Alma Mater Studiorum · Università di Bologna

Scuola di Ingegneria e Architettura
Dipartimento di Informatica · Scienza e Ingegneria · DISI
Corso di Laurea in Ingegneria Informatica

SISTEMI ANTI-DENIAL OF SERVICE IN AMBIENTI ANONIMI BASATI SU ZK-SNARK

Relatore:
Prof. Paolo Bellavista

Presentata da:
Straccali Leonardo

Anno Accademico 2022/2023

Indice

Elenco delle figure

Capitolo 1

Analisi del protocollo RLN

Nel presente capitolo, tratteremo la discussione del protocollo RLN (Rate Limiting Nullifier), che consente la costruzione di regole di rate-limiting anti DoS e spam in ambiente anonimo. La prima proposta del protocollo è stata sviluppata da Barry WhiteHat, un ricercatore attivo nel campo della blockchain e delle applicazioni Zero Knowledge, nel seguente post: "Semaphore RLN, rate limiting nullifier for spam prevention in anonymous p2p setting". Attualmente, RLN fa parte di (PSE) Privacy & Scaling Explorations, un team multidisciplinare sostenuto dalla Fondazione Ethereum, che esplora nuove tecnologie Zero Knowledge e altre primitive crittografiche. Alcuni progetti di rilievo includono zk-kit, Interep e Semaphore. RLN è ancora un protocollo poco conosciuto, e non esistono ancora grosse implementazioni al di fuori del contesto di ricerca. Il progetto in stato più avanzato è relativo a un lavoro portato avanti da Vac che sta lavorando sull'implementazione del protocollo all'interno di Waku v2, la seconda versione di un protocollo di comunicazione peer-to-peer privacy-preserving in ambiente decentralizzato.

Il meccanismo RLN prima di tutto impone agli utenti di generare una chiave privata che sarà utlizzata per ricavare un identity commitment utilizzando la funzione hash Poseidon. In seguito per registrarsi, gli utenti devono fornire il loro identity commitment, che viene memorizzato in una foglia dell'albero di Merkle, il quale rappresenta la struttura dati dei membri del servizio. Per evitare attacchi di tipo Sybil il protocollo dà la possibilità di richiedere agli utenti in fase di registrazione anche una "stake" ovvero un informazione di valore appartenente all'untente che sia di difficiele riproducibilità.

Per ogni interazione con l'applicazione, gli utenti devono generare una prova per dimostrare l'appatenenza all'albero di Merkle e devono rialscaire una prozione della loro chiave privata, dimostrando che si tratti effettivamente di una parte valida di essa. Durante questa procedura non vogliamo che nessuna informazione venga rilascaita oltre alla appartenenza e alal correttezza della porzione di chiave e per fare ciò sfruttiamo la tecnologia zk-SNARK

Per evitare lo spam, RLN implementa una regola anti-spam generale in cui gli utenti non possono effettuare più di X interazioni per epoca. Questo può essere applicato utilizzando lo schema di condivisione dei segreti di Shamir per dividere la chiave segreta in n parti, e se gli utenti effettuano più interazioni di quelle consentite per epoca, la loro chiave segreta può essere ricostruita.

Infine, il meccanismo RLN consente agli utenti di essere rimossi dall'albero di appartenenza da chiunque conosca la loro chiave segreta. Se si verifica un evento di spam, lo stake economico dello spammer può essere inviato al primo utente che segnala correttamente lo spammer fornendo la chiave segreta ricostruita dello spammer come prova.

Prima di esaminare in dettaglio lo sviluppo delle singole parti, analizzeremo gli strumenti utilizzati dal protocollo per implementare le sue funzionalità.

1.1 Strumenti

1.1.1 Alberi di Merkle

Gli alberi di Merkle sono una particolare struttura ad albero che sfrutta le proprietà delle funzioni di Hash al fine di ottimizzare le operazioni di ricerca e identificazione delle modifiche all'interno della collezione. La struttura degli alberi di Merkle è caratterizzata dalle di foglie, che rappresentano i dati, a cui è stata applicata una funzione di hashing, e dagli altri nodi dell'albero, che sono ottenuti applicando la funzione Hash ai loro figli, fino a raggiungere la radice. Negli alberi di Merkle la radice è una funzione Hash che rappresenta univocamente la struttura. Per capire meglio il concetto immaginiamo di avere un struttura ad albero binaria, e inseriamo all'interno della struttua gli elementi 1,2,3,4,5 a cui applichimao una funzione Hash, a questo punto otterremo la seguente struttura:

