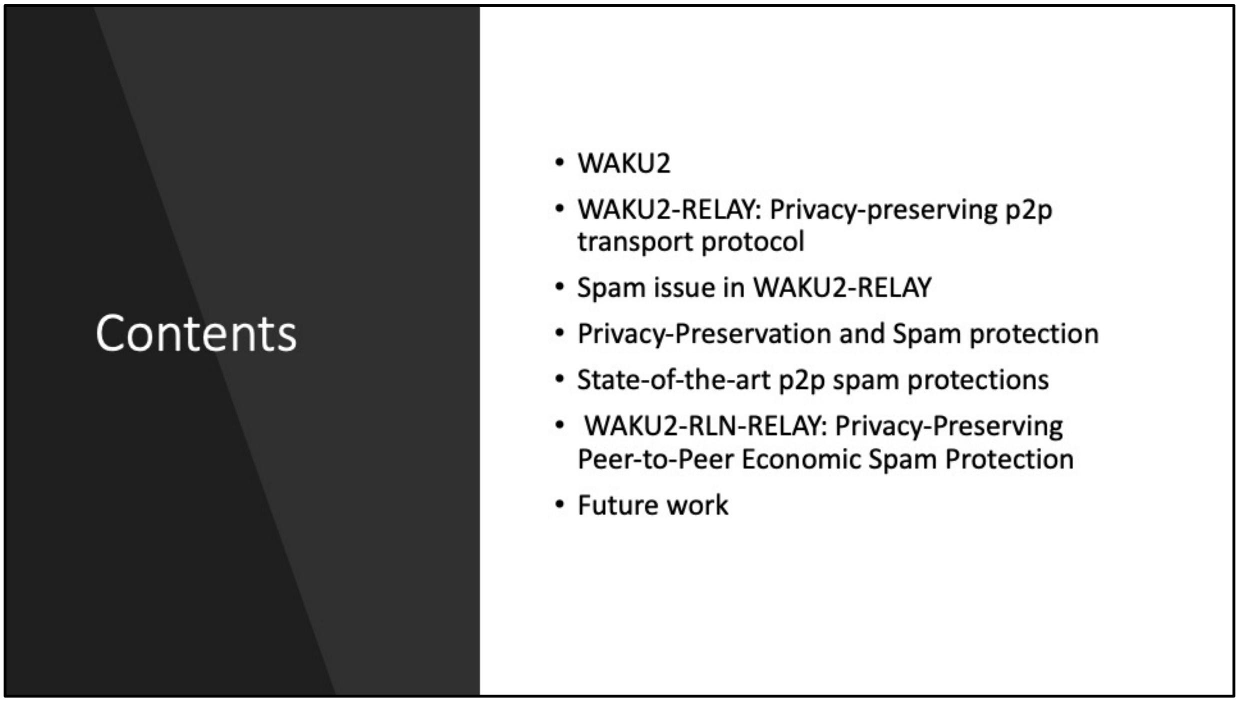
Hi,
Today, I am going to present waku-rln-relay which is a privacy preserving economic spam protection that suits private and anonymous messaging systems
This is a joint work by me, Oskar Thoren, Barry Whitehat, Wei Jie Koh, Onur Kilic, and Kobi Gurkan
The paper is also available in the link below



The agenda is as illustrated
I’ll start by discussing WAKU2 and WAKU-RELAY, I will review the spam issue and the related studies, then I’ll explain waku-rln-relay architecture and how it excels its counterparts. In the end, I will shed light on the future work.

WAKU2 [1]

- A family of modular, privacy-preserving peer-to-peer (p2p) protocols for private, secure, censorship resistant communication
- Suitable for resource restricted devices e.g., mobile phones
- WAKU2 protocols include:
  - **WAKU2-RELAY: privacy-preserving transport**
  - WAKU2-STORE: historical message storage
  - WAKU2-FILTER: light version of WAKU2-RELAY for bandwidth limited devices
  - **WAKU2-RLN-RELAY: spam-protected version of WAKU2-RELAY**
  - And many more …
- For the full list of RFCs is available in rfc.vac.dev

[1] https://rfc.vac.dev/spec/10/

Waku2 is a stack of peer-to-peer (p2p) protocols that enable anonymous and privacy-preserving communication.
It is designed to be able to run in resource-restricted environments.

The focus of today’s talk is WAKU-RELAY and WAKU-RLN-RELAY which are the transport layers of Waku

WAKU2-RELAY [1]

- Publisher-Subscriber Model
- Gossip-based Routing (extension of libp2p GossipSub-v1.1 [2])
- Anonymous and Privacy-Preserving

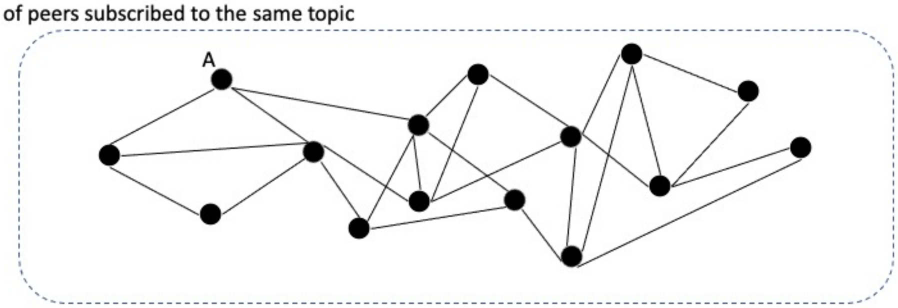
[1] https://rfc.vac.dev/spec/11/
[2] https://github.com/libp2p/specs/tree/master/pubsub/gossipsub

Let me start by WAKU2-RELAY transport protocol
It follows publisher-subscriber messaging model
Implements gossip-based routing protocol
Is a minor extension of libp2p GossipSub protocol
Its end goal is to enable an anonymous and privacy preserving p2p network layer

WAKU2-RELAY

- **Peers subscribed to the same topic form a mesh**

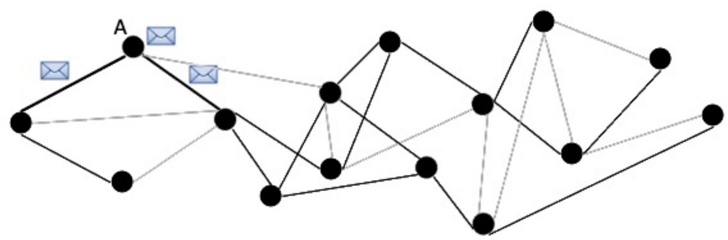
Mesh of peers subscribed to the same topic



Peers in wakurelay congregate around topics they are interested in and can send messages to that topic or receive messages within that topic

WAKU2-RELAY

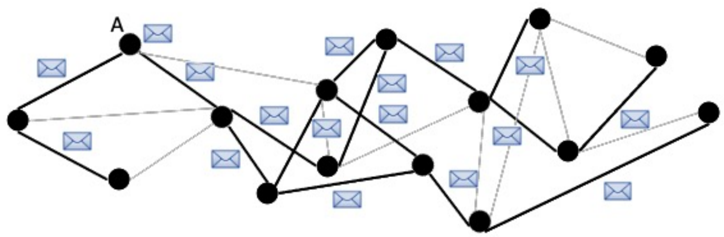
- Peers subscribed to the same topic form a mesh
- Peers route messages by sending them to a subset of their connections



To publish a message, the author forwards its message to a subset of neighbors.

WAKU2-RELAY

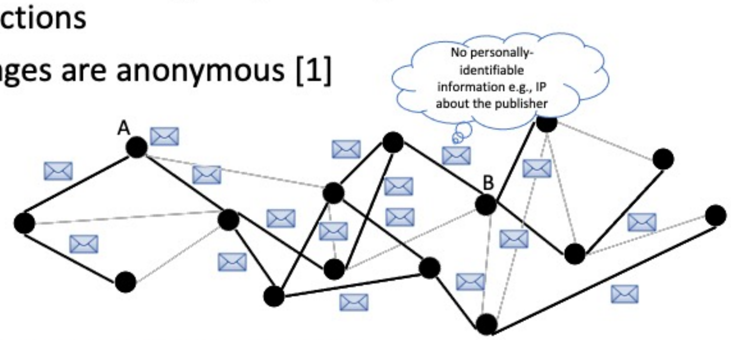
- Peers subscribed to the same topic form a mesh
- Peers route messages by sending them to a subset of their connections



The neighbors proceed similarly till the message gets propagated in the network of the subscribed peers.

WAKU2-RELAY

- Peers subscribed to the same topic form a mesh
- Peers route messages by sending them to a subset of their connections
- Messages are anonymous [1]

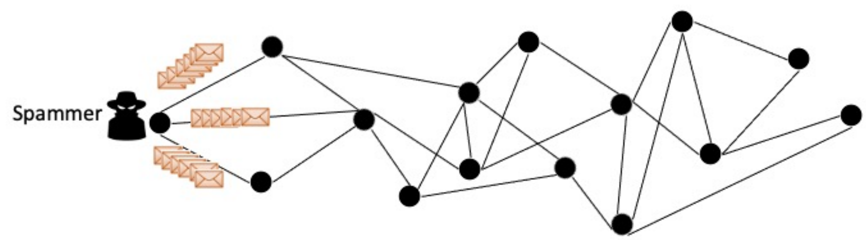


[1] https://rfc.vac.dev/spec/11/#security-analysis

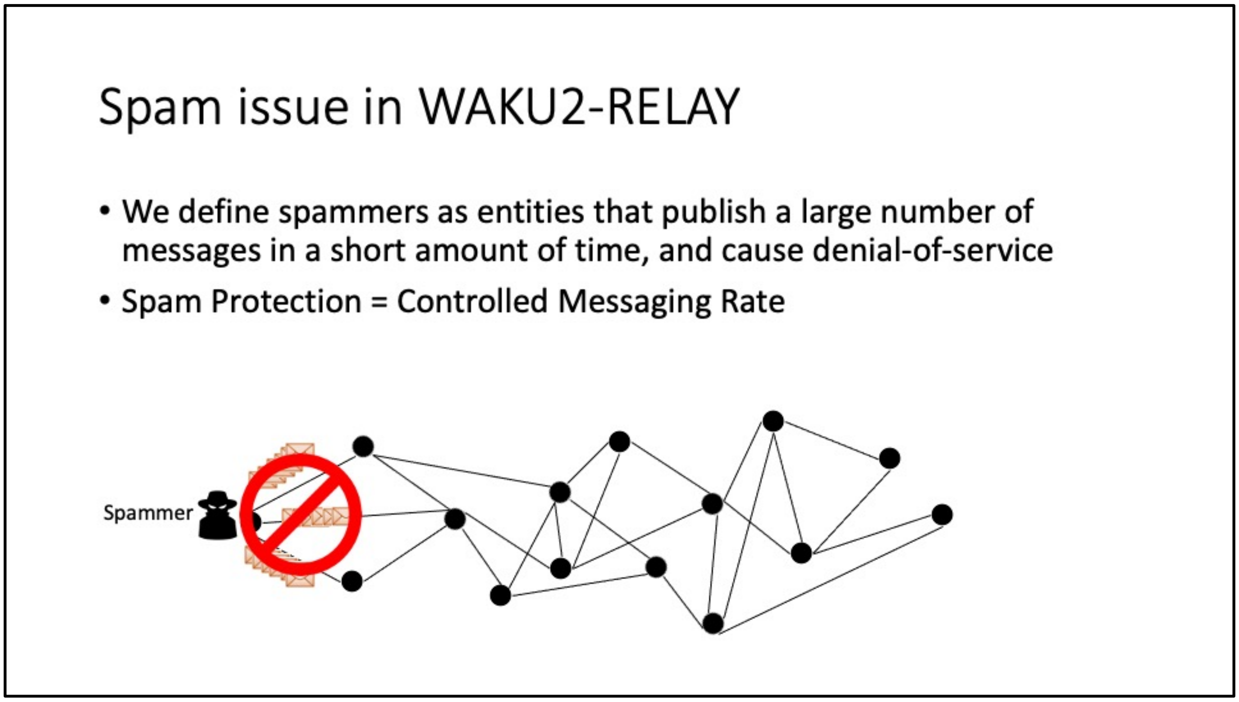
Messages are anonymous, i.e., there is no personally identifiable information like IP address attached to them, thus it is not feasible to identify the message origin.
For more detailed security analysis see the link below

Spam issue in WAKU2-RELAY

- **We define spammers as entities that publish a large number of messages in a short amount of time, and cause denial-of-service**



Waku Relay, as an open messaging network, is prone to spam messages and spammers;
we define spammers as entities that publish a large number of messages in a short amount of time, and cause Denial of Service attack.

