Authors and attributions

Blagoj and Barry WhiteHat.

Introduction

We propose an idea for a decentralised rate limiting service for web applications which offers protection from brute force, DoS and DDoS attacks, Web Scraping and API overuse.

We plan to implement this service by using the RLN (Rate Limiting Nullifier) construct and InterRep.

RLN (Rate Limiting Nullfier) is a construct based on zero-knowledge proofs that enables spam prevention mechanism for decentralized, anonymous environments.

For RLN to be used, the users first need to register to the application with their public key - they're added to a membership Merkle tree upon successful registration. After that they can use the protocol protected by RLN, but with each action they leak a portion of their private key and if they break the anti-spam threshold with a predetermined frequency of requests their private key can be fully reconstructed (by the properties of Shamir's Secret Sharing scheme). By having their private key revealed the user can be removed from the membership tree and their protocol related stake can be slashed, if staking is enabled for the RLN application. You can find out more about RLN here.

InterRep is a service for linking user web3 identities with their reputable social media accounts. You can find more about InterRep here.

Why it is important?

Request spamming attacks on application layer are big problem for many applications.

Brute force attacks, DoS and DDoS attacks, Web Scraping and API overuse can lead to revenue loss, increased infrastructure costs and valuable information leakage. Also brute force attacks can make certain websites impossible or at least prohibitively expensive to run.

The solutions on the market offering request spam protection are not efficient enough and degrade the user experience in the false-positives scenario - the users usually need to solve CAPTCHAs to verify their identity. The spam-resistance, efficiency level of spam detection and the user experience on the application layer can be largely improved by using RLN and rich user identities.

Additionally there aren't any privacy-first rate limiting services on the market, and the desirability for and privacy and anonymity is ever increasing. As ZK technology develops, more anonymity and privacy focused application will emerge on the market which will be in need of rate limiting services.

We also think that by combining new and experimental technologies such as RLN and InterRep which are more web3 native, we can provide useful services to the web2 world and attract more people into the web3 world.

This project is an interesting experiment that tries to bridge the gaps between web2 and web3 and will enable us to explore what impact the web3 technologies can have on the web2 world.

Description

Rate limiting/DDoS protection service for websites and web applications which offers protection from Brute force attacks, DoS and DDoS attacks, Web Scraping and API overuse. The request spam protection is on application layer of the network stack (layer 7 protection), and it will be functioning similarly in a traditional way as the Cloduflare's Rate Limiting product.

The Cloudflare Rate-Limiting service works by the websites defining rate-limiting rules, which when broken the user is either temporarily banned from accessing the application or is given CAPTCHA challenges to solve. The Cloudflare Rate-Limiting service also rate-limits the users by their IP address.

We can improve upon these features by offering to users to be able to create ZKPs instead of solving CAPTCHAs themselves, and for the applications to have better request spam protection by identifying the users by their rich user identities in an anonymous way. By using ZKPs, the applications are not aware of the real identities of their users and user privacy is preserved. A rich user identity is a Web3 identity such as Ethereum address linked with personal credentials of high value such as reputable twitter account, email, driver's licence, bank account. The rich user identities have the property of hard replicability, thus reducing the possibilities for sybil attacks a lot.

The initial idea is to implement the rate limiting service as a centralised server at the start and leverage a more distributed architecture later.

The rate limiting service will act as a middleman between the frontend clients and their backends.

It will rate limit the users based on their IP addresses, but also it will be able to verify ZK proofs that the users will generate. The users will need to be registered to the service to generate valid ZK proofs. The service will enable user registration for the users that want to avoid solving challenges manually by providing a user interface, where by using InterRep the users will be able to register to the service.

The server will host a single membership Merkle tree for all of the users, meaning that the users are registered at service level and not on application level. In other words after registering users will be able to access all of the applications protected by the rate limiting service.

If a user sends too many requests per second, the service will be able to reliably identify and remove the user from it's membership tree, by the properties of the RLN proof system. Once slashed the users will not be able to access the applications protected by the rate limiting service again, nor be able to register (this might be too restrictive, and we might loosen up these conditions).

Implementation details

The solution can be logically divided in three parts:

- Rate limiting service
- Frontend clients
- Backend apps

Rate limiting service

A device or group of devices (for now a centralised server) which will store a single RLN membership tree, additional data structures for keeping track of ZKP metadata such as nullifiers and key shares, data structures necessary for web3 identity linking (the InterRep part), as well as additional data structures such as a ban list.

Upon user registration the service will store the user identity commitment (Poseidon hash of their private key) in its RLN membership merkle tree.

If a user exceeds request limits the service will be able to reconstruct the user's private key and from the private key reveal the identity commitment and ban the user by removing them from the RLN membership merkle tree.

The service will also provide an UI through which the users will be able to register and also a REST API though which registered users can obtain the parameters necessary for generating valid ZK proofs in a trust-less manner.

Frontend clients

The frontend clients for the apps that want to be protected by the rate limiting service will need to implement a special library which will handle the communication with the rate limiting service.

The frontend clients will generate and store the private key for the user.

The library will be able to generate ZK proofs and include the ZK proof as well as additional parameters necessary for verification of it as a HTTP headers. All of the HTTP requests will be sent to the rate limiting service. The frontend clients will obtain the root of the membership merkle tree and their auth path (parameters necessary for generating the ZK proofs), from the rate limiting service via API calls.

Backend apps

The backend apps will only receive filtered requests, redirected from the rate limiting service only. If a user tries to access the backend app (while skipping the rate limiting service), they will be redirected to the rate limiting service first.

References and further reading

1. RLN introductory post - https://medium.com/privacy-scaling-explorations/rate-limiting-nullifier-a-spam-protection-mechanism-for-anonymous-environments-bbe4006a57d
2. InterRep overview - https://jaygraber.medium.com/introducing-interrep-255d3f56682
3. What is rate limiting - https://www.cloudflare.com/en-gb/learning/bots/what-is-rate-limiting/