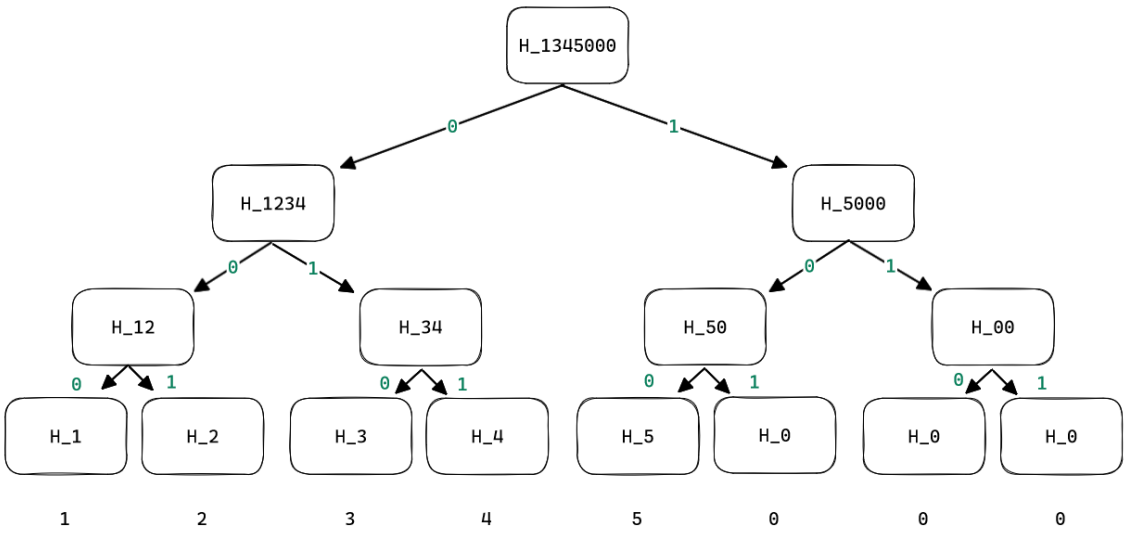

Figura 1.1: Esempio struttura albero di Merkle binario

Possiamo notare che la radice dei Merkle tree possiede la proprietà di essere una sorta di impronta digitale della struttura, in quanto qualsiasi modifica ai dati comporterebbe un cambiamento a cascata dei nodi fino alla radice stessa. Questa proprietà costituisce un vantaggio significativo in termini di efficienza. Infatti, consente una verifica rapida delle modifiche apportate alla struttura, rendendo gli alberi di

Merkle estremamente utili nei campi decentralizzati e nei file system condivisi, dove l'individuazione efficiente dei cambiamenti con poche informazioni è cruciale. Un'altra proprietà altamente utile degli alberi di Merkle è la loro capacità di verificare l'appartenenza di un elemento alla struttura in modo efficiente e senza la necesità di conoscere l'elemento in chairo. Nell'esempio precedente, se si desidera verificare che l'elemento 4 appartenga alla struttura, è sufficiente utilizzare gli elementi H_3, H_12 e H_5000 e il valore H_4, che non rivela alcuna informazion su l'elemento, per ricostruire tutti i nodi fino alla radice. Una volta ottenuta la radice basterà confrontarla con la radice corretta per convicersi della presenza dell'elemento o meno nella struttura.

