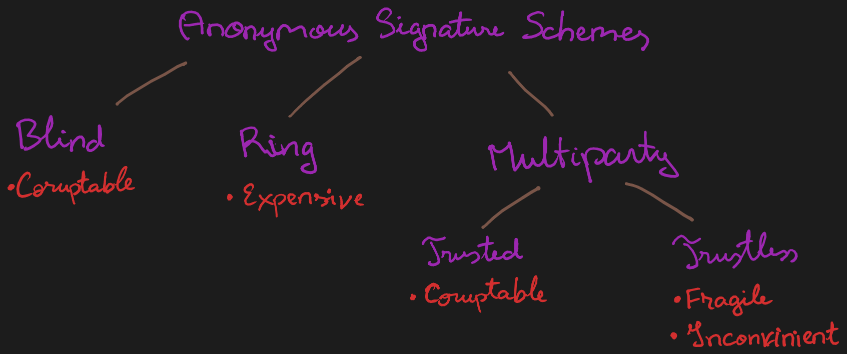
Julia for cryptography, security and voting

Over some long evenings after work, I entertained myself with designing a remote electronic voting system which I would find satisfactory for Westworld - possessing a strong software independence with strong privacy guarantees. In the process, I came up with trustless while robust and fairly convenient design based on pseudonym braiding, which I implemented in Julia for improving clarity and doing a sanity check. In this poster, I will briefly review cryptographic packages already available in Julia, ones which I implemented for key exchange, digital signatures, socket encryption, multiplexing and ledger distribution, and lastly the main ideas of PeaceVote package and current challenges.

Janis Erdmanis (GitHub: akels, Twitter: graphitewriter, janiserdmanis.org)

An idea worth exploring

Motivation: *The main challenge for the remote electronic voting system is to provide a public and trustless cryptographic proof that each vote is produced by a legitimate member who votes only once while still preserving strong privacy guarantees through distributed multiparty computations. Linkable ring signatures with TOR would be excellent but unfortunately are rather expensive computationally and in signature size. Alternatively, one can create an anonymous signature with a blind signature scheme, but it involves authority who has the ability to stuff ballot (no guarantees for legitimacy).*



Instead, I propose to use ballots for creation of a list of anonymous pseudonyms. Each voter participating in ballot secretly generates his pseudonym and puts that in the ballot box whose contents are afterwards published and mutual consent produced. This procedure (braiding) can be repeated with pseudonyms themselves, giving robustness and reasonable convenience as trustless ballots can be made small while still enlarging anonymity set with each ballot. The generated pseudonyms then can be used to sign votes and delivered to the voting box with TOR. To clarify and to check sanity, I chose to implement the protocol in Julia.

Initial Problem: *Julia missed libraries for making a key exchange, encrypting sockets and doing digital signatures for amateurs.*

Julia and basic cryptography: *Nettle for hashes and encryption, ECC for DSA with elliptic curves. Recently also AMCL for asymmetric cryptography.*

Filling gaps

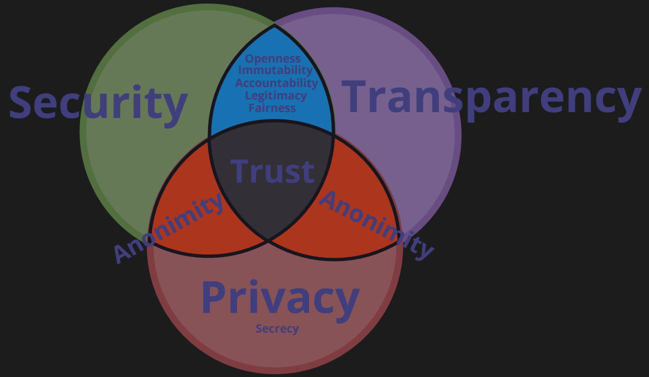
***CryptoGroups** – wraps ECC and implements RSA prime groups (G) under the same API.*

***CryptoSignatures** – currently implements DSA with API which allows swapping generators easily. Assumes data (for example hash) to be `BigInt`.*

