Guardian Handshake - Certificate Exchange

Tim Bansemer[*]

October 16, 2024

1 Introduction

This document describes the cryptographic interactions between two Guardians and their authoritative users.

The goal is to establish mutual trust between the two Aqua-Nodes. The process of how to establish trust is described in this document.

Each Aqua-Node consists of one Guardian and one Aqua-Container, specifically a Personal-Knowledge-Container (PKC). The two Aqua-Nodes are managed by users Alice and Bob. Users as well as Guardians have their own public/private key. The Guardians store their keys internally, while the users' keys are stored externally either in a software or hardware wallet. Additionally, the Guardians serve as security and enforcement points within the system.

Guardians establish secure communication with each other using Transport Security Layer (TLS).

After trust is established between Alice's and Bob's Guardians permissioned Aqua-Chains which are stored within the PKCs might be exchanged. The Guardians will act as enforcement and security components.

[*]Review: Ruben Hans Ehritt, Mika Elias Richter

2 Key/Cert Declarations:

Type	**Description**
K^{+}_{GB}	Public key of Bob's Guardian
K^{+}_{GA}	Public key of Alice's Guardian
K^{-}_{Bob}	Private key of Bob
K^{-}_{Alice}	Private key of Alice
K^{+}_{Bob}	Public key of Bob
K^{+}_{Alice}	Public key Alice
$Cert_{\text{GB}}$	Certificate of Bob's Guardian
$Cert_{\text{GA}}$	Certificate of Alice's Guardian

Table 1: Key and certificate declarations

2.1 Certification

Following the X.509 certificate structure within the Aqua-Protocol Guardian's public key (subject of the certificate) is signed by its authoritative user's private key (acting as the certificate authority and certificate issuer) to issue certificates attesting the Guardians identity:

$$K^{+}_{\text{GB}} + K^{-}_{\text{Bob}} = Cert_{\text{GB}}$$

$$K^{+}_{\text{GA}} + K^{-}_{\text{Alice}} = Cert_{\text{GA}}$$

3 Trust establishment

Assuming an initial exchange of messages between Bob and Alice occurs via a secure and confidential channel, the following steps are to be undertaken in order to establish trust:

- Bob sends his public key and his Guardian's certificate to Alice.

$$K^{+}_{\text{Bob}} + Cert_{\text{GB}} \tag{1}$$

- Alice sends her public key and her Guardian's certificate to Bob.

$$K^{+}_{\text{Alice}} + Cert_{\text{GA}} \tag{2}$$

4 Verification

In this project, the signatures of the certificates are done using K^{-}_{Alice}, K^{-}_{Bob} with Aqua-Chains which wrap the certificate issued by the Guardians.

All trust assumptions come from the users sharing each others public keys and verifying each others signatures on the Guardian certificates before importing them into their PKC. The valid certificates are unwrapped from the Aqua-Chain and imported into the local trusted root certificate store of the Guardian.

This allows the TLS handshake to be completed with a client and a server certificate using mutual TLS (mTLS) where both the client and the server authenticate each other using certificates.

5 Final established state

There is a mutually authenticated TLSv1.3 connection between the Guardians. Two server-client sessions are established where a Guardian acts as a client and as a server establishing secure and trusted connections. Both sessions use their own issued certificates respectively as a server and a client certificate.

A schema of the steps described above can be found here.

6 Used ciphers and limitations with TLSv1.3

Used cryptographic ciphers utilize Ethereum wallets using ECDSA algorithm specifically secp256k1.

Using Transport Layer Security Version 1.3 (TLSv1.3) to establish trust between the Guardians is desirable as it is the latest version of the protocol.

Currently it is not possible to use TLSv1.3 with secp256k1 as mentioned here since "there are no suitable signature schemes in the IANA TLS Signature Scheme registry" for TLSv1.3.

As we are currently implementing a prototype which limits the ability of the project team to choose a different encryption cipher, such as the BSI (recommended secp256r1), the options left are either to fall back to TLSv1.2 to issue and sign the certificates or use other workarounds.

6.1 Implementation

The Guardian is written in Rust. The Rust library rustls used in the project supports ECDSA_NISTP256_SHA256 which is secp256r1 but not *secp256k1* which will potentially cause implementation limitations.

Implementation resources for mTLS in rust are:

- rustls::ConfigBuilder
- rust mTLS example

6.1.1 Workaround

A workaround is to use TLSv1.3 with secp256r1 keys generated by the Guardians in parallel to the secp256k1 keys. The secp256r1 keys are used for creating self-signed certificates which are then signed within the Aqua-Chain by the sec256k1 keys of the authoritative users.

The certificate structure can be altered within the Aqua-Chain to include the secondary public key as well as the self-signed certificate of the Guardians. A support for secp256r1 is proposed and might be supported in the future following EIP-7212.

