



Fabcoin crypto crash course

Signatures and aggregate signatures

Todor Milev
FA Enterprise System, Inc.

Spring 2019



1 Signature algorithms

- Digital signature algorithm (DSA)
- Schnorr aggregate signatures
- Schnorr aggregate signature - Zilliqa
- Schnorr aggregate signature - [1, MPSW]



(Generalized Digital Signature Algorithm: generate public key)

- Given: cyclic group \mathcal{G} of prime order $p = |\mathcal{G}|$, generator $g \in \mathcal{G}$.
- Choose at random the private key $\text{secret} \in \{0, 1, \dots, p-1\}$.
- Compute the public key $\text{pub} = g^{\text{secret}}$.

(Generalized DSA: sign message)

- 1 Given: a number digest (message), $f\text{-}n$: $\text{toNumber} : \mathcal{G} \rightarrow \mathbb{Z}$.
 - 2 Choose at random $\text{nonce} \in \{0, \dots, p-1\}$.
 - 3 Compute $\text{challenge} = \text{toNumber}(g^{\text{nonce}}) \bmod p$.
 - 4 Compute $\text{solution} = \frac{\text{digest} + \text{challenge} \cdot \text{secret}}{\text{nonce}} \bmod p$.
 - 5 (challenge, solution) - final signature.
- Public key pub - element of \mathcal{G} but secret key secret is a number!
 - toNumber must be close to one-to-one: diff. in's \Rightarrow diff. out's.
 - Usually: $\text{digest} = H(M)$, where H - hash f-n, M - message to sign.
 - M , pub : not part of signature, but implied to be accessible.
 - Signature often denoted by (r, s) : r = challenge, s = solution.



(Generalized DSA: signing, key generation summary)

- *Private key: secret. Public key: $\text{pub} \stackrel{?}{=} g^{\text{secret}}$, g - generator of \mathcal{G} .*
- $\text{nonce} \in \{0, \dots, p-1\}$ - *secret random use-once num.; $p = |\mathcal{G}|$.*
- $\text{challenge} \stackrel{?}{=} \text{toNumber}(g^{\text{nonce}})$, $\text{solution} \stackrel{?}{=} \frac{\text{digest} + \text{challenge} \cdot \text{secret}}{\text{nonce}} \bmod p$.
- $(\text{challenge}, \text{solution})$ - *final signature.*

(Generalized DSA: verify signature)

$$1 \quad a = \frac{1}{\text{solution}} \stackrel{?}{=} \frac{1}{\frac{\text{digest} + \text{challenge} \cdot \text{secret}}{\text{nonce}}} = \frac{\text{nonce}}{\text{digest} + \text{challenge} \cdot \text{secret}} \bmod p.$$

$$2 \quad u_1 = a \cdot \text{digest} \stackrel{?}{=} \frac{\text{nonce} \cdot \text{digest}}{\text{digest} + \text{challenge} \cdot \text{secret}} \bmod p.$$

$$3 \quad u_2 = a \cdot \text{challenge} \stackrel{?}{=} \frac{\text{nonce} \cdot \text{challenge}}{\text{digest} + \text{challenge} \cdot \text{secret}} \bmod p.$$

$$4 \quad \begin{aligned} X &= g^{u_1} \text{pub}^{u_2} \stackrel{?}{=} g^{\frac{\text{nonce} \cdot \text{digest}}{\text{digest} + \text{challenge} \cdot \text{secret}}} \cdot (g^{\text{secret}})^{\frac{\text{nonce} \cdot \text{challenge}}{\text{digest} + \text{challenge} \cdot \text{secret}}} \\ &= g^{\frac{\text{nonce} \cdot \text{digest}}{\text{digest} + \text{challenge} \cdot \text{secret}} + \frac{\text{nonce} \cdot \text{secret} \cdot \text{challenge}}{\text{digest} + \text{challenge} \cdot \text{secret}}} = g^{\frac{\text{nonce} \cdot (\text{digest} + \text{challenge} \cdot \text{secret})}{\text{digest} + \text{challenge} \cdot \text{secret}}} = g^{\text{nonce}} \end{aligned}$$

$$5 \quad \text{If } \text{toNumber}(X) \stackrel{?}{=} \text{challenge: signature is valid (invalid otherwise).}$$

? = potential for cheating. If nonce is revealed secret can be computed.



(Schnorr signature: key generation (same as gen. DSA))

- *Private key*: secret. *Public key*: $\text{pub} = g^{\text{secret}}$, g - generator of \mathcal{G} , $|\mathcal{G}| = p$ is prime.

(Schnorr signature: signing)

- *Given*: a number digest (message).
- *Choose at random* $\text{nonce} \in \{0, \dots, p-1\}$.
- *Compute* $\text{challenge} = g^{\text{nonce}}$.
- *Compute* $\text{solution} = \text{nonce} + \text{secret} \cdot \text{digest}$.
- $(\text{challenge}, \text{solution})$ - *final signature*.

