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* $\mathrm{secret} \in \{0, 1, \dots, p-1\}$.
- *Compute the public key* $\mathrm{pub} = g^{\mathrm{secret}}$.

(Generalized DSA: sign message)

1. *Given: a number* digest *(message), f-n:* $\mathrm{toNumber} : \mathcal{G} \to \mathbb{Z}$.
2. *Choose at random* $\mathrm{nonce} \in \{0, \dots, p-1\}$.
3. *Compute* $\mathrm{challenge} = \mathrm{toNumber}(g^{\mathrm{nonce}}) \mod p$.
4. *Compute* $\mathrm{solution} = \frac{\mathrm{digest} + \mathrm{challenge} \cdot \mathrm{secret}}{\mathrm{nonce}} \mod p$.
5. $(\mathrm{challenge}, \mathrm{solution})$ *- final signature.*

- Public key pub - element of $\mathcal{G}$ but secret key secret is a number!
- $\mathrm{toNumber}$ must be close to one-to-one: diff. in's $\Rightarrow$ diff. out's.
- Usually: $\mathrm{digest} = H(M)$, 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): $r = \mathrm{challenge}$, $s = \mathrm{solution}$.

(Generalized DSA: signing, key generation summary)

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

(Generalized DSA: verify signature)

1. $a = \frac{1}{\mathrm{solution}} \stackrel{?}{=} \frac{1}{\frac{\mathrm{digest}+\mathrm{challenge}\cdot\mathrm{secret}}{\mathrm{nonce}}} = \frac{\mathrm{nonce}}{\mathrm{digest}+\mathrm{challenge}\cdot\mathrm{secret}} \mod p$.
2. $u_1 = a\cdot\mathrm{digest} \stackrel{?}{=} \frac{\mathrm{nonce}\cdot\mathrm{digest}}{\mathrm{digest}+\mathrm{challenge}\cdot\mathrm{secret}} \mod p$.
3. $u_2 = a\cdot\mathrm{challenge} \stackrel{?}{=} \frac{\mathrm{nonce}\cdot\mathrm{challenge}}{\mathrm{digest}+\mathrm{challenge}\cdot\mathrm{secret}} \mod p$.
4. $$\begin{aligned} X = g^{u_1}\mathrm{pub}^{u_2} &\stackrel{?}{=} g^{\frac{\mathrm{nonce}\cdot\mathrm{digest}}{\mathrm{digest}+\mathrm{challenge}\cdot\mathrm{secret}}} \cdot \left(g^{\mathrm{secret}}\right)^{\frac{\mathrm{nonce}\cdot\mathrm{challenge}}{\mathrm{digest}+\mathrm{challenge}\cdot\mathrm{secret}}} \\ &= g^{\frac{\mathrm{nonce}\cdot\mathrm{digest}}{\mathrm{digest}+\mathrm{challenge}\cdot\mathrm{secret}} + \frac{\mathrm{nonce}\cdot\mathrm{secret}\cdot\mathrm{challenge}}{\mathrm{digest}+\mathrm{challenge}\cdot\mathrm{secret}}} = g^{\frac{\mathrm{nonce}\cdot\cancel{(\mathrm{digest}+\mathrm{challenge}\cdot\mathrm{secret})}}{\cancel{\mathrm{digest}+\mathrm{challenge}\cdot\mathrm{secret}}}} = g^{\mathrm{nonce}} \end{aligned}$$
5. *If* $\mathrm{toNumber}(X) \stackrel{?}{=} \mathrm{challenge}$*: 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:* $\mathrm{pub} = g^{\mathrm{secret}}$, *$g$- generator of $\mathcal{G}$, $|\mathcal{G}| = p$ is prime.*

(Schnorr signature: signing)

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

(Schnorr signature: verification)

- $a = \mathrm{challenge} \cdot \mathrm{pub}^{\mathrm{digest}} \stackrel{?}{=} g^{\mathrm{nonce}} \cdot \left(g^{\mathrm{secret}}\right)^{\mathrm{digest}} = g^{\mathrm{nonce}+\mathrm{secret}\cdot\mathrm{digest}}$.
- $b = g^{\mathrm{solution}} \stackrel{?}{=} g^{\mathrm{nonce}+\mathrm{secret}\cdot\mathrm{digest}}$.
- *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* $\mathrm{secret}_1, \dots, \mathrm{secret}_n$ *corresponding to public keys* $\mathrm{pub}_1 = g^{\mathrm{secret}_1}, \dots, \mathrm{pub}_n = g^{\mathrm{secret}_n}$.

