HDS Coin - Stage 1
SEC 2017-2018

Gabriel Maia, 81878
Instituto Superior Técnico, Lisboa
gabriel.maia@tecnico.ulisboa.pt

Artur Esteves, 91063
Instituto Superior Técnico, Lisboa
arturesteves@tecnico.ulisboa.pt

Ricardo Morais, 91064
Instituto Superior Técnico, Lisboa
ricardojhmorais@tecnico.ulisboa.pt

Our HDS Coin implementation respects all of the requirements specified in the project assignment, including the security and dependability guarantees. In our system, the server and each ledger are uniquely associated with a ECDSA key pair using the secp256r1 curve; each party never discloses its private keys. Elliptic-curve cryptography was selected because, for equivalent security levels, EC keys are much shorter than RSA keys [1]. This provides an acceptable user experience when identifying each ledger.

Plain HTTP, without any protocol-level security features such as TLS, is used for communication between the clients and the server. The server exposes a REST API and uses JSON serialization for transferring objects, such as transaction requests and account audit data. Signatures (generated/verified using the ECDSA private/public keys of each party, respectively) provide message integrity and non-repudiation where needed. The server stores ledgers and their transaction history in a SQLite database; the clients do not store any information other than their key pairs.

Clients open an account in the system in a non-repudiable way by signing an initial transaction in which they declare the initial balance of the account. We assume that, like on a real bank, opening an account incurs a cost, making it impractical to open a large number of accounts, therefore mitigating Sybil attacks. Account registration is protected against message duplication, as the server will only allow each public key to register once, and against message manipulation, as the registration request is signed by the registering ledger.

In our system, a complete transaction consists of two parts: one is associated with the source ledger, who immediately spends their balance when submitting a send request, and the other is associated with the receiving ledger, whose balance is only affected when he submits a valid receive request. Clients sign their part of the transaction, which contains the source, destination, amount, a random nonce (to help ensure all transactions are unique), and the signature of their previous transaction (thus building a transaction chain for each ledger, ensuring transaction order is not manipulated). When confirming a transaction, clients also include the hash of the pending sending transaction as part of the signed request. The server enforces the uniqueness of each transaction by rejecting transactions with the same signature as a previous one, protecting against message duplication. Transaction signatures, checked by the server, protect against message manipulation.

Message freshness, integrity and non-repudiation is provided with each server response, which is especially important in the checkAccount and audit operations, as clients rely on their result to know the state of their accounts and to sign future transactions - clients don't store any information; they must always request their transaction history from the server by auditing themselves before creating new transactions or accepting pending ones.

With each request, clients include a random nonce they wish to see repeated in the response. This nonce is sent as a HTTP header or as part of the request object (transactions already include a nonce). A correct server must then include this nonce in the response. The whole response (including the nonce) is signed by the server; the signature is provided over a HTTP header. Clients check this signature using the server public key, which they must know, and confirm that the nonce in the response matches the nonce they included in their request. This mechanism protects againt repetition of server responses.

Protection against repetition of client requests is not necessary, because all sensitive operations (registration and sending/receiving money) are otherwise protected against duplication, as previously explained.

References

[1] E. Barker. Recommendation for key management. *NIST Special Publication 800-57 Part 1 Revision 4*, page 53.