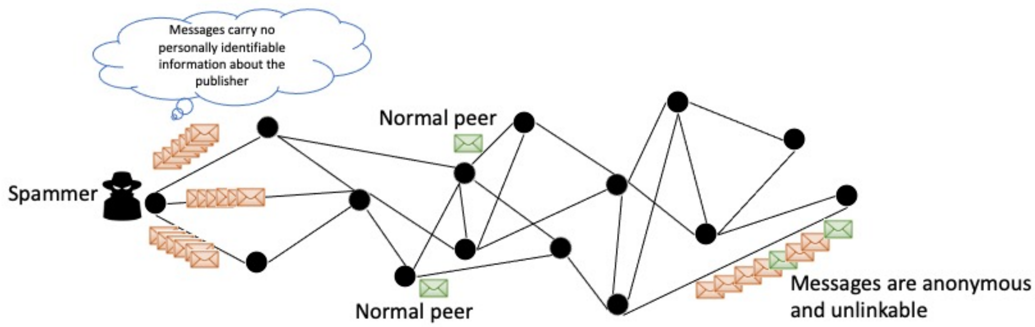


Waku Relay, as an open messaging network, is prone to spam messages and spammers;
we define spammers as entities that publish a large number of messages in a short amount of time, and cause Denial of Service attack.

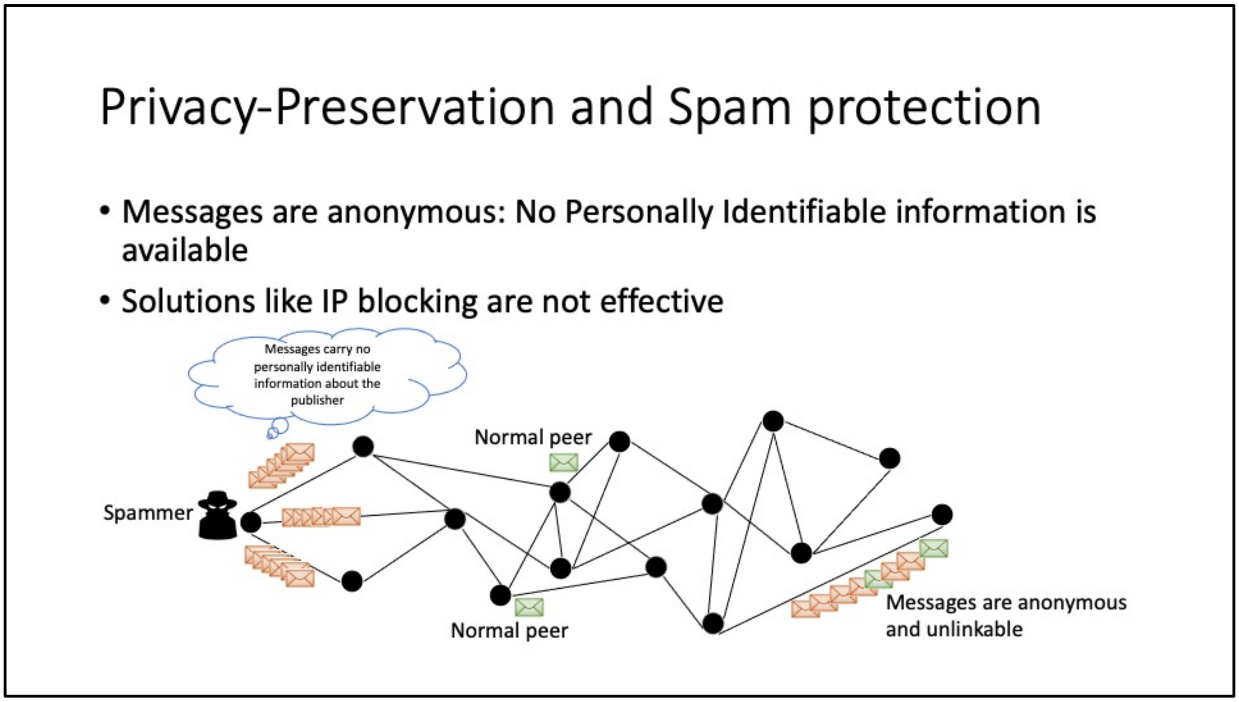
With that definition, Spammers can be controlled if we can control their messaging rate

Privacy-Preservation and Spam protection

- Messages are anonymous: No Personally Identifiable information is available



But there is an issue, messages are anonymous (no information like IP is available) and not linkable to their authors
A routing peer wont be able to tell apart spam messages from non-spam



But there is an issue, messages are anonymous (no information like IP is available) and not linkable to their authors

A routing peer wont be able to tell apart spam messages from non-spam

Thus solutions like IP blocking are not effective

State-of-the-art p2p spam protections

- Proof-of-work [1] deployed by Whisper [2]
  - Computationally expensive
  - Not suitable for network of heterogeneous peers with limited resources
- Peer Scoring [3] in libp2p
  - Local to each peer
  - No global identification of spammer
  - Subject to inexpensive attacks using bots
  - Prone to censorship

[1] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Annual 456 international cryptology conference. Springer, 1992.
[2] https://eips.ethereum.org/eips/eip-627.
[3] https://github.com/libp2p/specs/blob/master/pubsub/gossipsub/gossipsub-v1.1.mdpeerscoring.

The state-of-the-art p2p spam protection techniques are Proof of Work (POW) deployed by Whisper
and Peer scoring method adopted by libp2p

The PoW is not computationally efficient and does not fit resource limited devices (limited resources won't be able to participate and benefit from the messaging system)

In peer scoring
The solution is local since each peer monitors and scores its direct connections and drops the connections with low scores.
However, a spammer would be still able to continue its activity by switching its connection from one peer to another as soon its score drops a threshold.
Peer scoring is also subject to inexpensive attacks where the spammer can deploy millions of bots to send bulk messages.
It is also prone to censorship

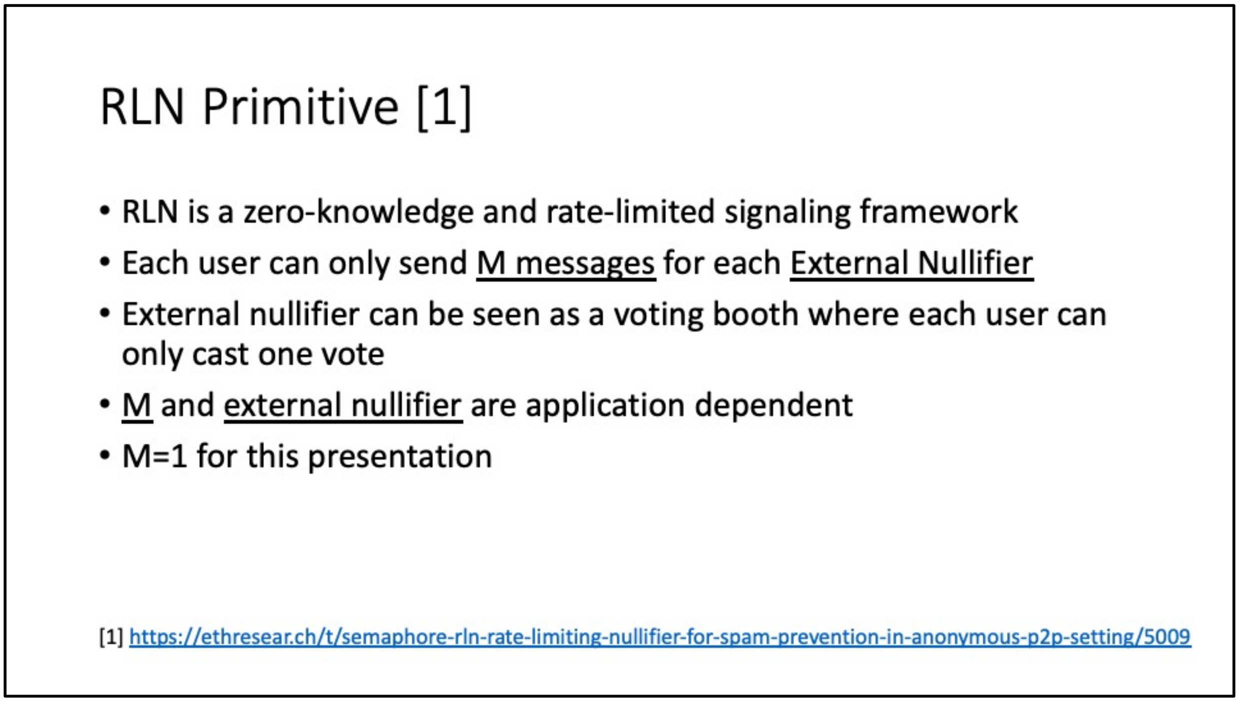
WAKU2-RLN-RELAY [1]

WAKU2-RLN-RELAY = WAKU2-RELAY + Rate Limiting Nullifiers (RLN)

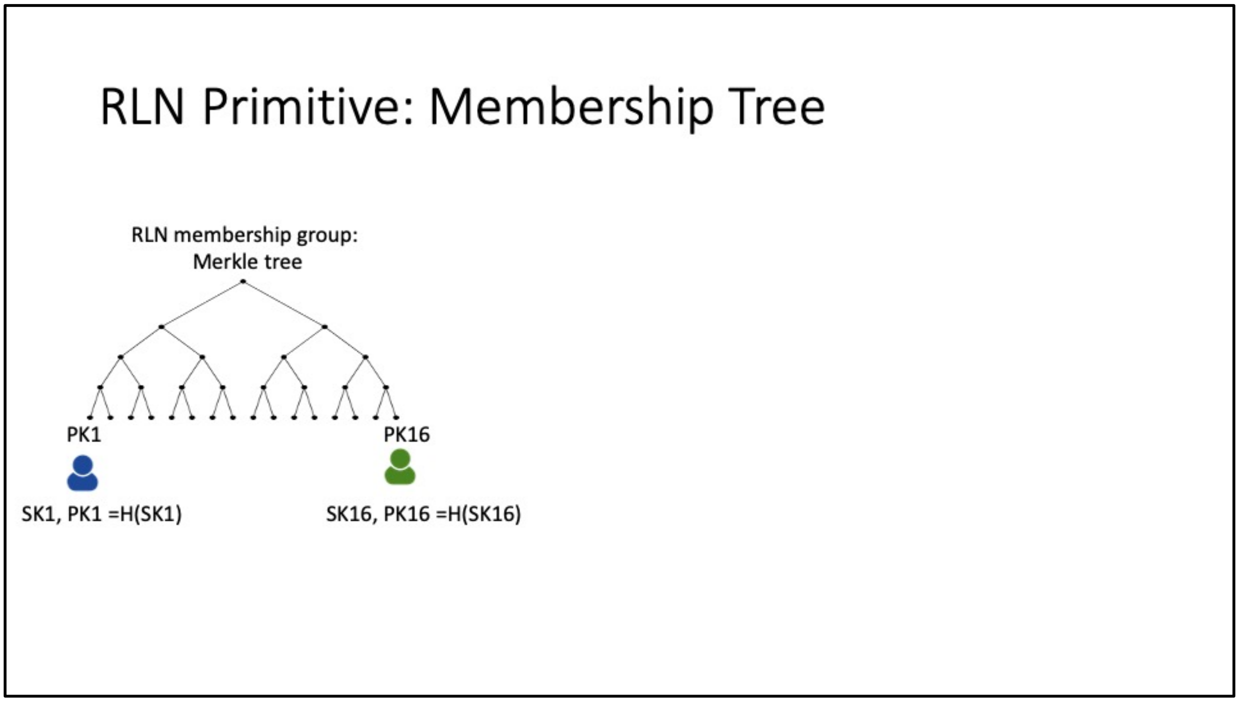
- P2p solution
- Global spam protection
- Privacy preserving
- Efficient
- Economic incentives