(Schnorr signature: verification)

- $a = \text{challenge} \cdot \text{pub}^{\text{digest}} \stackrel{?}{=} g^{\text{nonce}} \cdot (g^{\text{secret}})^{\text{digest}} = g^{\text{nonce} + \text{secret} \cdot \text{digest}}$.
- $b = g^{\text{solution}} \stackrel{?}{=} g^{\text{nonce} + \text{secret} \cdot \text{digest}}$.
- If $a \stackrel{?}{=} b$ the signature is valid (invalid otherwise).



- To generate an aggregate signature (multi-signature) means to have independent parties combine their signatures into a single smaller signature.
- We describe two Schnorr-based aggregate signatures.

(Schnorr-based aggregate signature assumptions)

- \mathcal{G} - cyclic group with generator g , $|\mathcal{G}| = p$ prime.
- *Secrets* $\text{secret}_1, \dots, \text{secret}_n$ corresponding to public keys $\text{pub}_1 = g^{\text{secret}_1}, \dots, \text{pub}_n = g^{\text{secret}_n}$.
- The secrets secret_i are assumed to have distinct owners.
- Communication between key owners happens over open network.
- Malicious signers may craft $(\text{secret}_i, \text{pub}_i)$ to attack the system.

Secr.: $\text{secret}_1, \dots, \text{secret}_n$, pub. keys: $\text{pub}_1 = g^{\text{secret}_1}, \dots, \text{pub}_n = g^{\text{secret}_n}$.

(Zilliqa non-PBFT Schnorr multi-signature: preparation (once))

- *Select a special node called an aggregator.*
 - *Each signer: one-time send public key $\text{pub}_i \stackrel{?}{=} g^{\text{secret}_i}$.*
 - *Aggregator: send back challenge message to be signed.*
 - *Each signer: send signed challenge back.*
- The preparation simply ensures each signer owns secret_i from which pub_i is computed.
 - If preparation skipped: malicious signer can spoof public key by

$$\text{pub}_i \cdot \prod_{j \in \mathcal{V}} (\text{pub}_j)^{-1}$$

allowing him to single-handedly fake the aggregate signature for himself and the victim nodes in the set \mathcal{V} .

Secr.: $\text{secret}_1, \dots, \text{secret}_n$, pub. keys: $\text{pub}_1 = g^{\text{secret}_1}, \dots, \text{pub}_n = g^{\text{secret}_n}$.

(Zilliqa ~~non~~-PBFT Schnorr aggregate signature: signing)

- *Each signer: choose random nonce_i , compute $q_i = g^{\text{nonce}_i}$.*
- *Each signer: send q_i to aggregator. Let \mathcal{A} : set of healthy nodes.*
- *Aggregator: compute $\text{Pub} = \prod_{i \in \mathcal{A}} \text{pub}_i = \text{pub}_1 \cdot \dots \cdot \text{pub}_n$.*
- *Aggregator: compute $Q = \prod_{i \in \mathcal{A}} q_i$.*
- *Aggregator: compute $\text{challenge} = H(Q, \text{Pub}, \text{digest})$, H - hash f -n.*
- *Aggregator: send $\text{challenge}, \text{Pub}, \text{digest}$ to signers. Bad net: reset.*
- *Each signer: verify $\text{challenge} = H(Q, \text{Pub}, \text{digest})$.*
- *Each signer: compute $\text{solution}_i = \text{nonce}_i - \text{challenge} \cdot \text{secret}_i$.*
- *Each signer: send solution_i to aggregator. Bad net: reset.*
- *Aggregator: compute $\text{solution} = \sum_i \text{solution}_i$.*
- *Aggregator: final signature: $(\text{challenge}, \text{solution}), \mathcal{A}$.*
- To make algorithm fault tolerant: add highlighted steps.
- Requires black-listing bad actors from second net transaction on.



Secr.: $\text{secret}_1, \dots, \text{secret}_n$, pub. keys: $\text{pub}_1 = g^{\text{secret}_1}, \dots, \text{pub}_n = g^{\text{secret}_n}$.

(Zilliqa non-PBFT Schnorr aggregate signature: verification)

- Given: aggregate public key: Pub, message: digest, aggregate signature: (challenge, solution).

$$\text{Pub} \stackrel{?}{=} \prod_i \text{pub}_i$$

$$\text{challenge} \stackrel{?}{=} H\left(\prod_i g^{\text{nonce}_i}, \text{Pub}, \text{digest}\right) = H\left(g^{\sum_i \text{nonce}_i}, \text{Pub}, \text{digest}\right)$$

$$\text{solution} \stackrel{?}{=} \sum_i (\text{nonce}_i - \text{challenge} \cdot \text{secret}_i)$$

- Compute

$$\begin{aligned} X = g^{\text{solution}} \text{Pub}^{\text{challenge}} &\stackrel{?}{=} g^{\text{solution}} \left(\prod_i \text{pub}_i \right)^{\text{challenge}} = g^{\text{solution}} \left(\prod_i g^{\text{secret}_i} \right)^{\text{challenge}} \\ &= g^{\text{solution}} \prod_i g^{\text{secret}_i \cdot \text{challenge}} = g^{\sum_i (\text{nonce}_i - \text{secret}_i \cdot \text{challenge})} g^{\sum_i \text{secret}_i \cdot \text{challenge}} \\ &= g^{\sum_i (\text{nonce}_i - \text{secret}_i \cdot \text{challenge}) + \sum_i \text{secret}_i \cdot \text{challenge}} = g^{\sum_i \text{nonce}_i}. \end{aligned}$$

- If $H(X, \text{Pub}, \text{digest}) = \text{challenge}$ signature - valid (otherwise invalid).