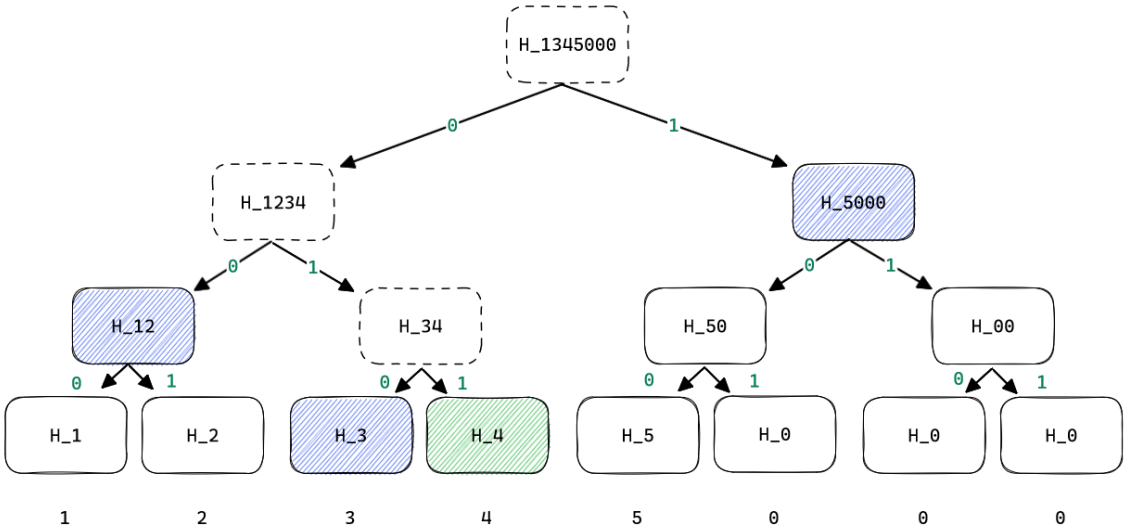

Figura 1.2: Esempio Merkle proof

Questo processo viene chiamato "Merkle proof" o più in generale "proof of membership". Tale processo rappresenta l'approccio che adotteremo per dimostrare la capacità di un utente già registrato e non rimosso di interagire con il sistema nella fase di Interazione del protocollo RLN.

1.1.2 Nuove funzioni di Hash

Indubbiamente, una delle tecniche più utilizzate in crittografia sono le funzioni di Hash. Anche la tecnologia zk-SNARK non può fare a meno di esse. Infatti, come abbiamo già visto in molte situazioni, l'utilizzo di metodi di hashing è stato necessario per ridurre le dimensioni delle informazioni e per nasconderle.Tuttavia, quando si utilizzano le tradizionali funzioni di Hash come la versione sha-256 nel campo delle zero knowledge proof, si verifica un problema. Queste funzioni non sono state concepite per lavorare in domini di campi finiti e, pertanto, l'utilizzo di metodologie come la ripetizione di operazioni bit-wise tende ad aumentare considerevolmente la dimensione dei vincoli del circuito della prova che le implementa. Questo aumenta notevolmente il tempo e la dimensione richiesti per generare le prove. Negli ultimi anni, per superare il problema descritto, sono state utilizzate nuove versioni di funzioni di Hash come Pedersen, Mack e Poseidon. Queste funzioni sono state

progettate specificamente per lavorare nei campi finiti e con l'obiettivo di ridurre al minimo i tempi di generazione e i vincoli necessari per costruire le prove. Tra queste funzioni, la funzione Poseidon è quella che ha ottenuto i risultati migliori e che verra utilizzata nella costruzione del circuito per il protocollo RLN. Di seguito presento delle tabelle contenteti un confronto delle prestazioni tra le più note funzione di Hash per le tecnologie zero knowledge e la funzione sh256, tabelle tratte da "POSEIDON: A New Hash Function for Zero-Knowledge Proof Systems"[1].Nelle tabelle sottostanti possiamo notare che ci sono diverse configurazioni della funzione Poseidon. Infatti, una grande differenza tra le funzioni tradizionali e Poseidon è la possibilità di scegliere il numero di iterazioni e la dimensione del campo finito su cui lavorare, in modo da selezionare la versione più performante a seconda del caso.

Table 4: Number of R1CS constraints for a circuit proving a leaf knowledge in the Merkle tree of 2^{30} elements.

Arity	Width	R_F	R_P	Total constraints
POSEIDON-128				
2:1	3	8	57	7290
4:1	5	8	60	4500
8:1	9	8	63	4050
Rescue-x^5				
2:1	3	16	-	8640
4:1	5	10	-	4500
8:1	9	10	-	5400
Pedersen hash				
510	171	-	-	41400
SHA-256				
510	171	-	-	826020

Table 5: Bulletproofs performance to prove 1 out of 2^{30}-Merkle tree.