[1] https://rfc.vac.dev/spec/17/

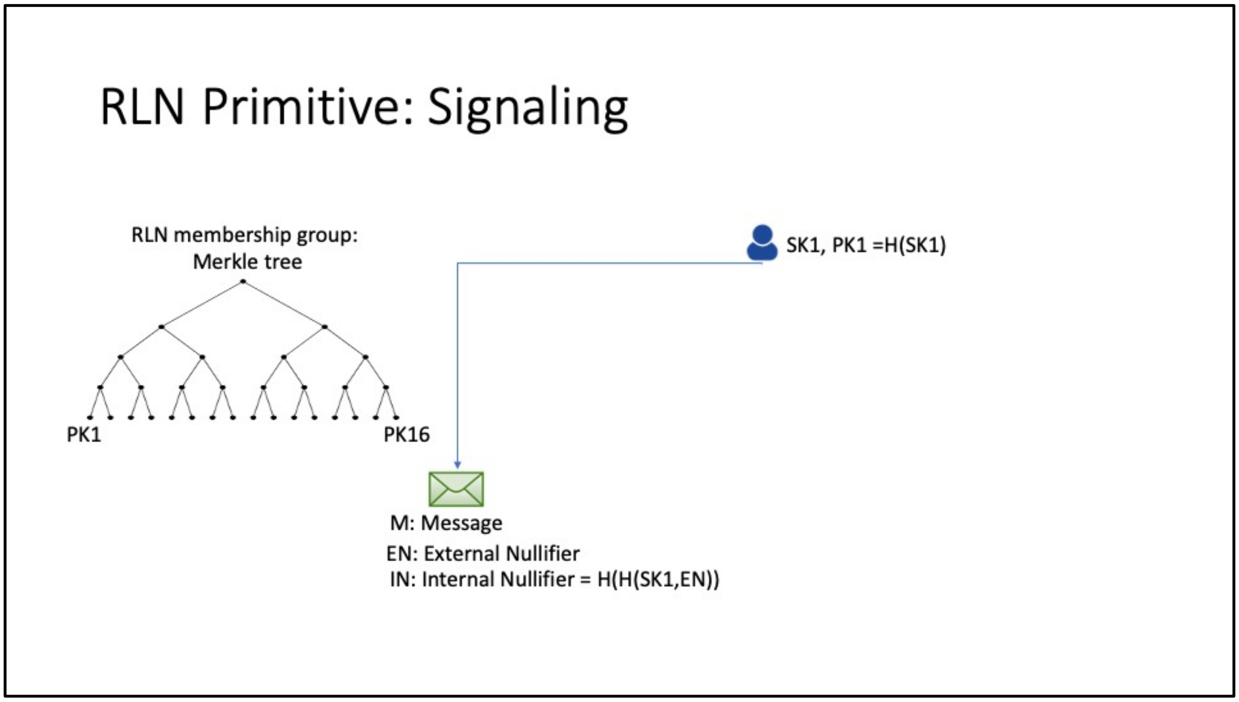
In WAKU-RLN-RELAY we cope with the aforementioned issues
it uses waku-relay (that I previously explained) as an anonymous transport protocol
and combines it with rate limiting nullifier construct to control the messaging rate
The end result has a p2p structure, with no central entity involved
it **allows global identificatio**n and removal of spammers
it is **privacy-preserving** since there is no need to personally identifiable information e.g., email address, IP, etc. about peers to be able to identify and block spammers
It is **efficient** i.e., with no unreasonable computational, storage, memory, and bandwidth requirement, as such, it fits the network of heterogeneous peers with limited resources.
It has **economic-incentives, i.e.,** there is a financial punishment for the spammers and a financial reward for those who catch spammers.



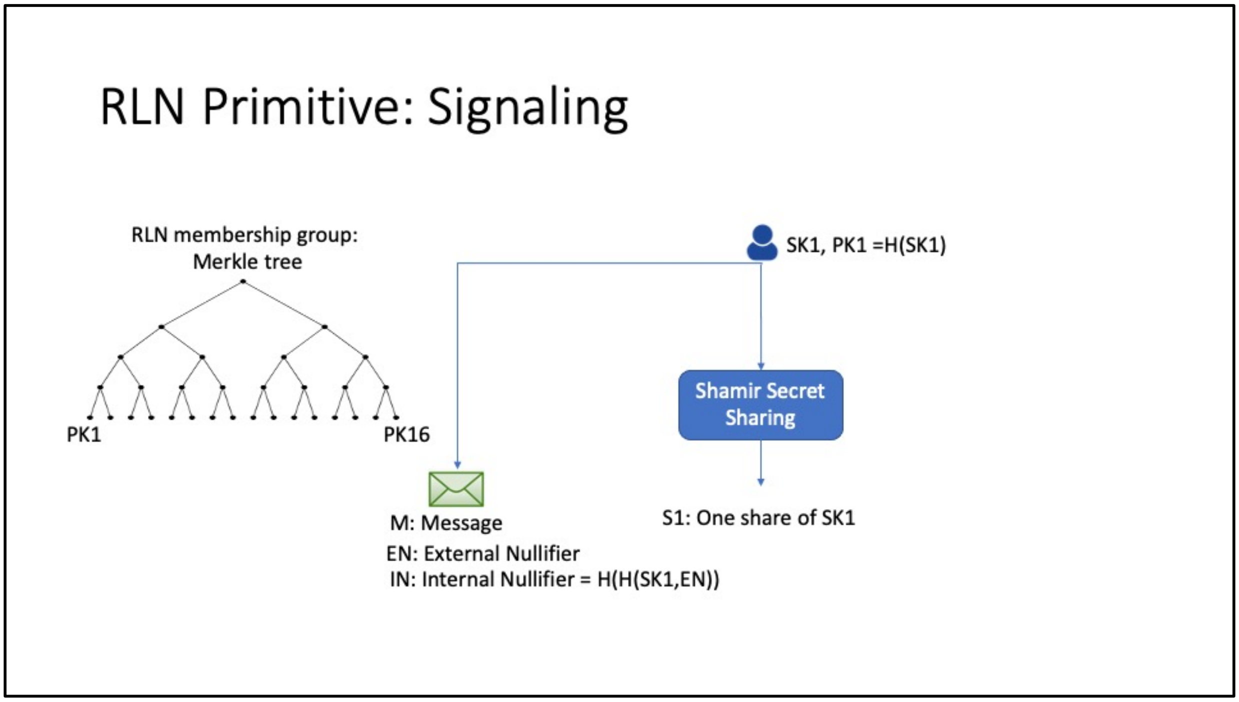
- Lets begin with the RLN construct
- it is a zero-knowledge and rate-limited signaling framework
- It allows a set of users to broadcast arbitrary signals (where signal is any value like a string, vote, etc.) while proving they are among a group of authorized users without disclosing their identities
- The idea is that each user can only send M messages for a specific external nullifier. External nullifier can be seen as a voting booth where each user can only cast one vote
- For the rest of this presentation we consider M as 1



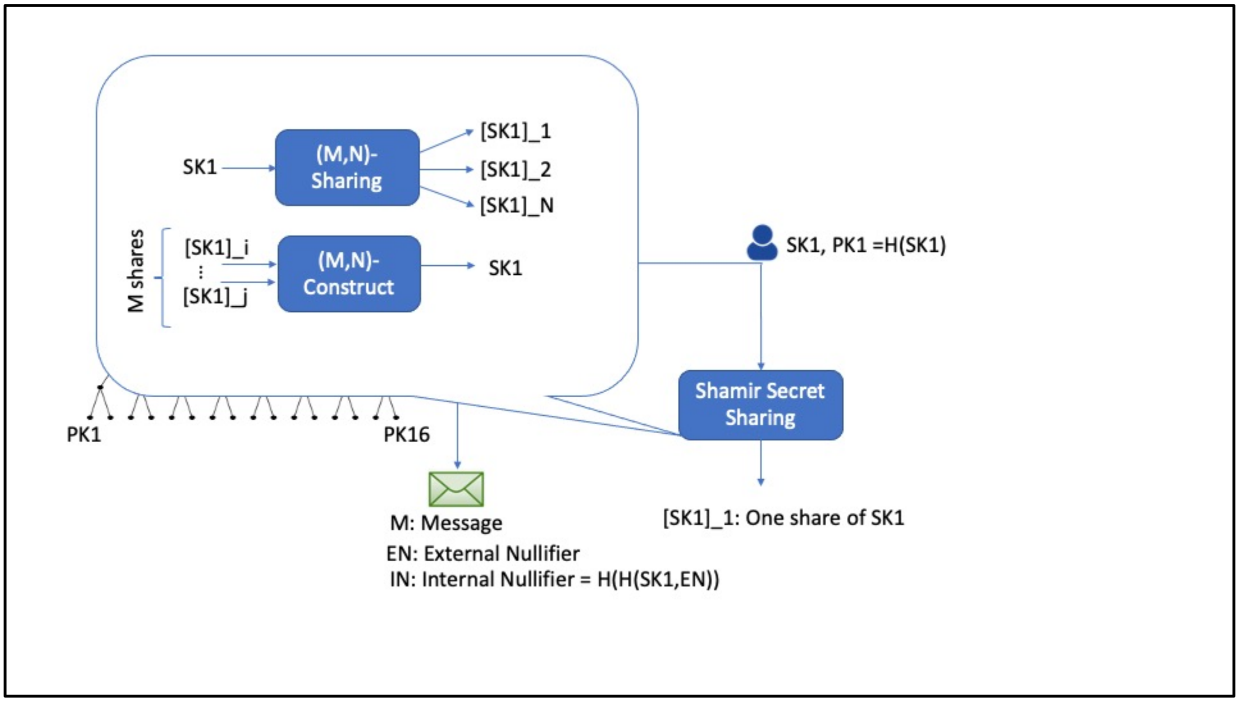
- It consists of a merkle tree that represents a group of authorized users, each user has a pk registered in this tree, the corresponding SK is only known to the user



- For the signaling/publishing:
- The user specifies an external nullifier as well as an internal nullifier which is derived from the SK and the external nullifier (as you can see in the slide)

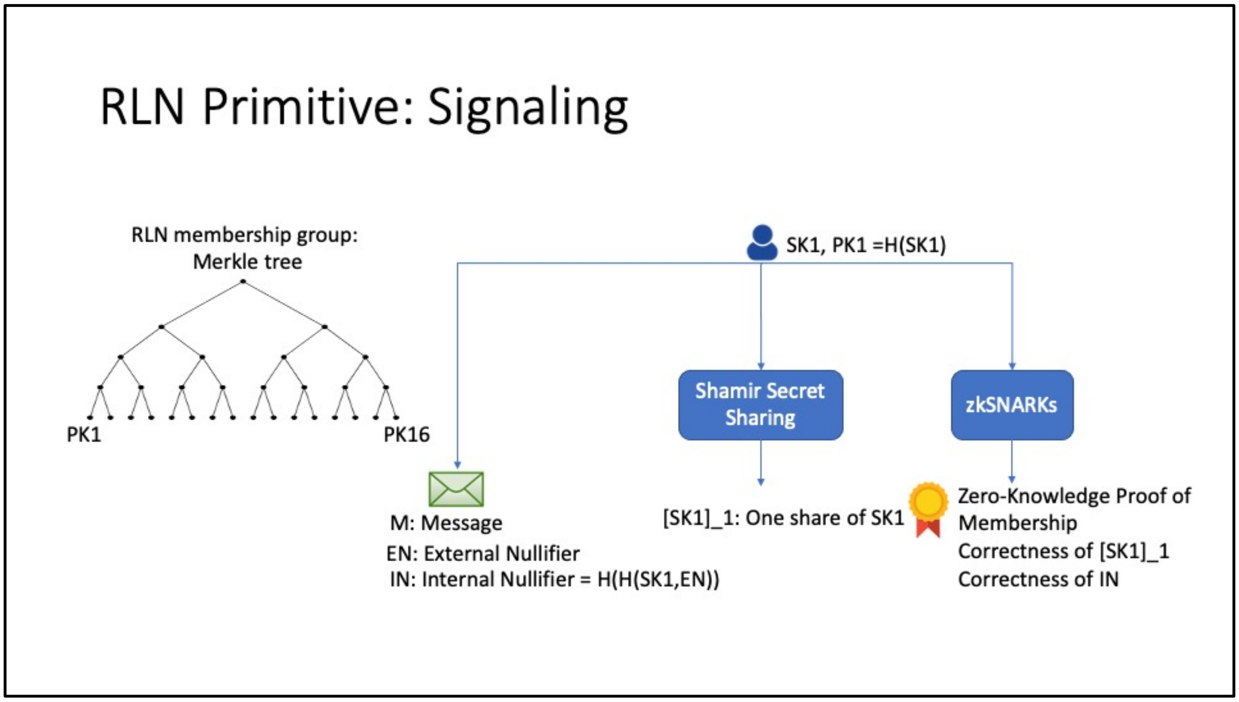


- The user also discloses a share of its secret key using shamir secret sharing scheme
- This share will be used to remove the user from the group in case of double signaling

