHUB & STRUCTURE



README GPL-3.0 license

Virtual and Distributed HSM

A cheaper, more practical, and secure way to protect your secret keys and perform cryptographic operations.

→ Context

Hardware Security Modules (HSMs) play a crucial role in enterprise environments by **safeguarding sensitive cryptographic keys** and **performing essential cryptographic operations**. However, these devices are *expensive and difficult to manage*, making them inaccessible to startups and small organizations. This work presents the development of a Virtual and Distributed HSM that can be practically deployed in real-world environments while providing robust security guarantees comparable to those of physical HSMs.

```
hsm.client.HsmClientKt                    keyGen         <client id> <index key id> <schnorr
                                          sign           <client id> <index key id> <schnorr
                                          enc            <client id> <index key id> <data>
                                          dec            <client id> <index key id> <ciphert
                                          getPk          <client id> <index key id> <schnorr
                                          valSign        <client id> <signature> <initial da
                                          availableKeys  <client id>
                                          help

hsm.client.ThroughputLatencyEvaluationKt  keyGen   <initial client id> <number of clients> <n
                                          sign     <initial client id> <number of clients> <n
                                          valSign  <initial client id> <number of clients> <n
                                          enc      <initial client id> <number of clients> <n
                                          dec      <initial client id> <number of clients> <n
                                          all      <initial client id> <number of clients> <n
```

Example

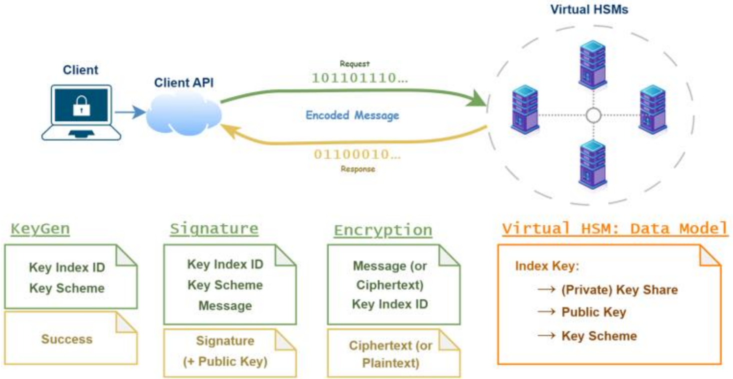
The following commands demonstrate the usage of the operations of key generation, signature, and encryption/decryption.

First, initialize the required number of servers, in this case we are using 4:

```
./run.sh hsm.server.HsmServerKt 0
./run.sh hsm.server.HsmServerKt 1
./run.sh hsm.server.HsmServerKt 2
./run.sh hsm.server.HsmServerKt 3
```

Then, execute the available operations using the client API:

```
./run.sh hsm.client.HsmClientKt keyGen 1 myfirstblskeypair123 bls
./run.sh hsm.client.HsmClientKt keyGen 1 mysymmetrickeyid symmetric
./run.sh hsm.client.HsmClientKt sign 1 myfirstblskeypair123 bls SignThisUsefulMessagePlease
./run.sh hsm.client.HsmClientKt enc 1 mysymmetrickeyid VerySecretKey
./run.sh hsm.client.HsmClientKt dec 1 mysymmetrickeyid -5312fffa88a1ffffffefeffffffdfb248282d98
```



EXPERIMENTAL RESULTS

Table 1: **Performance evaluation of our system** including latency mean (*ms*), standard deviation (*ms*), and operations per second (*Op/s*).

(n, t)		*(4, 1)*			*(7, 2)*		
Operation		*Mean*	*Std. Dev.*	*Op/s*	*Mean*	*Std. Dev.*	*Op/s*
DKG	Schnorr	20.51	1.67	48.76	25.87	2.04	38.65
	BLS	130.12	14.53	7.69	287.73	21.37	3.48
Signature	Schnorr	21.54	1.75	46.43	27.87	2.26	35.88
	BLS	81.74	4.29	12.23	150.14	19.03	6.66
Validation	Schnorr	3.88	1.02	257.78	3.70	0.74	270.11
	BLS	11.01	1.13	90.80	10.85	0.68	92.18
Encryption		52.66	3.74	18.99	75.26	5.49	13.29
Decryption		51.26	3.40	19.51	74.03	5.09	13.51

EXPERIMENTAL RESULTS

Table 2: **Performance evaluation of our system** and the results from SoftHSM and VirtualHSM (*hardware-based*) in Op/s.

(n, t)		*(4, 1)*		
Operation		*Mean*	*Std. Dev.*	*Op/s*
DKG	Schnorr	20.51	1.67	48.76
	BLS	130.12	14.53	7.69
Signature	Schnorr	21.54	1.75	46.43
	BLS	81.74	4.29	12.23
Validation	Schnorr	3.88	1.02	257.78
	BLS	11.01	1.13	90.80
Encryption		52.66	3.74	18.99
Decryption		51.26	3.40	19.51

Key Size	SoftHSM	VirtualHSM
Key Generation (RSA)		
1024-bit	5.63	77.57
2048-bit	4.36	16.06
3072-bit	3.44	5.51
Signature (RSA)		
1024-bit	4.46	4435.77
2048-bit	1.50	1559.0

Table 3: **Performance evaluation of the centralized versions of** Schnorr and BLS signatures.

Operation		*Threshold*	*Non-Threshold*
Signature	Schnorr	46.43	87.32
	BLS	12.23	429.75

CONCLUSIONS

To the best of our knowledge, this is the first work in which all these functionalities were accomplished using **threshold cryptography protocols in a realistic, practical, and fully distributed system**.

The implemented features allow our system to be used as a **cryptocurrency wallet** since both share similar responsibilities and joining these technologies only helps improve the security of these wallets.

FUTURE WORK

Implement the **Crypto-Wallet API** and the **PKCS#11 API**, the former being *custom-made* and the latter corresponding to a widely known and used specification in physical HSMs.

Develop **two small applications that interact with both APIs**. Specifically, implement a simple cryptocurrency wallet that uses the available functionalities and interacts with blockchains.

VIRTUAL AND DISTRIBUTED HARDWARE SECURITY MODULE FOR SECURE KEY MANAGEMENT

Presented by: **Diogo Novo**