EtF Aura Overview

Tony Riemer

June 2023

1 Basic Setup: No thresholds

Assume that we have a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$, a hash-to-G1 function $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \mathbb{G}_2 \to \{0,1\}^n$ (for example, sha256).

Let $A = \{A_i\}_{i \in [n]}$ be the set of authorities for some $n > 1$. Assume that the we are using a proof-of-authority round-robin based slot lottery that runs in sequential epochs, $e_1, e_2, ...$ and so on, where each epoch e_k consists of a static number of slots, $sl_1, ..., sk_\delta$ for some $\delta > 1$. An epoch can run in several sessions. Our goal is to construst an 'encryption to the future' network on top of Aura consensus. We do this by realizing a threshold IBC among the validator set, allowing for blocks to be sealed using identity based signatures. In order to do so, we introduce a key derivation function (KDF) based on secret sharing techniques. Using Aura, we can calculate the 'slot' schedule quite easily since the validator set is static. Given the set A, the slot winner for slot sl_j in epoch e_k is $A_{j \mod |A|}$.

Using the well defined slot schedule, we can also define identities for each slot, for example, by defining $ID_{j,k} = A_{j \mod |A|} || e_k || sl_j$ where $||$ is concatenation. From this, each authority calculates a public key $\hat{Q}_{j,k} = H_1(ID_{j,k})$, and later we will see how a secret key can be derived.

1.1 Setup + keygen

During the setup phase, the Boneh-Franklin IBE requires that a semi-trusted authority issues a secret key that is known to all members of A. Rather than issuing a single secret key, we will use a distributed key generation protocol to create a shared secret at the beginning of each session. The implication of this is that later on, when encryption, at least a threshold of validators will have to issue a secret (i.e. behave honestly) in order for decryption to occur.

At the end of each session, assume that some randomly selected authority is made responsible for calculating the next *session seed*, a randomly selected $s \in \mathbb{Z}_p$. They also produce a *session public key*, $P_{pub} = sP \in \mathbb{G}_1$, where $P \in \mathbb{G}_1$ is a generator. Then, each authority derives a secret $R_i \in \mathbb{Z}_p$ and shares the public key $Q_i = R_i \cdot P$.

1.2 Identity-Based Block Seals and Validation

During a slot winner A_i's slot s_i, they are responsible for sealing a block and encoding the secret key in the block header. Secret keys, in this sense, are similar to a one-time-pad. Let $B_{j,k}$ be the block proposed by slot winner A_j during epoch e_k. Then a signature is calculated as:

$$\langle S_{j,k}, T_{j,k} \rangle = \Big\langle R_j \hat{Q}_{j,k} + rH(B_{j,k}), rP \Big\rangle \tag{1}$$

where R_j is the secret created during setup ($Q_i = R_j P$) and $r \in \mathbb{Z}_p$ is randomly selected.

The signature is verified by checking the pairing:

$$e(S_{j,k}, P) = e(\hat{Q}_{j,k}, Q_j) + e(H(B_{j,k}), T_{j,k}) \tag{2}$$

Observe:

$$\begin{aligned} e(S_{j,k}, P) &= e(R_j \hat{Q}_{j,k} + rH(b)) \\ &= e(R_j \hat{Q}_{j,k}, P) + e(rH(b), P) \\ &= e(\hat{Q}_{j,k}, R_j P) + e(H(b), rP) \\ &= e(\hat{Q}_{j,k}, Q_j) + e(H(b), T_{j,k}) \end{aligned} \tag{3}$$

1.3 Encryption to the future

Let $M \in \{0,1\}^*$ be the message we want to encrypt and let $r \in \mathbb{Z}_p$ be random. There are two options here... Firstly, we can encrypt to a given slot seed and then use a some type of symmetric crypto to do encryption/decryption when it publishes its secret key. This is pretty easily done by following BasicIdent:

$$C = \langle U, V \rangle = \Big\langle rP, M \oplus H_2(e(\hat{Q}_{j,k}, Q_j)) \Big\rangle \tag{4}$$

Then decryption is accomplished by calculating:

$$V \oplus H_2(e(R_j \hat{Q}_{j,k}, U)) = M \tag{5}$$

Observe

$$\begin{aligned} e(R_j \hat{Q}_{j,k}, U) &= e(R_j \hat{Q}_{j,k}, rP) \\ &= e(\hat{Q}_{j,k}, P)^{R_j, r} \\ &= e(\hat{Q}_{j,k}, R_j P)^r \\ &= e(\hat{Q}_{j,k}, Q_j^r) \end{aligned} \tag{6}$$

2 DKG Setup

There is a major flaw in the result above. Mainly, it implies that a single authority must be trusted in order for decryption to be realized. If for any reason the authority fails to publish a key, the encrypted data will forever remain unencrypted (until quantum computers can crack it, of course). With that said, we need to use an MPC protocol, a distributed key generator, to establish a shared session public/private keypair, where each validator has a share of the key. For this, we'll assume that we still have a semi-trusted authority (i.e. the root).