- We present the aggregate signature scheme from [1, MPSW18].
- Different from Zilliqa in one crypto step (makes it more secure).
- The scheme does not specify communication protocol.
- For signing, we propose to combine [1, MPSW18] with the signing protocol of Zilliqa.
- For initial setup and general outline of the networking we present our own setup that leverages the foundation chain.

Secr.: $\text{secret}_1, \dots, \text{secret}_n$, pub. keys: $\text{pub}_1 = g^{\text{secret}_1}, \dots, \text{pub}_n = g^{\text{secret}_n}$.

[1, MPSW18] aggregate signature: signing)

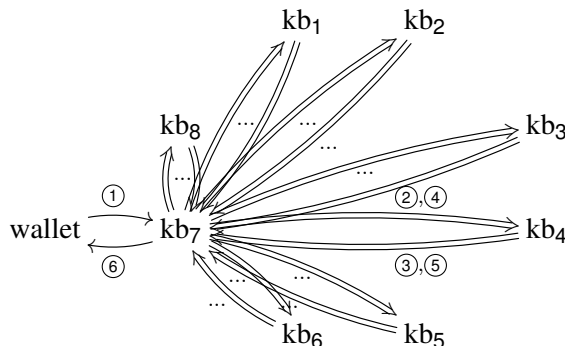
- *Aggregator: send protocol start to signers.*
- *Each signer: Choose random nonce_i , compute $q_i = g^{\text{nonce}_i}$.*
- *Each signer: send q_i to aggregator. Let \mathcal{A} : set of healthy nodes.*
- *Aggregator: compute $a_i = H(\mathcal{A}, \text{pub}_i)$. Compute $\text{Pub} = \prod_{i \in \mathcal{A}} \text{pub}_i^{a_i}$.*
- *Aggregator: compute $Q = \prod_{i \in \mathcal{A}} q_i$.*
- *Aggregator: compute challenge = $H(Q, \text{Pub}, \text{digest})$.*
- *Aggregator: send challenge, Pub, digest to signers. Bad net: reset.*
- *Each signer: verify challenge = $H(Q, \text{Pub}, \text{digest})$. Bad: reset.*
- *Each signer: compute $\text{solution}_i = \text{nonce}_i - \text{challenge} \cdot \text{secret}_i$.*
- *Each signer: send solution_i to aggregator. Bad net: reset.*
- *Aggregator: compute $\text{solution} = \sum_i \text{solution}_i$.*
- *Aggregator: final signature: (challenge, solution), \mathcal{A} .*

Like Zilliqa except a_i 's: those give extra security (Wagner-alg. attack).



- Signature verification for [1] is similar to Zilliqa's and we omit it.

- Intended use case: aggregate signature for each transaction, completed within 0-5 seconds after initiation. Sample networking:



- ① Wallet sends transaction "A sends B amount, signed by A". Node kb₁ is designates itself as an aggregator.
- ② Aggregator starts protocol, requesting the q_i 's
- ③ Signers report their q_i 's, proving they are online.
- ④ Aggregator sends challenges out.
- ⑤ Signers reply with solutions to their challenges.
- ⑥ Aggregator reports signature to wallet.

Design goals for the Kanban protocol

The slide(s) list some of the design goals considered; will be updated.

- ① KB's inherit authority from the foundation blockchain (POW).
- ② Except state between blocks, Kanban network's state aims to be function of longest foundation chain.
- ③ Data not directly in foundation chain: store error check (hash) in the found. chain. If no hash collisions, consistent with preceding.
 - EVM state can be used for storage.
 - A heavy-weight storage engine (callable by EVM) could be designed to reduce EVM state.
- ④ Should Kanban networking be reverted (longest chain overtake), all transaction chains approved by Kanban that do not involve reverted coinbases/incentives should be recoverable.



Kanban-Fabcoin interface

- Each Kanban communicates with the fabcoin interface through a smart contract.
- Contracts may be user-defined.
- It is assumed that there is one or more “hard-coded” (“reserved”) contracts, one of which is designated the “default” contract.
- The non-default “hard-coded” contracts may be used for older versions of the default.
- “Hard-coded” contracts can simply be contract addresses.
- Unless stated otherwise, it is assumed each KB node communicates with the “default” contract.

Default smart contract interface data representation

- We specify messages sent from KB to the smart contract in the JSON format.
- Suppose a wallet wants to fetch info on the KB network. Then the wallet sends a JSON that may look like:

```
{ "request": "smartContract", "id": 1, "command": "KBInfo"
```



References I



Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille.

Simple schnorr multi-signatures with applications to bitcoin.

Cryptology ePrint Archive, Report 2018/068, 2018.

<https://eprint.iacr.org/2018/068>.