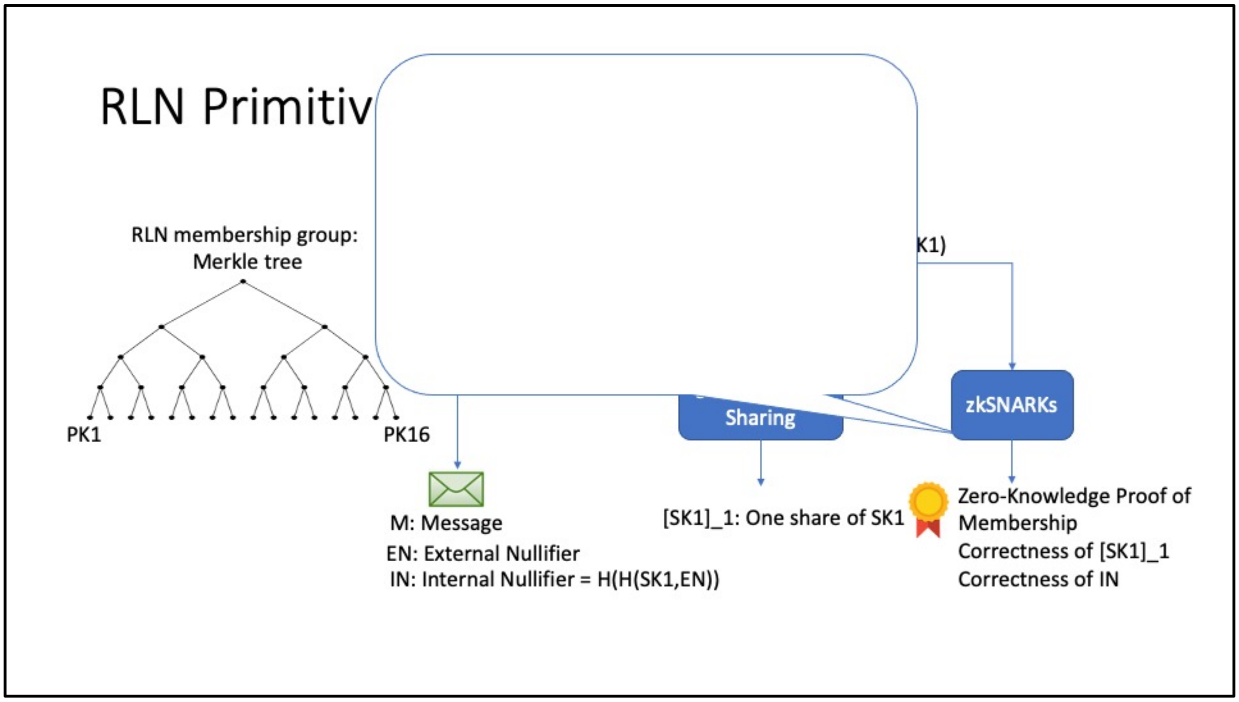


Shamir secret sharing (or a secret sharing scheme) allows to split a secret data sk into N pieces

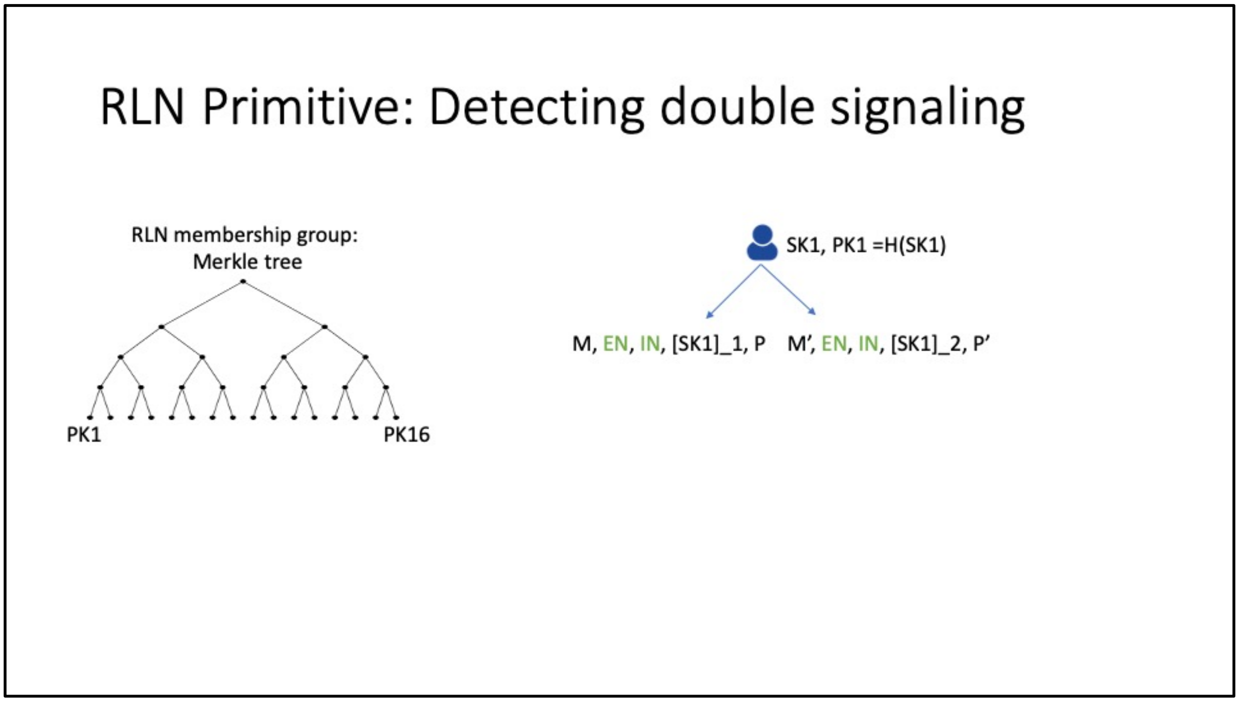
Construct the sk back by having a subset of M sk shares



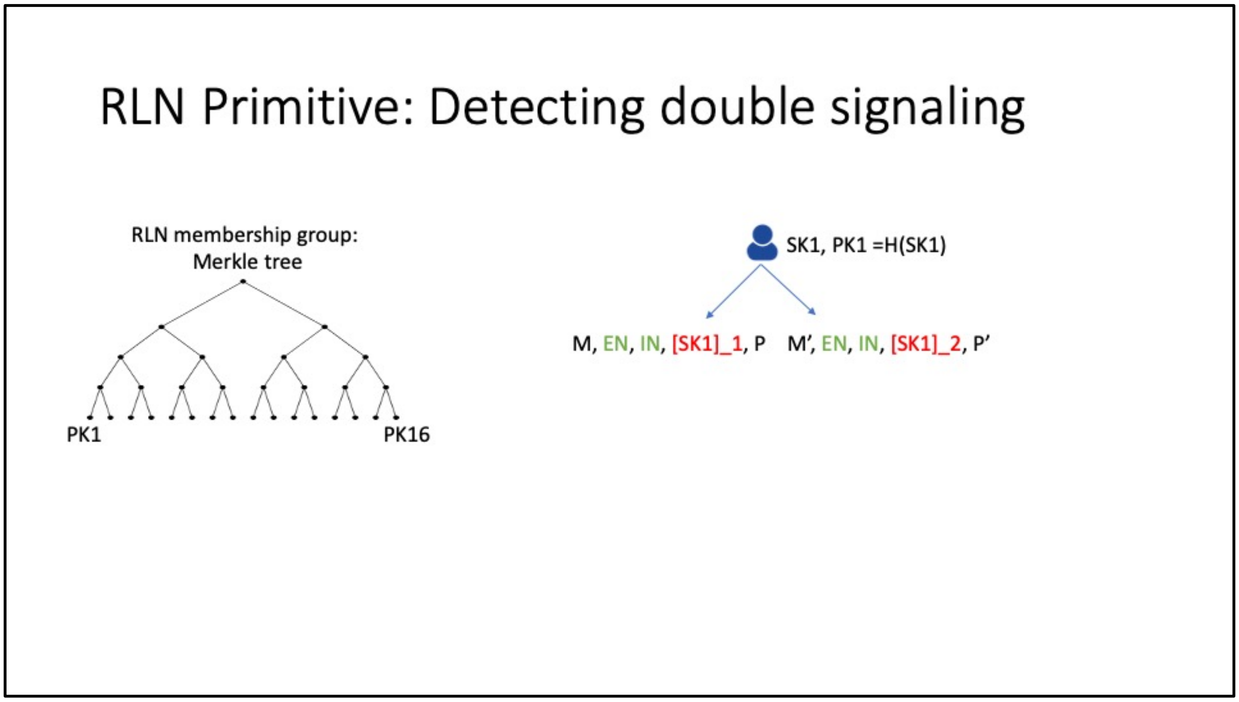
- Finally, the user proves in zero knowledge manner that:
- It is part of the group
- And that has computed the secret share and the Internal nullifier correctly



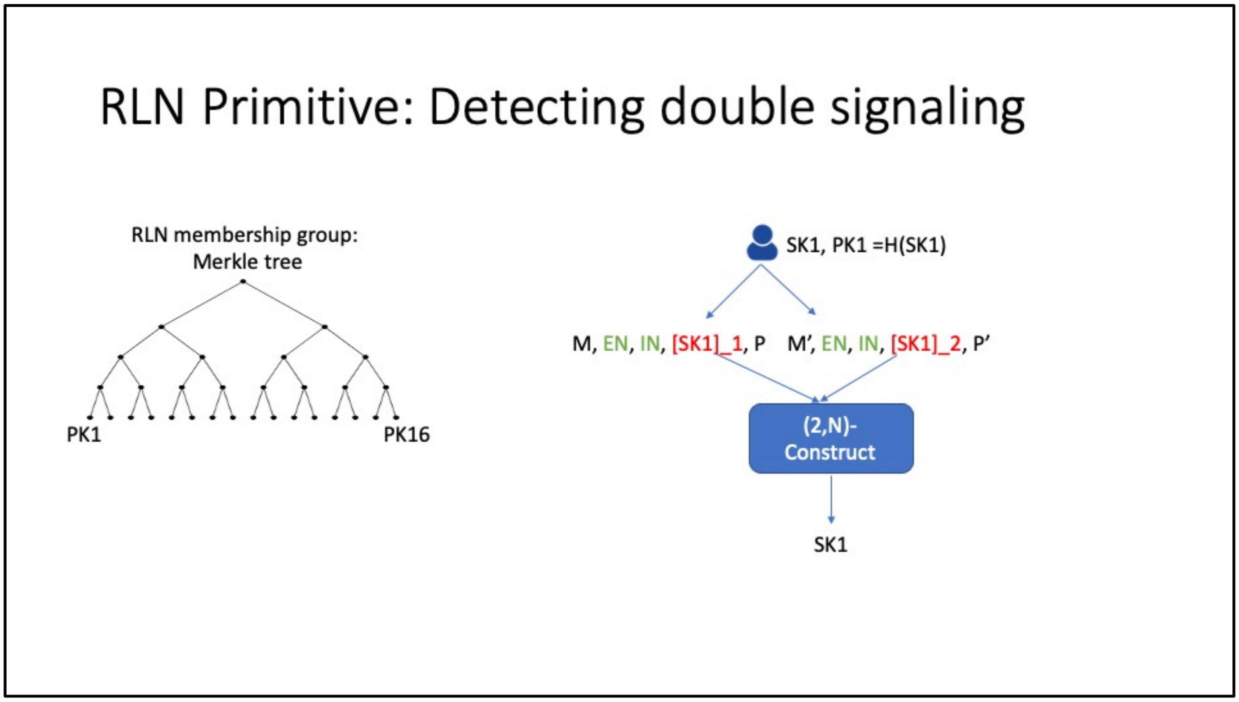
- Finally, the user proves in zero knowledge manner that:
- It is part of the group
- And that has computed the secret share and the Internal nullifier correctly