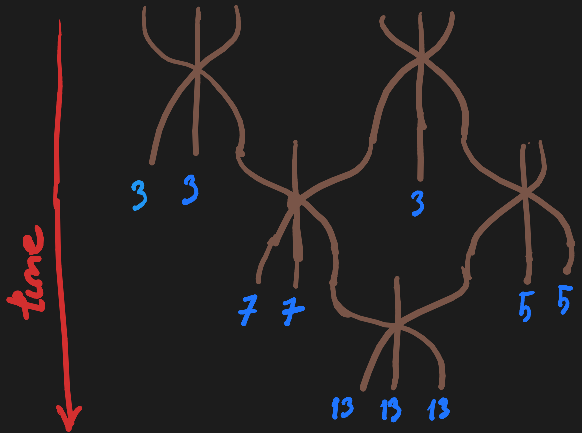
***DiffieHellman** – implements DH key exchange algorithm which allows to swap G and do single side or both side authentication.*

***SecureIO** – allows constructing encrypted `SecureSocket` which extends `write` and `read` methods for composability.*

Transparency and Privacy



Solution: Instead of signing votes, we shall sign pseudonyms. In this way each pseudonym would link to the next one in a braid like structure:



Trustless ballots: A trustless voting system can be constructed on an absolute consent basis. A gatekeeper gives access to multiple participants at the same remote mix session (which participants authenticate) and after that receives all messages in randomized order. The ballot is successful if each participant signs the received messages collectively (on the basis of having some UUID) and delivers the signature to the gatekeeper. This protocol is implemented in SynchronicBallot.

Challenge: This system, however, is extremeellly fragile and inconvinient as everyone must be trusted and everyone must be online at the same time.

The lines represent pseudonyms, and the nodes are braids where new pseudonyms through trustless ballot system are created. The numbers represent the anonymity set size of each pseudonym. In other words, the number of participants on which behalf the pseudonym owner can issue a signature.

As anonymity set of each pseudonym grows exponentially, the number of braidings for each participant is of the order of log(N). All data necessary to validate pseudonyms are of the order N log(N).

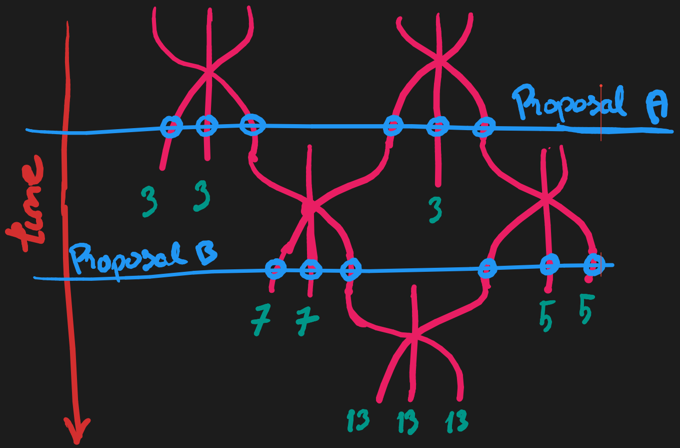
PeaceVote

Essential Definition

Guardian is maintainer, defender and approver of new members to its self-governing community (Deme). Its main task is to run services such as braider and vote recorder and prevent them from DDOS and other attacks. By design, the guardian is powerless to deanonymize pseudonyms even if services record additional data unless collaboration with mix is happening.

BraidChain is a time-ordered list of braids and pseudonym registration certificates issued by the guardian (or delegated authority) which are cryptographically linked with inputs the contract signing pseudonyms and outputs the pseudonyms on the braid.

