Securing Hyperledger Supply Chain Apps

Rebecca Aspler
Director, Product Management
Hyperledger Fabric Security
Transaction Flow

• An application leveraging a supported SDK (Node, Java, Python) utilizes one of the available API's to generate a transaction proposal.
• The proposal is a request to invoke a chain-code function with certain input parameters, with the intent of reading and/or updating the ledger.
• The SDK takes the user's cryptographic credentials to produce a unique signature for this transaction proposal.
Authentication, Public keys, and Private Keys

Certificate Authority

Principal

Digital Certificate

Certificate Revocation List

request certificate

issue certificate

public key

private key
Certificate Authorities
Wallet Types

A wallet contains a set of user identities (=authentication keys). An application selects one of these identities when it connects to a channel.
Wallet Types

- File
- Memory
- HSM
- Database
- vHSM
Unbound MPC
Multi Party Computation

**Pure-software approach**
- The key never exists as one entity. It is created and maintained as $N$ random shares
- You can place the random shares at different places
- Use of shares without ever bringing them together
- The Share are refreshed after each transaction

**Underlying technology**
- MPC cryptography protocol
- Machines jointly working while keeping inputs private (Zero Knowledge Proof)
- Security guarantee – mathematically proven
Secure as Cold. Purely in Software

You control the full key lifecycle
Revoke keys instantly at any time

Never trust any single entity with key material, even when generated and when in use
Mathematical guarantee of confidentiality unless key shares from all nodes are exposed simultaneously

Keys can be used for any purpose by any service or app, anywhere, without obtaining key material
Key Part Refresh

Frequent refresh intervals using jointly chosen random number means attackers must have access to both servers simultaneously.

1. Machines M1 and M2 choose random number \( r \) via secure coin tossing protocol.

2. Given private key \( K \) and existing key shares \( K_1 \) and \( K_2 \):
   - M1 computes \( K'_1 = K_1 + r \)
   - M2 computes \( K'_2 = K_2 - r \)

Given \( K_1 \) and \( K'_2 = K_2 - r \), nothing can be learned about the private key, \( K \).
CASP – Risk Based Policies

- Distributed cryptography - no single point of compromise
- Key material is never in the clear
- Supports any device and platform
- Async approval of transactions
- Ledger Agnostic
- Sophisticated MofN Quorums
# Unbound Crypto Asset Security Platform

## Key Features

<table>
<thead>
<tr>
<th>Pure Software</th>
<th>Asset Agnostic</th>
<th>Crypto Agile</th>
</tr>
</thead>
<tbody>
<tr>
<td>A software solution providing hardware level security using MPC for Blockchain based Crypto Assets.</td>
<td>Support for the top any asset, any platform and any client.</td>
<td>Supporting ECDSA and EDDSA(Ed25519) curves; Adding new curves as needed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deterministic Wallets</th>
<th>Multi Party Approval</th>
<th>Risk Based Policies</th>
</tr>
</thead>
</table>
| Supporting BIP 32/44 Cryptographically Enforced | M-of-N quorum (in N groups) enforcement, multiple approvers required for transactions. | Example risk related parameters:  
- Amount,  
- Asset type,  
- Time in the week  
- Time in the day |
Case Study
Supply Chain Case Study
Keys Management - Transaction Signing

**Transaction Signing**

- As an overarching **Security Guideline**, Account Keys once stored on HSM will never leave the HSM.
- Keys Management module exposes a **robust and lightweight API** for application integration.
- Applications will need to **sign transactions** via Key Management APIs before posting to respective blockchains.
- **High performance** to ensure high system throughput.

**Example**

App sends Raw Transaction for signing

**RiskBlock Apps**

Raw Transaction

**APIs**

Signed Transaction

Signed Transaction posted to blockchain

**Keys Management**

Signed Transaction returned from Keys Mgmt.

**HSM**

**Blockchain**

**Raw Transaction**
**Transaction Signing**

- As an overarching **Security Guideline**, Account Keys once stored on HSM will never leave the HSM.
- Keys Management module exposes a **robust and lightweight API** for application integration.
- Applications will need to **sign transactions** via Key Management APIs before posting to respective blockchains.
- **High performance** to ensure high system throughput.