2.1 Setup/Keygen

The root node generates a degree t polynomial $f(x) = \sum_{i=0}^{t}(a_i x^i)$ where each a_i is randomly selected from $\mathbb{Z}_p$. Then it generates shares $f(1), ..., f(n)$ (corresponding to each slot $sl_1, ..., sl_n$ in the epoch) and Pedersen commitments $F(1), ..., F(n)$. The first coefficient, $f(0)$, is the 'master secret key'. Let $P_{pub} = f(0)P$ be the 'master public key' which is published before each session.

Now, we need to distribute the keys. The simplest way is to use a symmetric crypto algorithm, for example El Gamal, using the underlying keys of the authorities. That is, in the network each authority has a keypair (sk_i, pk_i) where sk_i is securely stored offchain and accessible via the keystore. That is, for each $i = 1, ..., n$, the semi-trusted authority generates a random $k_i \in \mathbb{Z}_p$ and calculates $(C_i = k_i P K_i + H_1(M) \in \mathbb{G}_1, K_i = k_i P \in \mathbb{G}_q)$. On session planning, each authority $A_i \in A$ recover the secret share by computing $f(i) = C_i - sk_i \cdot K_i$.

Now, in context of the setup phase in the previous section, instead of choosing a random $R_i \in \mathbb{Z}_p$, each $A_j \in A$ sets $R_j = f(j)$, and so they have the public key $\hat{Q}_{j,k} = H_1(ID_{j,k})$ and the secret key $R_j \hat{Q}_{j,k} = f(j)\hat{Q}_{j,k}$. They compute $Q_j = R_j P$ and publish the value.

2.2 Signing and Verification

$$\langle S_{j,k}, T_{j,k} \rangle = \left\langle R_j \hat{Q}_{j,k} + rH(B_{j,k}), rP \right\rangle \tag{7}$$

where $r \in \mathbb{Z}_p$ is randomly selected and $\hat{Q}_{j,k} = H_1(ID_{j,k})$, as above.

Signature verification requires that a threshold The signature is verified by checking the pairing:

$$e(S_{j,k}, P) = e(\hat{Q}_{j,k}, P_{pub}) + e(H(B_{j,k}), T_{j,k}) \tag{8}$$

Observe for any given j, k, we have that:

$$\begin{aligned} e(S_{j,k}, P) &= e(R_j \hat{Q}_{j,k} + rH(b)) \\ &= e(R_j \hat{Q}_{j,k}, P) + e(rH(b), P) \\ &= e(\hat{Q}_{j,k}, R_j P) + e(H(b), rP) \\ &= e(\hat{Q}_{j,k}, Q_j) + e(H(b), T_{j,k}) \end{aligned} \tag{9}$$

The choice of publishing Q_j here implies that we do not need to use threshold signatures, instead we verify with the published verification key. In the case where we are using a proof of authority style consensus, this works really nicely. For a network using a proof of stake style consensus, threshold signatures might make sense here.

2.3 Threshold Encryption

Distinct from the non-threshold requirement for signatures, we DO want to be able to have a threshold requirement for decryption.

To encrypt data, instead of encrypting for a specific slot, we will encrypt for the entire epoch. If a threshold of validators behave honestly, then the message will be made decrypytable with high probability by the end.

Recall, we want our encryption function to be identity based, so we need an identity that can be derived from the slot schedule itself. But also... it would be best if the encryptor could specifiy the threshold that they want. Yeah! that's what I want, and it's going to happen in a 'linear' way. So I can choose any k-of-$|A|$ threshold, and the higher the K the longer the potential wait for decryption, but the lower chance of it not getting decrypted at all, unless the value is too high... there's defintiely a sweet spot, but we can leave that up to whatever the encrypting parties want.

So to encrypt, let $\hat{Q} = \sum_j \hat{Q}_{j,k}$ and compute:

$$C = \langle U, V \rangle = \left\langle rP, M \oplus H_2(e(\hat{Q}, P_{pub})) \right\rangle \tag{10}$$

And then to decrypt, we will need to collect at least a threshold t of secrets, where t is the degree of the polynomial created by the semi-trusted authority (prior to the session start). That is, assume that we know $R_1, ..., R_k$ for some $k \geq t$. Then we can recover the message by calculating:

$$V \oplus H_2(e(S\hat{Q}, U)) = M \tag{11}$$

where $s = f(0)$ is the master secret key. Since we used threshold encryption and we know enough the the R_is, we can compute s using Lagrange interpolation.

Observe:

$$\begin{aligned} e(s\hat{Q}, U) &= e(s\hat{Q}, rP) \\ &= e(\hat{Q}, P)^{s,r} \\ &= e(\hat{Q}, sP)^r \\ &= e(\hat{Q}, P_{pub})^r \end{aligned} \tag{12}$$