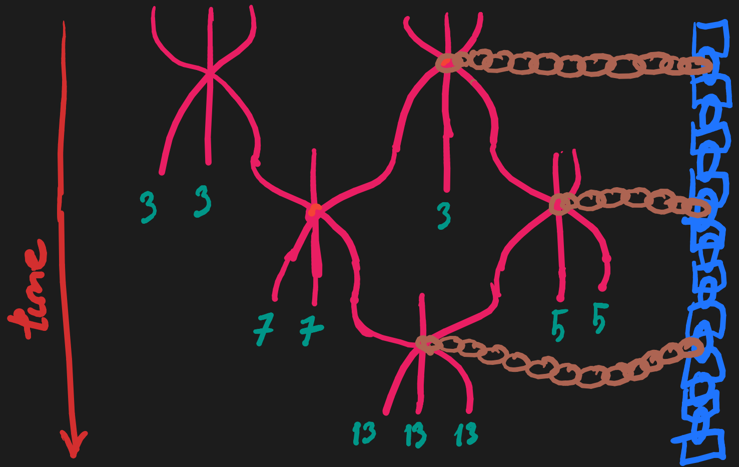
KeyChain is a list of time-stamped keys generated during braiding which allows using owned pseudonyms to issue signatures on behalf of the members of the corresponding Deme.



Proposing: The guardian specified authority (could be specific members of the Deme) issues a proposal with UUID and time-stamp which defines eligible pseudonyms as an intersection with BraidChain (see the figure on the left).

Voting: The voter gets the proposal and picks owned pseudonym key accordingly, fills, signs and delvers the vote anonymously.

BraidChain supported ledgers



The braiding procedure already contains absolute consent from each participant of the braid. It is tempting to also add a ledger hash as part of consent, which then makes it tamper-resistant.

That gives a medium where proposals, votes can be stored and as well the elements of the BraidChain itself. Later one can be used to speed up validation of a long BraidChain by only verifying hashes and some N braid signatures from the bottom.

Future work and References

Current user API for PeaceVote

```
braidchain = BraidChain(braidchainconfig,deme)
system = BraidChainServer(braidchain,server)

### Testing the server ###

maintainer = Signer(deme,"maintainer")

for i in 1:3
    account = "account$i"
    keychain = KeyChain(deme,account)
    cert = Certificate(keychain.member.id,maintainer)
    @show record(braidchain,cert)
end

# Now let's test braiding

@sync for i in 1:3
    @async begin
        account = "account$i"
        keychain = KeyChain(deme,account)
        braid!(braidchain,keychain)
    end
end

# Proposing

keychain = KeyChain(deme,"account2")
proposal = Proposal("Found peace for a change?",["yes","no","maybe"])
cert = Certificate(proposal,keychain.member)
record(braidchain,cert)

sleep(1)

loadedledger = load(braidchain)
messages = attest(loadedledger,braidchain.deme.notary)
index = proposals(messages)[1]
proposal = messages[index]

# Now we can vote

for i in 1:3
    account = "account$i"
    keychain = KeyChain(deme,account)

    option = Vote(index,rand(1:length(proposal.document.options)))
    cert = Certificate(option,braidchain,keychain)
    record(braidchain,cert)
end

# Now let's count
sleep(1)

@show tally = count(index,braidchain)
```

Future Work

Move pseudonym anonymization (`BraidChain`, `KeyChain`, `braid!`) outside the PeaceVote package and stabilize API.

Formalize data on the BraidChain. Make its state commands explicit, such as adding registrator, adding member, rebraiding. Put nonessential data such as proposals and votes in supporting ledger.

Multiple daemons are necessary to run PeaceVote protocol (about 7). Each of them might break and may need to be restarted, could produce a log, and has its inputs and outputs. Explore creating a `Service{T}` type to formalize and improve modularity.

Wrap TOR or other traffic anonymizer to prevent linkability between pseudonym and IP.

Create an interactive terinal CLI for early testing and make it easier for potential guardian to set up the server. And...

References

Janis Erdmanis (2020). PeaceVote. Draft.

Rivest, R. L. (2008) On the notion of 'software independence' in voting systems.

Juvonen, Atte. (2019). A framework for comparing the security of voting schemes.

Also peacefounder.org and PeaceFounder GitHub organization