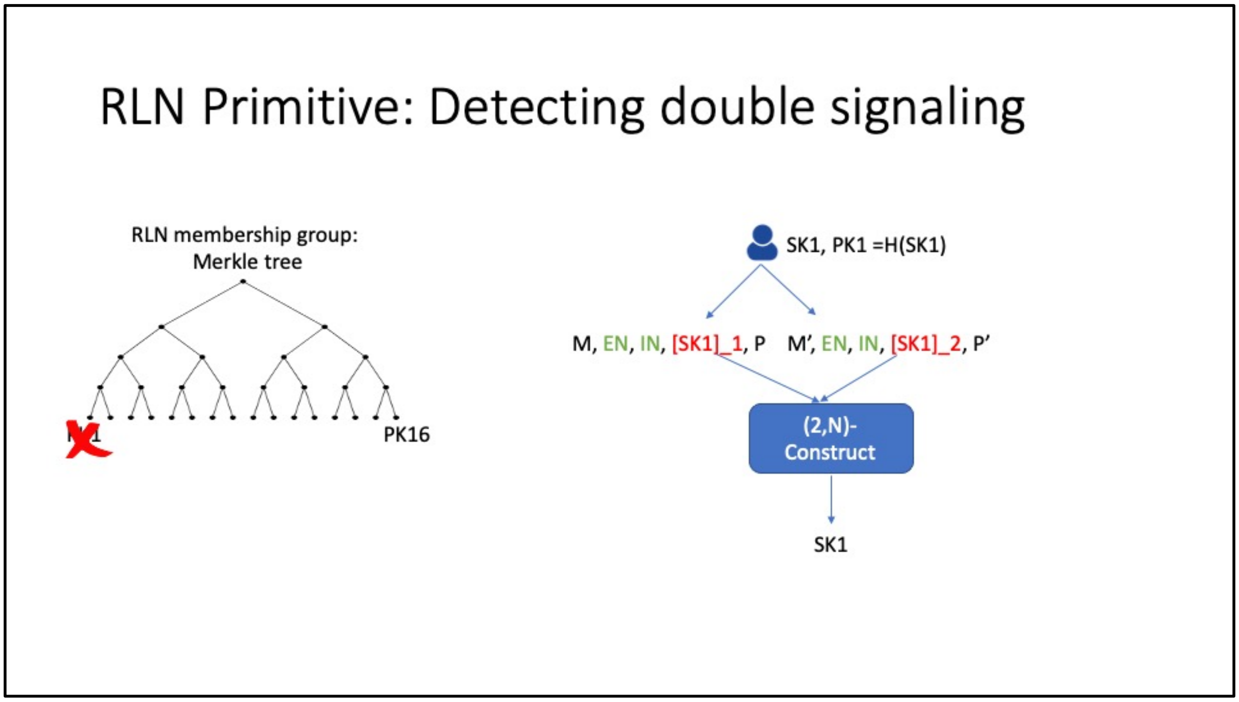
- If a user publishes more than 2 times for the same external nullifier, it will end up having two messages with the same external and internal nullifiers (remember those are deterministic values computed from SK and external nullifier)
- So the double signaling attempt can be detected



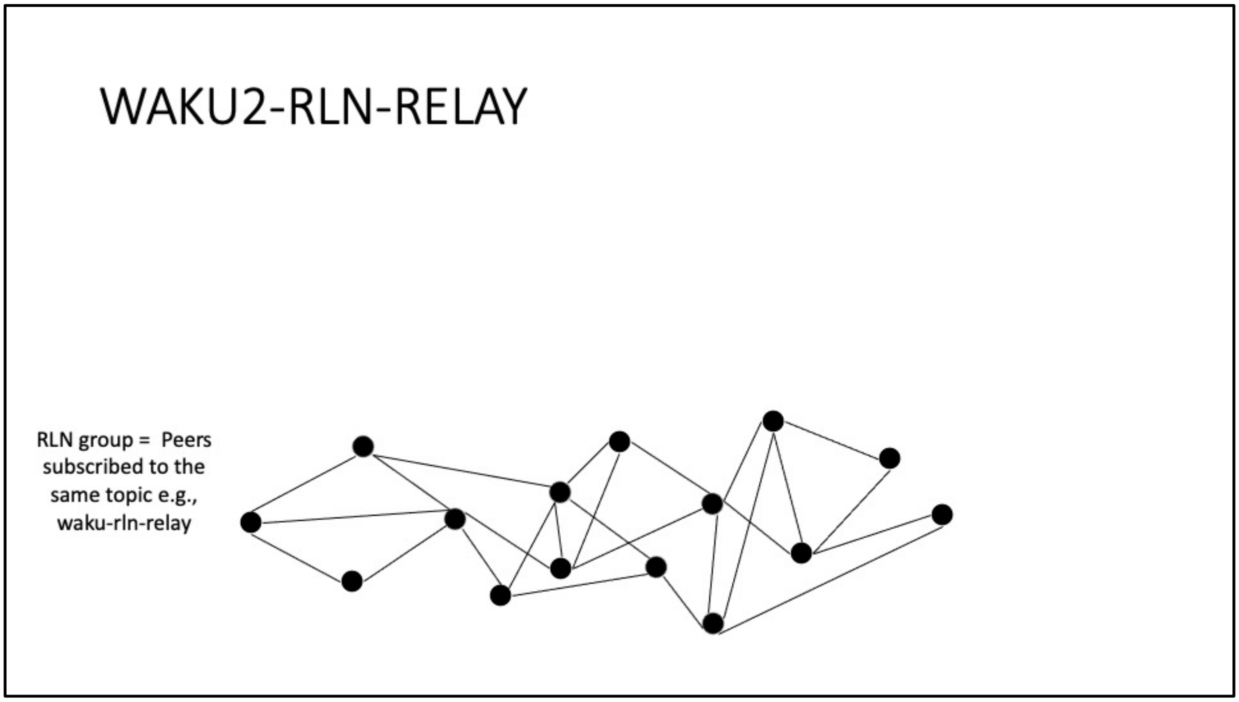
- Also, it will disclose 2 shares of its SK



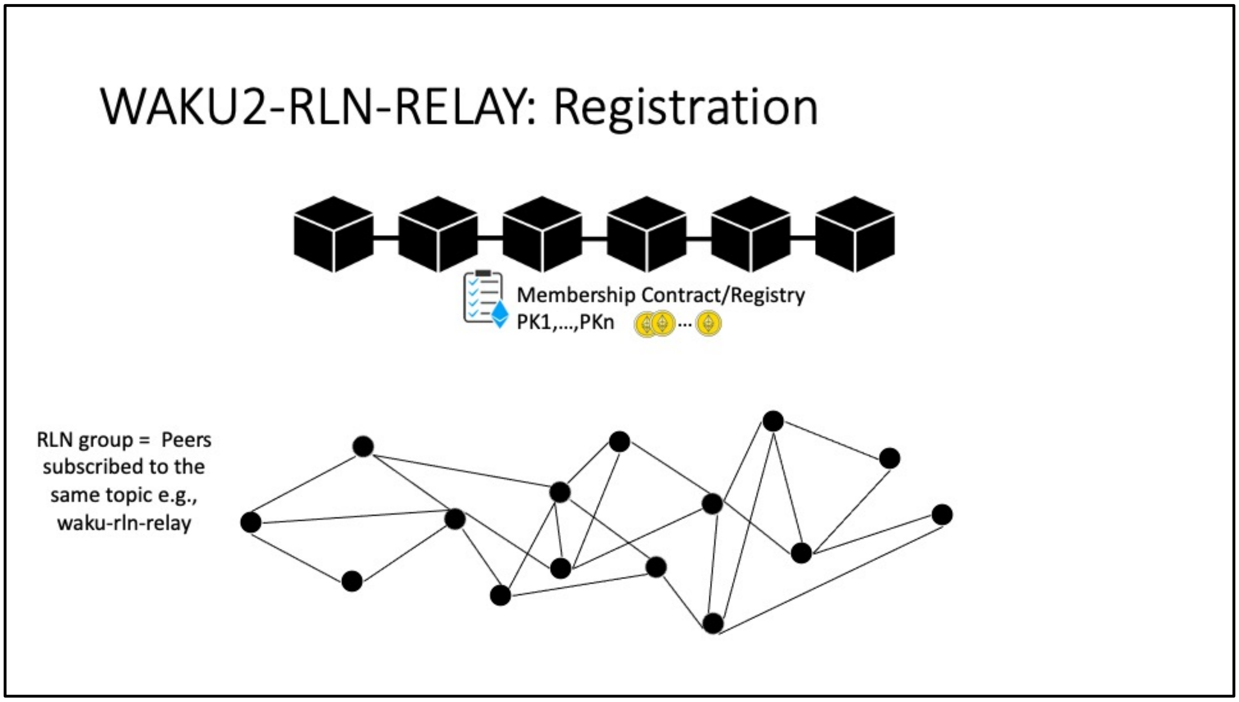
- using which the corresponding SK can be reconstructed (property of shamir secret sharing).



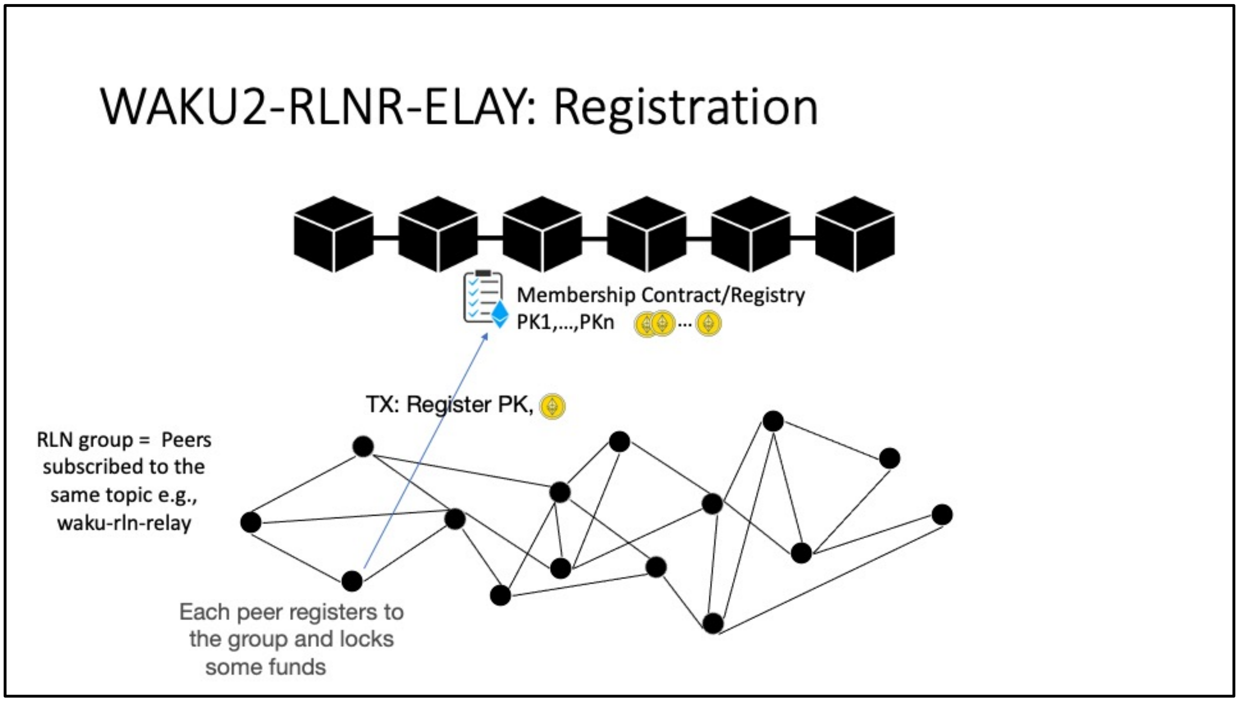
- And the user gets removed from the tree. Thus, it can no longer use that sk for messaging.