6.1.2 Client-Identification

For the Guardians to select what data is accessible to whom a form of secure session management for clear differentiation of the incoming requests is needed. This can be done by identifying the client session by reading the client certificate and extracting, for example, the certificates Common Name (CN) which can be set to contain the wallet key of the Guardian or another unique identifier.

The Client Certificate from Session is being used.

7 Conclusion

For the establishment of trusted communication between the Guardians it is benefitial to stay as close as possible to the established and the latest security standards, such as TLSv1.3. Utilizing well-known and trusted procedures simplifies the implementation process and makes it easy for third parties to relate to the security practices of the project.

Creating new procedures, on the other hand, leads to an additional complexity and to potential implementation mistakes and increases required time investments to understand the proposed security mechanisms. This can lead in the best case scenario to reduced trust and in the worst case to a compromised security model.

mTLSv1.3 provides all functionality required to establish secure connections between Guardians with clear client identification required for permissioned data exchange which makes it a suitable candidate for a first transport layer implementation.

8 Schema

1. A Guardian can act as a server and as a client
2. There are 2 sessions (Client – Server and Server – Client) between every pair of communicating Guardians
3. Client Guardian issues and signs a certificate containing its public key and imports it into PKC (1)
4. User of the PKC signs the certificate of his Guardian (2)
5. User exports via an independent communication channel the signed Guardian certificate alongside with user's public key to the user of server-side Guardian (3)
6. User of the server-side Guardian verifies the incoming Aqua-Chain, imports it into his PKC and signs it (4-5)
7. Server-side Guardian reads the user-signed data and stores the certificate of client-side Guardian (6-7)
8. Thereafter a Client – Server session can be established with session ID based on the client certificate
9. The same way a second session is established

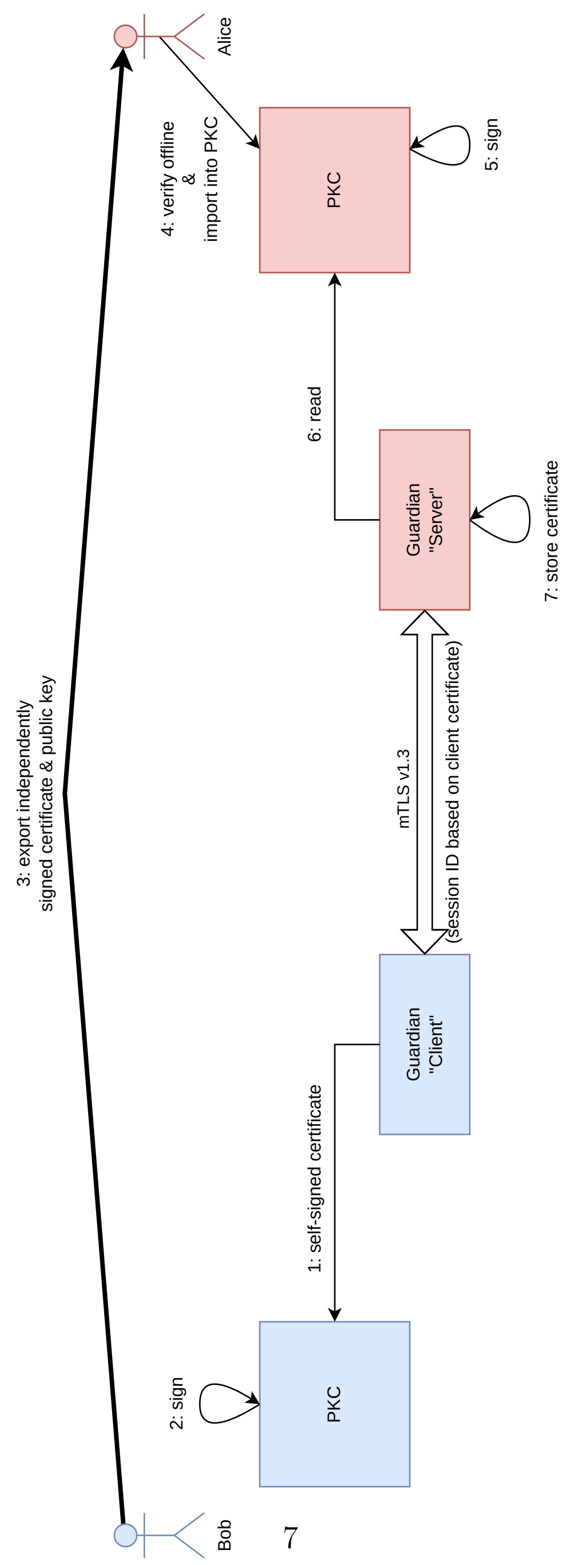
Alice
PKC
5: sign
4: verify offline & import into PKC
6: read
Guardian "Server"
7: store certificate
3: export independently signed certificate & public key
mTLS v1.3
(session ID based on client certificate)
Guardian "Client"
1: self-signed certificate
PKC
2: sign
Bob

Figure 1: Guardian Handshake