Field	Arity	Merkle 2^{30}-tree ZK proof Bulletproofs time		R1CS constraints
		Prove	Verify	
POSEIDON-128				
BLS12-381	2:1	16.8s	1.5s	7290
	4:1	13.8s	1.65s	4500
	8:1	11s	1.4s	4050
BN254	2:1	11.2s	1.1s	7290
	4:1	9.6s	1.15s	4500
	8:1	7.4s	1s	4050
Ristretto	2:1	8.4s	0.78s	7290
	4:1	6.45s	0.72s	4500
	8:1	5.25s	0.76s	4050
SHA-256 [BBB+18]				
$GF(2^{256})$	2:1	582s	21s	762000

Figura 1.3: Confronto con altri algoritmi Hash

1.1.3 Nullifier

I "nullifier" sono dei valori utilizzati per confermare o annullare operazioni. Nelle applicazioni che garantiscono l'anonimato, questi valori vengono spesso utilizzati per evitare il problema del "double-signaling", ovvero impedire che un utente utilizzi o esegua un'operazione che dovrebbe essere unica per ogni utente più di una volta. Questo problema può verificarsi perché, in assenza di un collegamento tra i dati e le identità degli utenti, non è possibile verificare se un utente ha già effettuato o completato una determinata operazione. Ad esempio, in un'applicazione elettorale è importante impedire che un singolo utente voti più di una volta, mentre in una blockchain è fondamentale evitare che la stessa moneta venga spesa più volte. Per evitare questo problema si utilizzano i nullifier ovvero valori univoci legati all'operazione che riamangono privati fino al momento dell'effetuazione dell'eoperazione e una volta attauta vengono resi pubblici, e salvati in modo da poterli consultare successivamente.

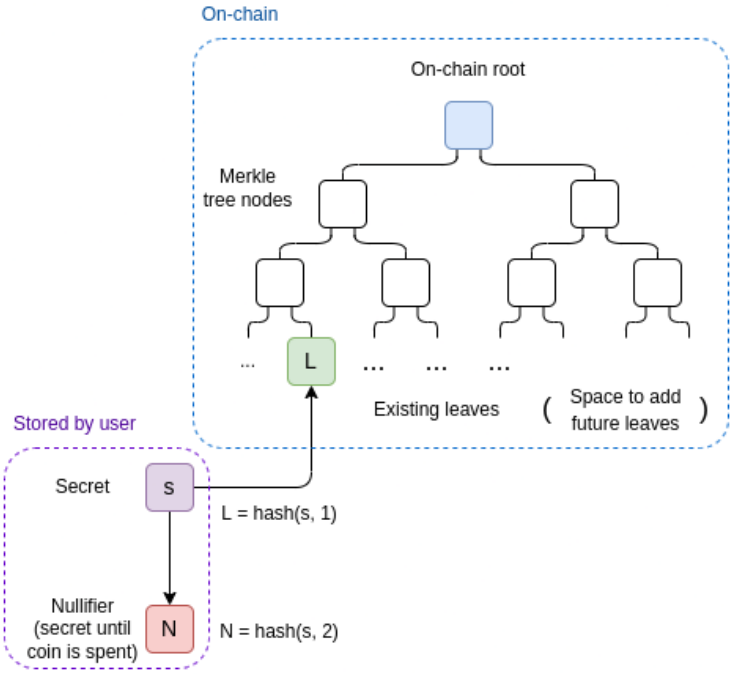

Figura 1.4: Rappresentazione della creazione di un nullifier, tratta da [2]

Dalla descrizione fornita potrebbe subito venire in mente il concetto di "rate-limiting" e l'uso dei "nullifier" per attuare questa procedura in modo anonimo. In effetti, è possibile limitare gli utenti utilizzando questa metodologia, ma non si potrà rivelare l'identità dell'utente malintenzionato che, in questo modo, potrebbe riprovare indisturbato ad attacare il sitema.

1.2 Registrazione

La registrazione consiste nella prima fase del protocollo RLN, in questa fase gli utenti, previa la generazione di una chiave privata e ricavando da questa un "identity commitment" utilizzando la funzione hash Poseidon. Per semplificare la spiegazione da qui in poi, utilizzeremo il simbolo a_0 per indicare la chiave privata.

$$identityCommitment = Poseidon(a_0)$$

in alcuni casi potremmo avere la necessità di elaborare maggiormente la creazione dell'identity commitment per aumentare il livello di sicurezza o associare all'identity commitment anche una stake. Una volta generato l'identity commitment l'utente viene inserito all'interno del albero di Merkel del sistema. A questo punto l'utente sarà in grado di generare dell "proof of membership" e quindi di interrogare il sisteam. Per evitare attacchi di tipo Sybil, è posibile implementare dell'tecniche che vincolino l'inserimento dell'utente nell albero al rispetto di alcuni vincoli, come posseso identità digitali o di un profilo social accreditato o ancora un portafolgio di criptovalute.