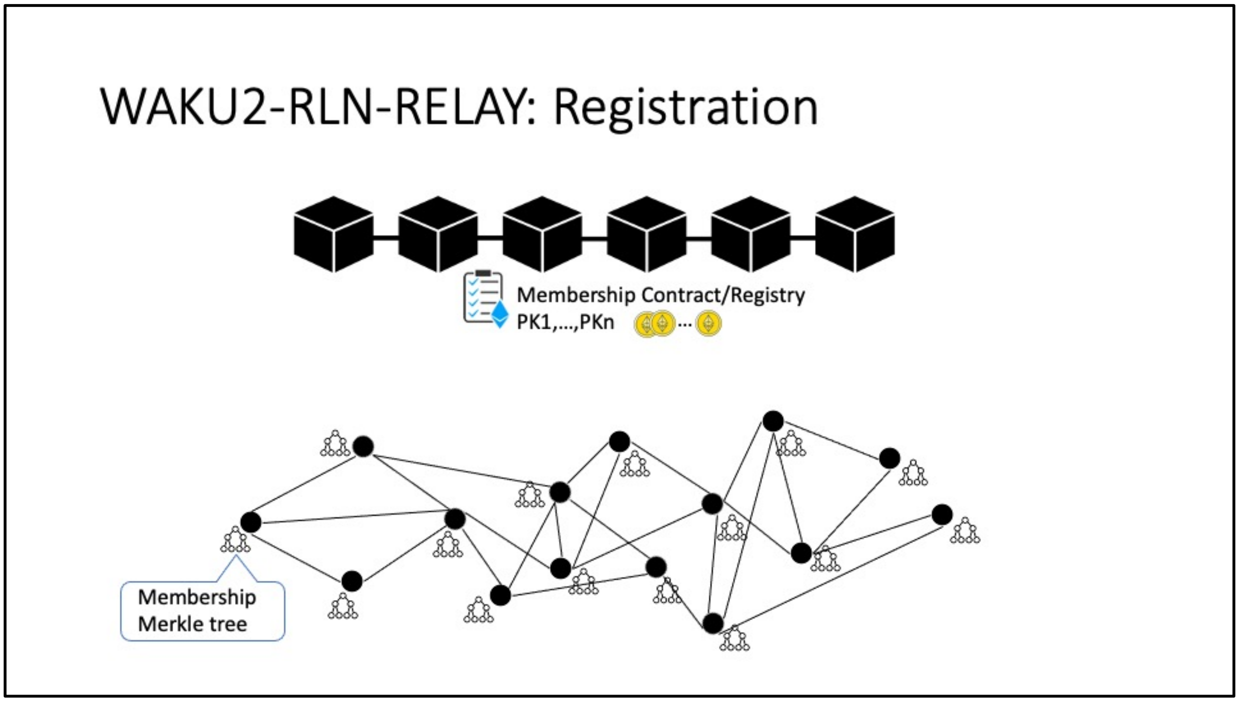
Now lets see the end to end integration of RLN into Waku-rln-relay transport protocol
HEre, the rln group consists of peers that belong to the same GossipSub layer (subscribed to the same topic)



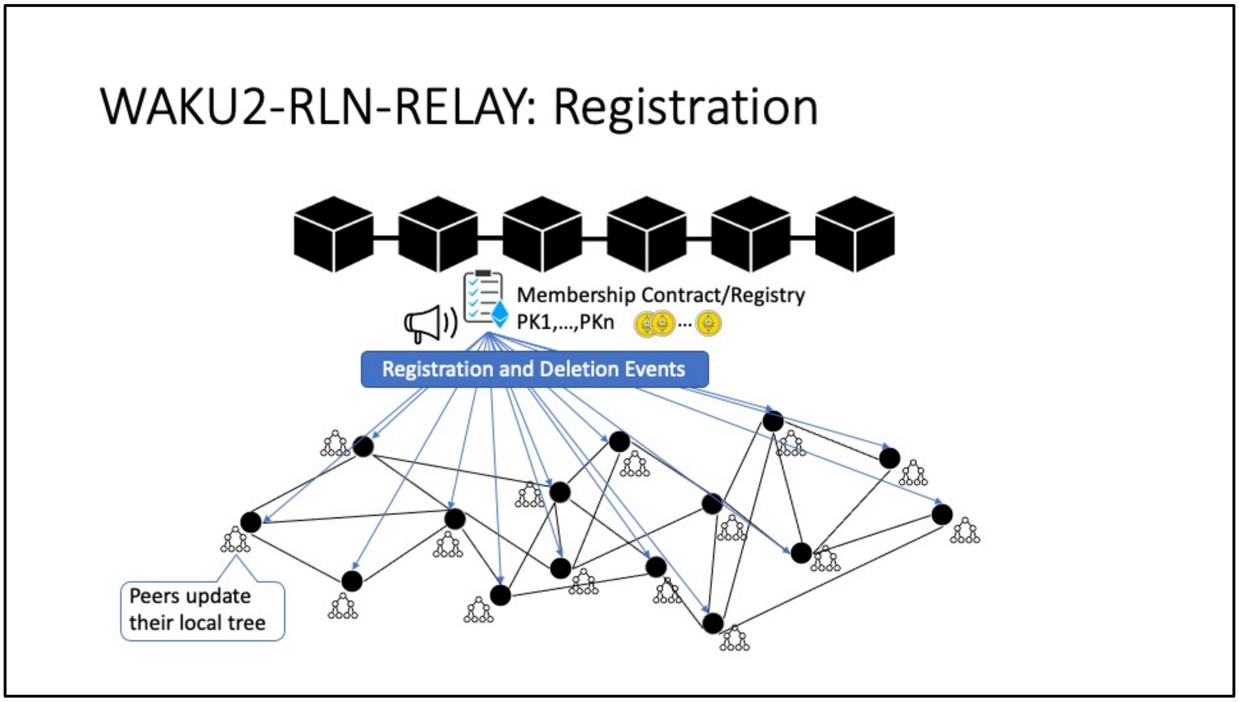
Each peer has a rln pk, and the list of pks is stored in a contract deployed on the Eth blockchain.



A peer willing to publish a message should register by sending a tx to the contract that contains its rln pk and some amount of Ether. This amount is deposited on the contract to prevent spam activity.



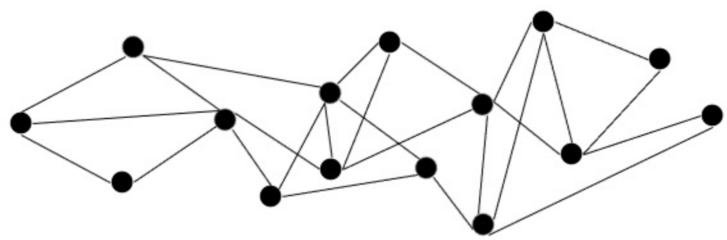
Peers construct the rln membership Merkle tree locally



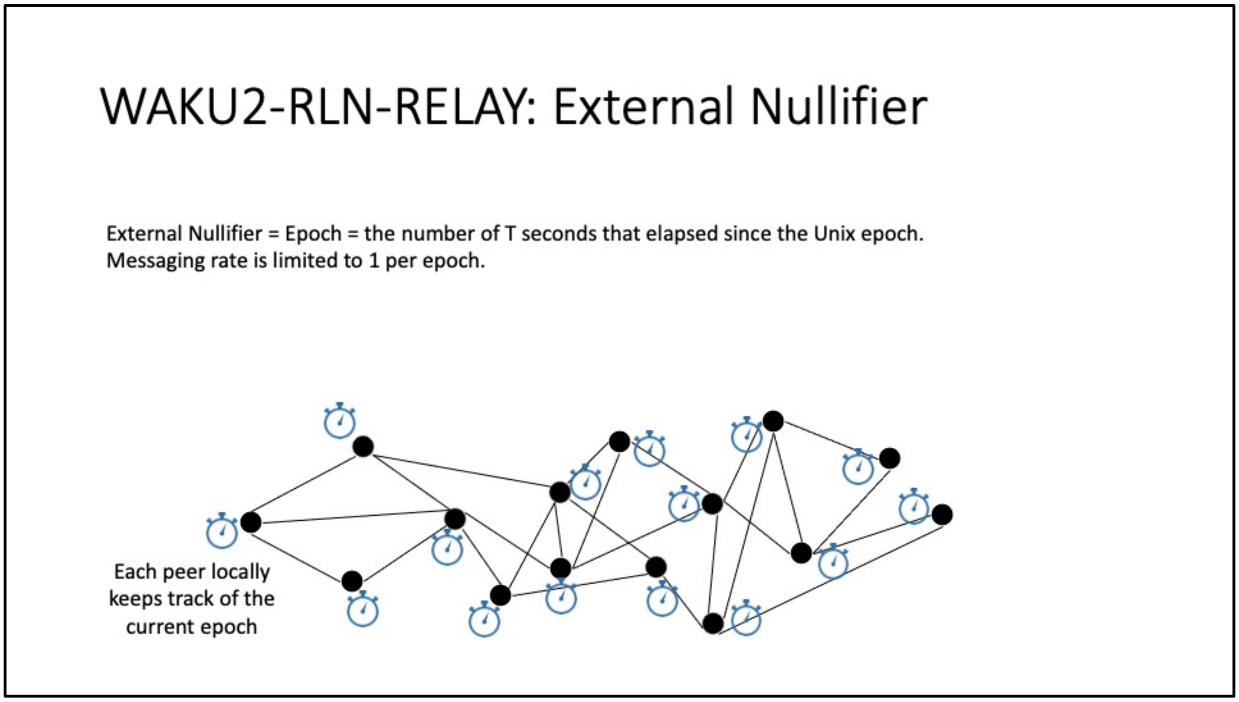
And listen for the registration and deletion events emitted from the contract in order to update their trees.

WAKU2-RLN-RELAY: External Nullifier

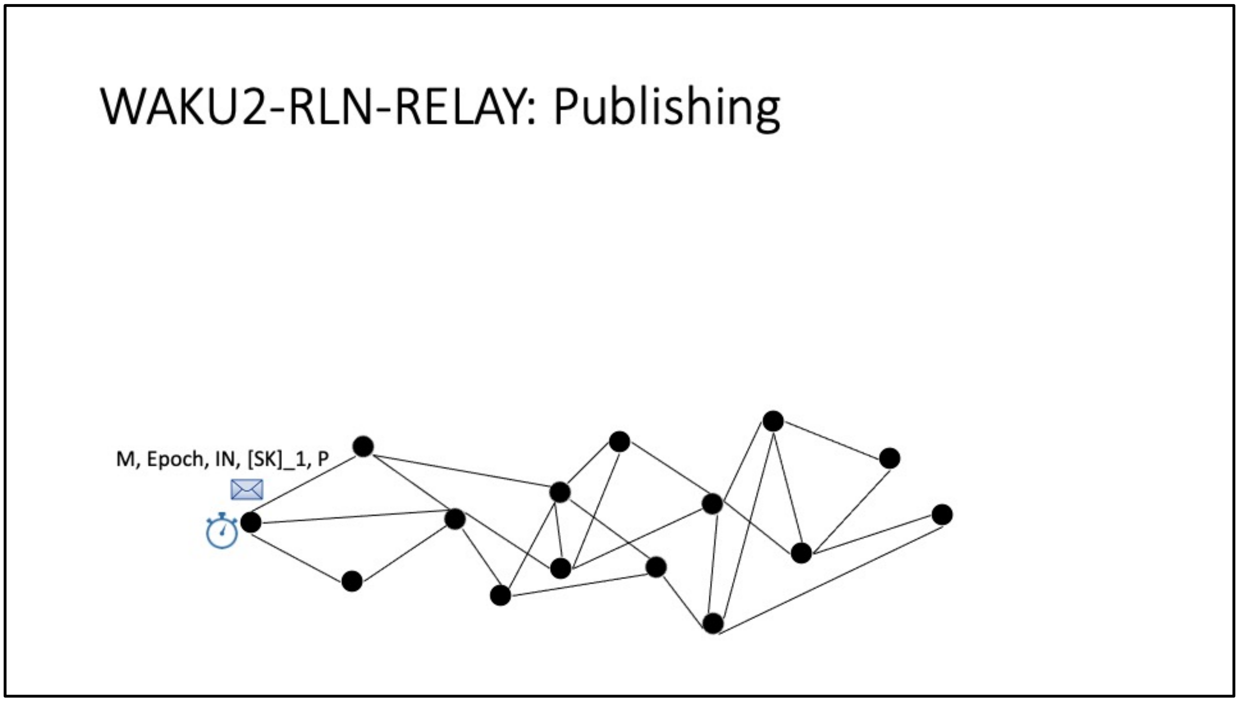
External Nullifier = Epoch = the number of T seconds that elapsed since the Unix epoch.
Messaging rate is limited to 1 per epoch.