- The secrets secret_i are assumed to have distinct owners.
- Communication between key owners happens over open network.
- Malicious signers may craft $(\mathrm{secret}_i, \mathrm{pub}_i)$ to attack the system.

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

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

- *Select a special node called an aggregator.*
- *Each signer: one-time send public key* $\mathrm{pub}_i \stackrel{?}{=} g^{\mathrm{secret}_i}$.
- *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

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

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

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

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

- *Each signer: choose random* nonce_i, *compute* $q_i = g^{\mathrm{nonce}_i}$.
- *Each signer: send* q_i *to aggregator. Let* $\mathcal{A}$: *set of healthy nodes.*
- *Aggregator: compute* $\mathrm{Pub} = \prod_{i\in\mathcal{A}} \mathrm{pub}_i = \mathrm{pub}_1 \cdot \dots \cdot \mathrm{pub}_n$.
- *Aggregator: compute* $Q = \prod_{i\in\mathcal{A}} q_i$.
- *Aggregator: compute* $\mathrm{challenge} = H(Q, \mathrm{Pub}, \mathrm{digest})$, *H- hash f-n.*
- *Aggregator: send* $\mathrm{challenge}, \mathrm{Pub}, \mathrm{digest}$ *to signers. Bad net: reset.*
- *Each signer: verify* $\mathrm{challenge} = H(Q, \mathrm{Pub}, \mathrm{digest})$.
- *Each signer: compute* $\mathrm{solution}_i = \mathrm{nonce}_i - \mathrm{challenge} \cdot \mathrm{secret}_i$.
- *Each signer: send* $\mathrm{solution}_i$ *to aggregator. Bad net: reset.*
- *Aggregator: compute* $\mathrm{solution} = \sum_i \mathrm{solution}_i$.
- *Aggregator: final signature:* $(\mathrm{challenge}, \mathrm{solution}), \mathcal{A}$.

- To make algorithm fault tolerant: add highlighted steps.
- Requires black-listing bad actors from second net transaction on.

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

(Zilliqa non-PBFT Schnorr aggregate signature: verification)

- *Given: aggregate public key:* Pub, *message:* digest, *aggregate signature:* $(\mathrm{challenge}, \mathrm{solution})$.

$$\begin{aligned}
\mathrm{Pub} &\stackrel{?}{=} \textstyle\prod_i \mathrm{pub}_i \\
\mathrm{challenge} &\stackrel{?}{=} H\left(\textstyle\prod_i g^{\mathrm{nonce}_i}, \mathrm{Pub}, \mathrm{digest}\right) = H\left(g^{\sum_i \mathrm{nonce}_i}, \mathrm{Pub}, \mathrm{digest}\right) \\
\mathrm{solution} &\stackrel{?}{=} \textstyle\sum_i \left(\mathrm{nonce}_i - \mathrm{challenge} \cdot \mathrm{secret}_i\right)
\end{aligned}$$

- *Compute*

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

- *If* $H(X, \mathrm{Pub}, \mathrm{digest}) = \mathrm{challenge}$ *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.: $\mathrm{secret}_1, \ldots, \mathrm{secret}_n$, pub. keys: $\mathrm{pub}_1 = g^{\mathrm{secret}_1}, \ldots, \mathrm{pub}_n = g^{\mathrm{secret}_n}$.

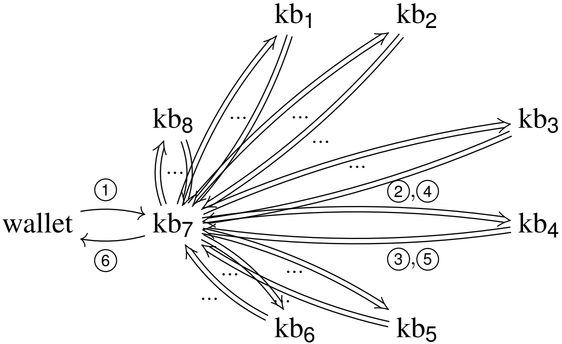
([1, MPSW18] aggregate signature: signing)

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

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 kb_1 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 udpated.

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.