Tra i progetti del gruppo PSE troviamo anche un progetto chiamato Interep, il cui scopo è essenzialemtene estrapolare la reputazione di un utente e inserire un iden-

tity commitment a lui associato all'inetrno di alcuni gruppi di reputazione. Questi gruppi (alberi di Merkel) sono divisi in base al grado di reputazione degli utenti. Attualemente il progetto Iterep permette di estrapolare la reputazione di un utente dagli acocunt socail di Github, Twitter e Reddit.

1.3 Interazione

Passata la fase di registrazione, gli utenti avranno la possibilità di effettuare richieste. Ora ci chiediamo come possaimo implementare una regola di rate-limiting del tipo: "Un utente non può fare più di k richieste per un determinato lasso di tempo e (epoca)"?

RLN utilizza l'algoritmo Shamir's Secret Sharing (SSS) che permette di suddividere un segreto in n parti in cui ciascuna parte del segreto non rivela nulla, ma se ne vengono combinate k dove $k < n$ allora il segreto può essere ricostruito. Ogni volta che l'utente fa una richiesta al sistema, rilascia una delle n porzioni in cui la sua chiave privata a_0 è stata divisa. In questo modo, se l'utente raggiungesse il valore di soglia k imposto dalla regola di rate limiting il sistema sarebbe in grado di ricostruire a_0 svelando l'identità dell'utente.

La procedura per dividere e ricostruire il segreto si basa ancora una volta sull'utilizzo di polinomi, in particolare sull'interpolazione di Lagrange. Il grado del polinomio da utilizzare per ricostruire la chiave privata a partire dai suoi componenti dipende strettamente dal numero di richieste che si desidera consentire. In particolare, per interpolare (cioè ricostruire) un polinomio di grado k, abbiamo bisogno di almeno $k + 1$ punti.

Per consolidare il tutto vediamo un esempio. immaginiamo di voler applicare una regola di rate limiting in cui : "Un utente non può fare più di 1 richiesta al minuto".

Per prima cosa costruiamo un nullifier, che ci servirà per identificare i messaggi inviati all'interno di un epoca:

$$externalNullifier = Poseidon(epoch, rln\_identifier)$$

dove *epoch* è l'epoca in cui è stato invito il messaggio e $rln\_identifier$ è un valore univoco per tutta l'applicazione che utilizza il protocollo. Poi proseguiamo costruendo il polinomio che dovrà essere ricostruito dal verificatore (il sitema), costruiamo il polinomio in modo che con due richieste ovvero due punti sia possibile ricostruire il segreto, quindi usiamo un poilinomio di primo grado costruito come segue:

$$A(x) = a_1 * x + a_0$$

possaimo notare che il polinomio valutato in 0 vale a_0 la nostra chiave privata, mentre a_1 è definito come $a_1 = Poseidon(a_0, externalNullifier)$ e permette di variare il polinomio in base all'epoca in cui viene fatta la richietsa. Quando un utente invia una richiesta al sistema calcola anche due coordinate $x = Poseidon(richiesta)$ e ottenendo $y = A(x)$ che identificano una porzione delle segreto ovvero un punto sul polinomio. Se un utente malitenzionato inviasse un altro messagio con le nuove coordinate $x_2 = Poseidon(richiseta_2)$ e $y_2 = A(x_2)$ nella setessa epoca (ovvero

con lo stesso polinommio) il sitema sarebbe in grado di ricostruire il polinomio interpolando i due punti. Nel caso di polinomio di primo grado questa procedura è immediata