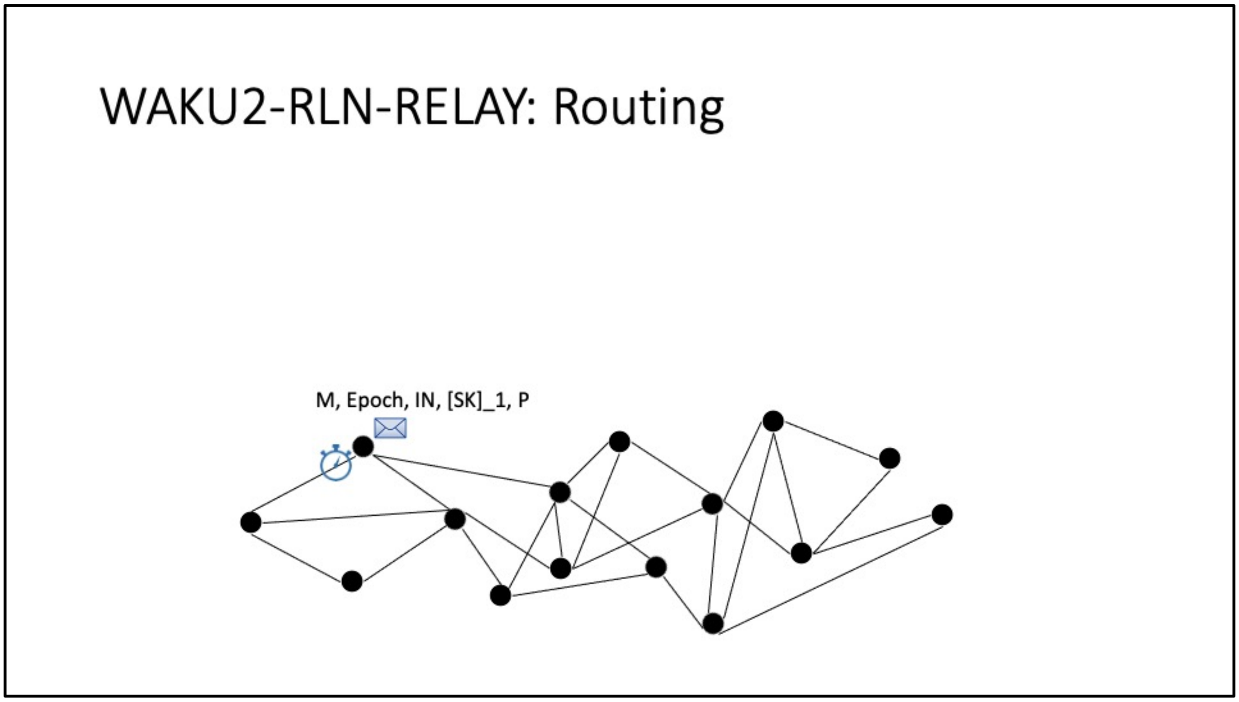
For the external nullifier we define it as the number of T seconds (where T is a system design parameter) that elapsed since the Unix epoch.
Peers are allowed to publish one message per epoch without being slashed



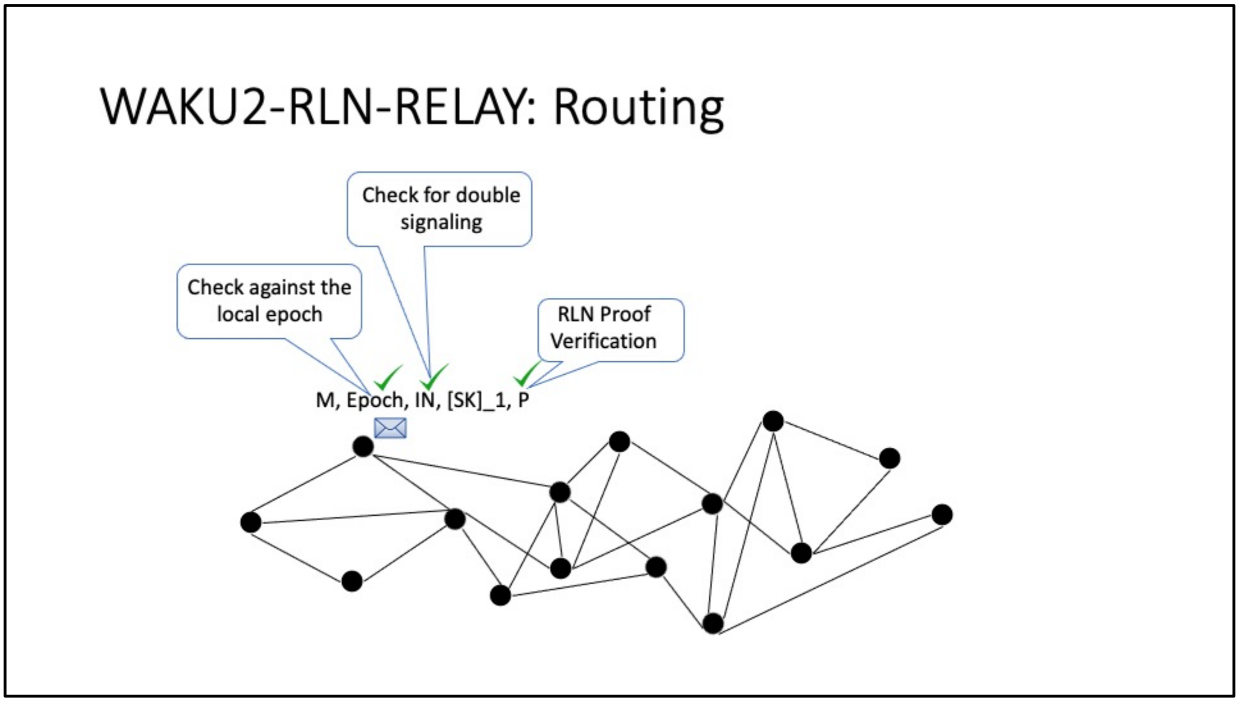
Each peer locally keeps track of the current epoch.



Message publishing in the network is the same as RLN framework
The message owner, attaches the external and internal nullifiers, together with the share of its sk, and zero knowledge proof part to the message



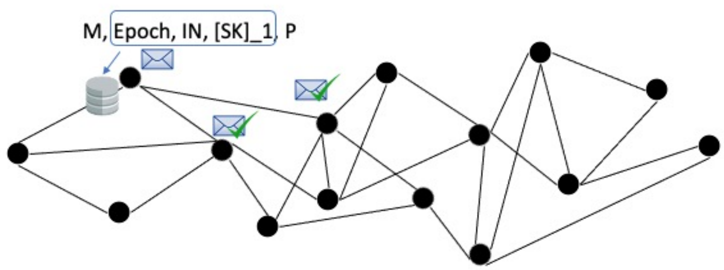
A routing peer follows the regular routing protocol of wakurelay (gossipSub protocol)
and in addition
does the verification steps of RLN construct



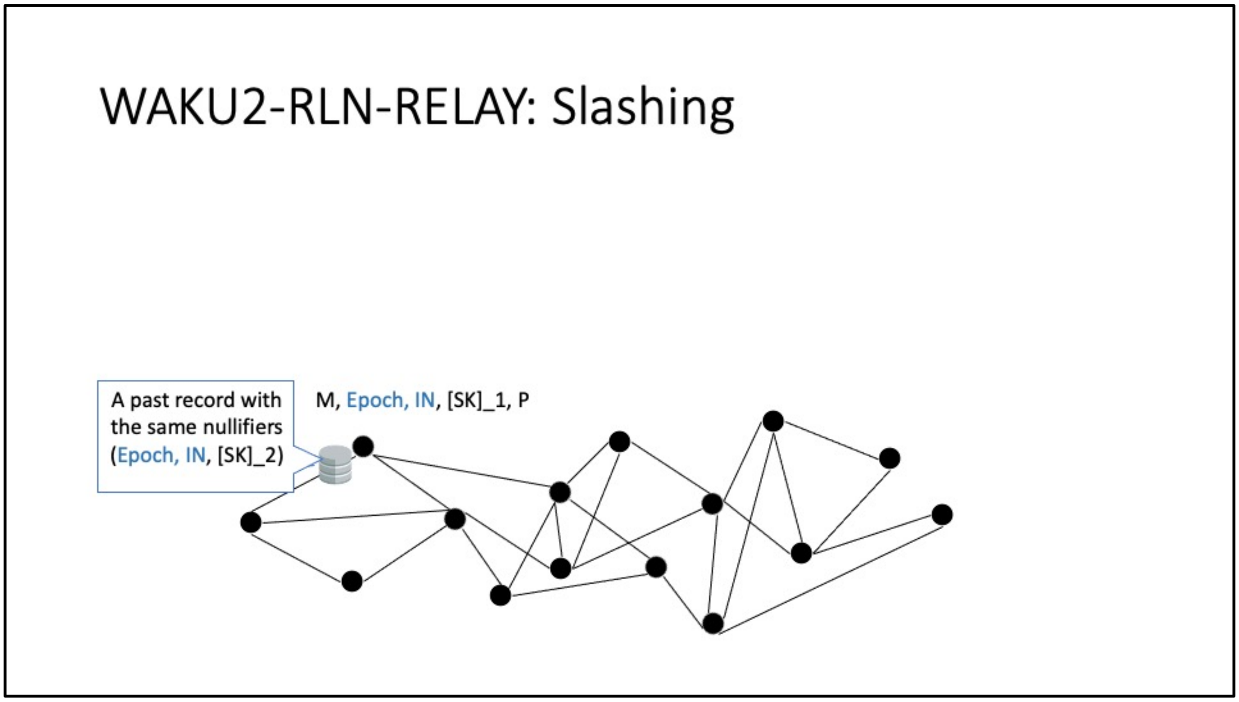
It verifies the proof
Also validates the Epoch of the incoming message against its local Epoch to see if there is a huge gap or not

and checks the nullifiers to see if double signaling has happened
If all the checks pass, relays the message

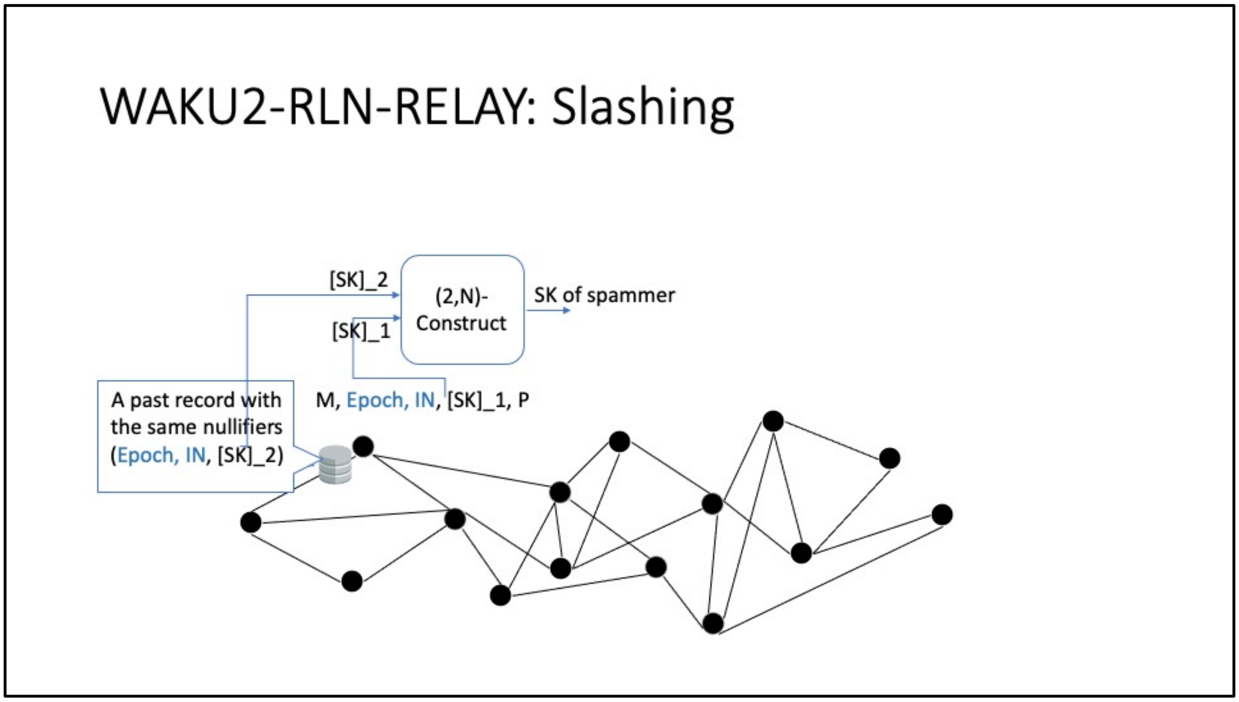
WAKU2-RLN-RELAY: Routing



keeps record of of the nullifiers of the messages, it is needed to catch double signaling for the future messages

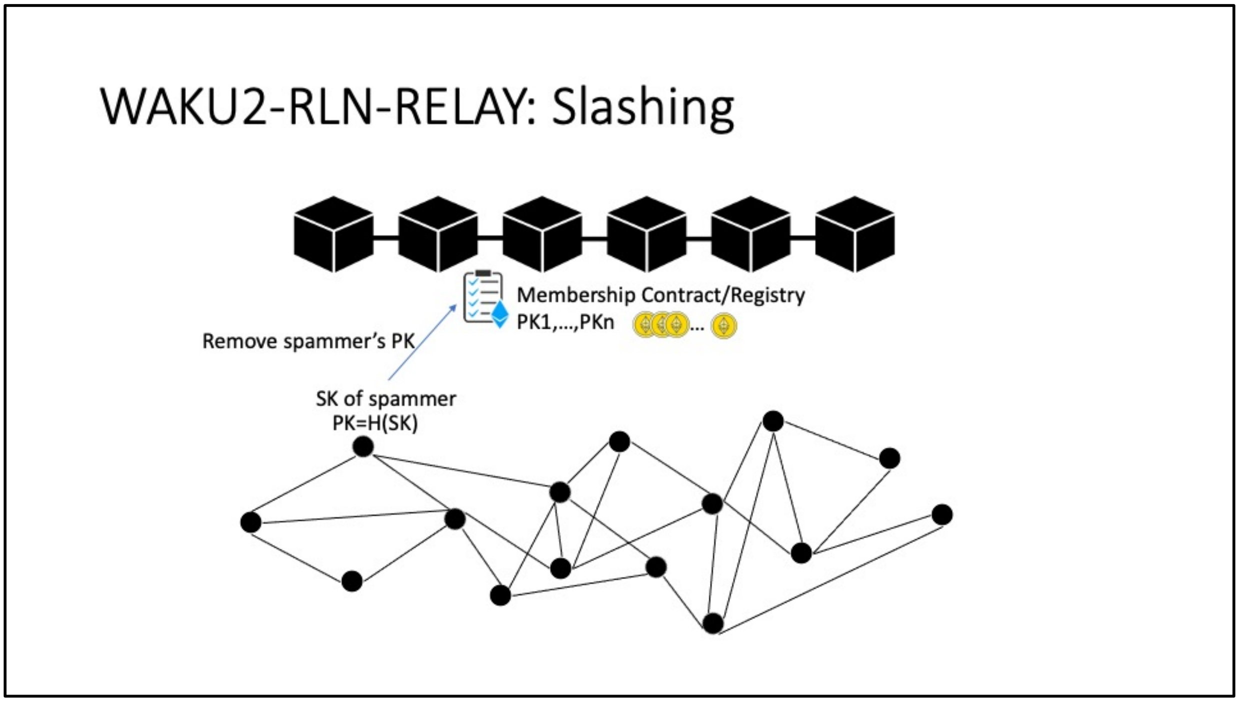


But what if the routing peer finds out that the messaging rate is violated, i.e., an old message with the same Epoch and internal nullifier values

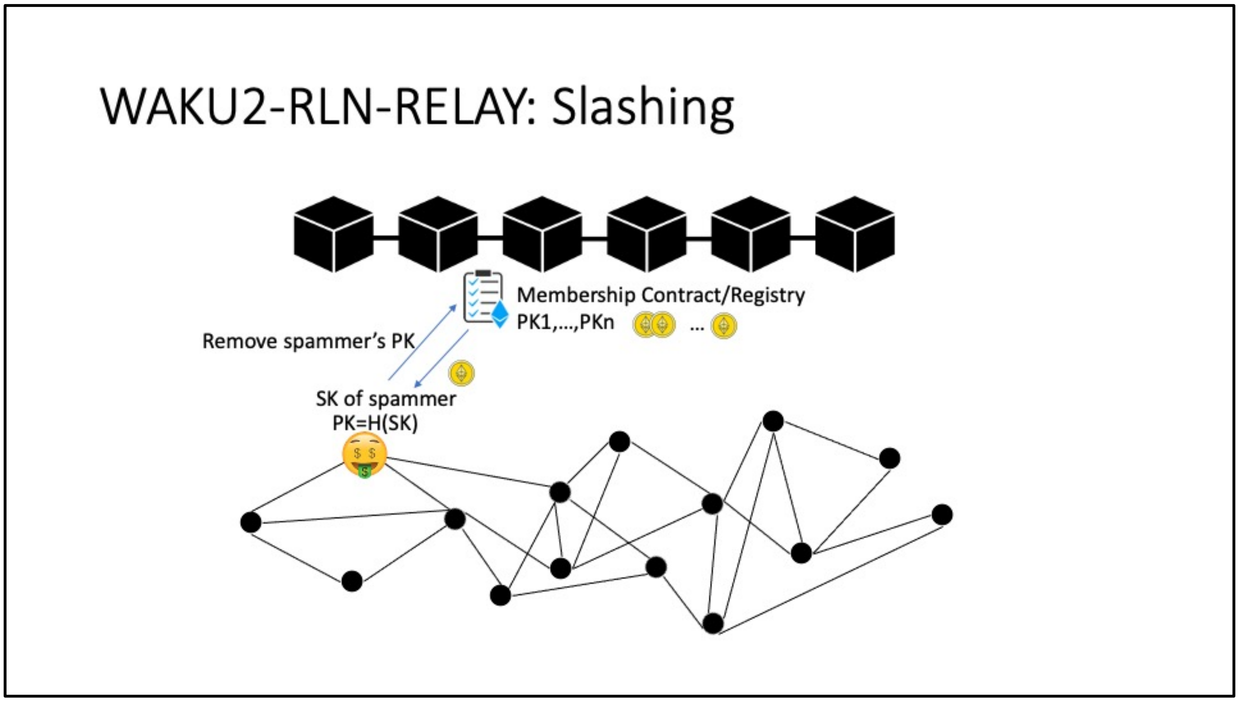


But what if the routing peer finds out that the messaging rate is violated, i.e., an old message with the same Epoch and internal nullifier values

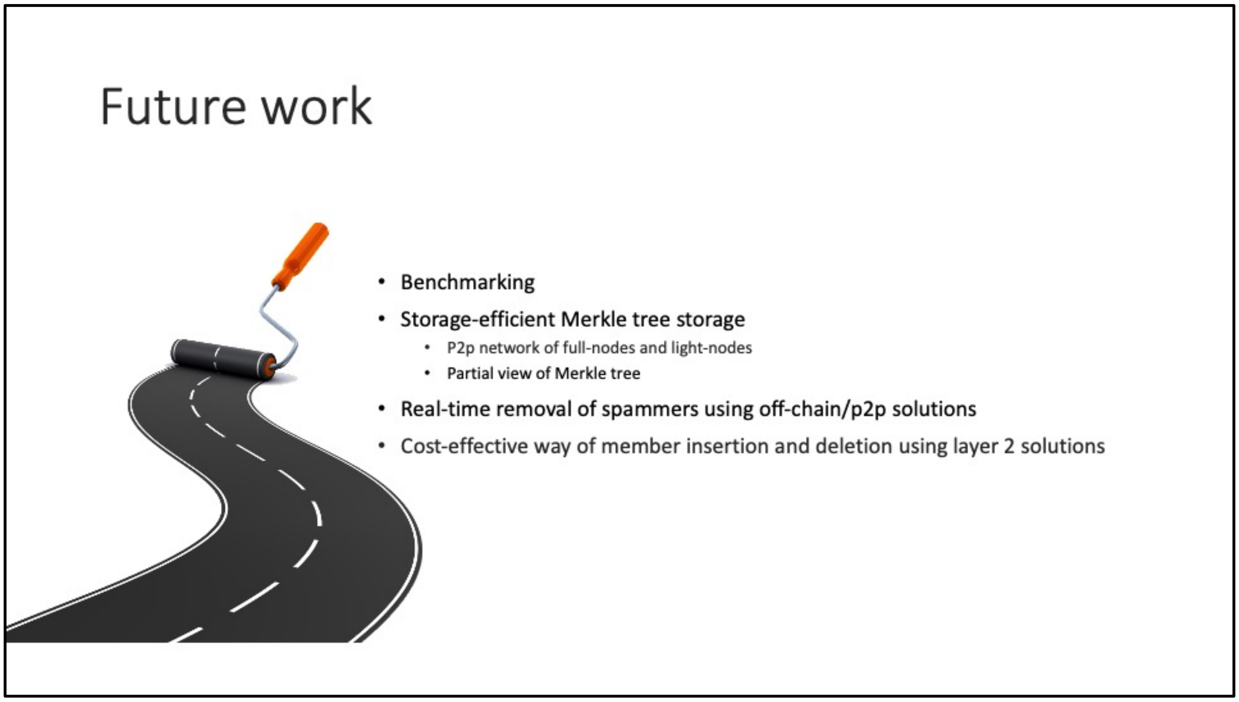
it reconstructs the sk of the spammer



Sends a transaction to the contract and removes the spammer pk from the group



and withdraws the spammer's fund



Now, lets browse our future plan:
Benchmarking is the first on our future work list
The next is to address storage overhead. Currently, peers maintain the entire tree locally which takes up to 67 MB for tree with depth 20 (and almost 274 GB for d=32). This overhead wont fit resource limited devices.

- One solution is to use the light-node and full-node architecture where resource-full nodes retain the entire tree and serve it to the nodes with limited storage.
- Another possible solution is to have a partial view of the tree and yet being able to construct and update the tree root and the authentication path when group state changes

On-chain slashing is subject to delay (the tx has to be mined), so is the removal of spammers. We are seeking an off-chain slashing method, where peers communicate the pk of the slashed user in a p2p manner, to provide real-time spam-protection

The other direction to pursue is to provide a cost-effective way of member insertion and deletion using layer 2 solutions. The reason is that currently these operations cost almost 40 k gas, which translates to 15 USD which might be not affordable by the users, so an alternative solution is worth investigation

References

- Waku-rln-relay specs: https://rfc.vac.dev/spec/17/
- Waku-rln-relay paper: https://github.com/vacp2p/research/blob/master/rln-research/Waku_RLN_Relay.pdf
- Vac post on Waku-rln-relay: https://vac.dev/rln-relay
- Nim-Waku implementation: https://github.com/status-im/nim-waku
- js-Waku implementation: https://github.com/status-im/js-waku
- RLN Ethereum research post: https://ethresear.ch/t/semaphore-rln-rate-limiting-nullifier-for-spam-prevention-in-anonymous-p2p-setting/5009
- RLN medium post: https://medium.com/privacy-scaling-explorations/rate-limiting-nullifier-a-spam-protection-mechanism-for-anonymous-environments-bbe4006a57d
- RLN circuits: https://github.com/appliedzkp/rln
- RLN circuits spec: https://hackmd.io/7GR5Vi28Rz2EpEmLK0E0Aw
- RLN in Rust: https://github.com/kilic/rln

Thank you

Asymptotic Performance

- The following table summarizes the result of our analysis for Method A. The gas consumption results are borrowed from https://hackmd.io/JoxnlDq3RT6WhtA-KBxtYg?both#A-Pubkey-map.
- Parameters in the table should be interpreted as below:
- - `d` is the tree depth
- - `H` is the size of the hash output
- - `N` is the number of Merkle tree leaves which is `2^d`
- - The gas consumption is for `d=32`
- - A Batch consists of `B=128` keys. <!-- TODO: recalculate these for d=20. -->
-
- | Method | Gas cost for registration | Gas cost for deletion | Storage per user | User computation per update| Bandwidth cost per update| Security | Supported operations| Bootstrapping computation cost| Bootstrapping Bandwidth|
- | :-------------- | :------------------------ | :------------ | :------------------ |:-------------- | :------------------------ | :------------ | :------------------ | :------------ | :------------------ |
- | A: Offchain root | Batch: 20k, Single: 40k | Batch: 20k, Single: 40k | _full tree:_ `d`=32 274 GB, `d`=20 67 MB

partial tree: `d`=32 2.048 KB, `d`=20 0.128 KB | O(log(`d`)) to recalc root and auth path | `H` | None | Insertion and Deletion | `N` hashing | O(`N`)|

Use this for more https://hackmd.io/U8nFzfLpQVGu5Zu0dVfJ_Q

Asymptotic Performance

The following table summarizes the result of our analysis for Method A. The gas consumption results are borrowed from https://hackmd.io/JoxnIDq3RT6WhtA-KBxtYg?both#A-Pubkey-map.
Parameters in the table should be interpreted as below:

- `d` is the tree depth
- `H` is the size of the hash output
- `N` is the number of Merkle tree leaves which is `2^d`
- The gas consumption is for `d=32`
- A Batch consists of `B=128` keys.

Method	Gas cost for registration	Gas cost for deletion	Storage per user	User computation per update	Bandwidth cost per update	Security	Supported operations	Bootstrapping computation cost	Bootstrapping Bandwidth
A: Offchain root	Batch: 20k, Single: 40k	Batch: 20k, Single: 40k	*full tree:* `d` =32 274 GB, `d` =20 67 MB *partial tree:* `d` =32 2.048 KB, `d` =20 0.128 KB	O(log(`d`)) to recalc root and auth path	`H`	None	Insertion and Deletion	`N` hashing	O(`N`)

Implementation Details

- zkSNARKs Groth16
- RLN circuit: RLN Rust lib
- Key generation: powers of tau MPC + MPC for circuit dependent parameters