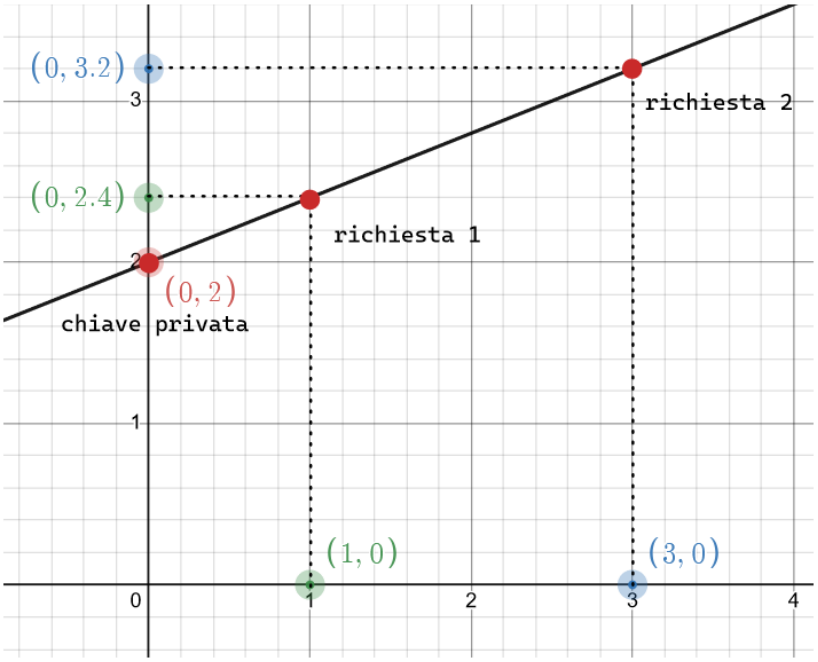

Figura 1.5: Grafico SSS pilinomio primo grado

RLN utilizza l'algoritmo Shamir's Secret Sharing (SSS) che permette di suddividere un segreto in n parti in cui ciascuna parte del segreto non rivela nulla, ma se ne vengono combinate k dove $k < n$ allora il segreto può essere ricostruito. l'algoritmo SSS utilizza l'interpolazione di lagrange per Per compredere il tutto riportamoci ad un esempio immaginiamo di voler implementare una regola del tipo "Un utente non deve poter effettuare più di 1 richiesta al minuto"

1.4 Punizione

La fase di punizione o slashing varia molto a seconda del contesto applicativo e alla presenza o meno di una stake, ma la logica è sepre

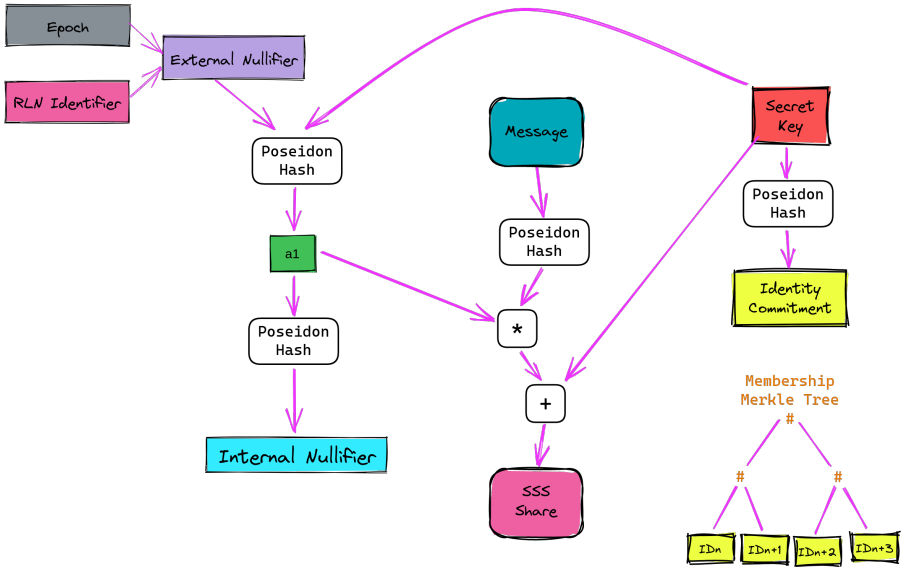

Figura 1.6: Diagramma funzionametno RLN, tratto da [3]

Bibliografia

[1] Lorenzo Grassi et al. *Poseidon: A New Hash Function for Zero-Knowledge Proof Systems*. Cryptology ePrint Archive, Paper 2019/458. https://eprint.iacr.org/2019/458. 2019. URL: https://eprint.iacr.org/2019/458.

[2] Vitalik Buterin. «Some ways to use ZK-SNARKs for privacy». In: (). URL: https://vitalik.ca/general/2022/06/15/using_snarks.html.

[3] *RLN (Rate-Limiting Nullifier) Docs*. URL: https://rate-limiting-nullifier.github.io/rln-docs/.