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

Introduzione

L'implementazione di strumenti atti a combattere gli attacchi di tipo **denial of service** (**DoS**), sono da molto tempo materia di studio e di ricerca. Questi attacchi possono rappresentare una minaccia grave, soprattutto quando si verificano su servizi cruciali e sensibili. Per tale motivo, negli anni sono state sviluppate molte strategie diverse per impedire a singoli o ai gruppi di computer (noti anche come "zombie armies") di attaccare servizi o reti. Attualmente una delle metodologie di protezione più comuni agli attacchi DoS è il **Rate-limiting**, che permette di imporre una frequenza massima accettabile, per le richieste, al fine di evitare sovraccarichi e congestionamenti delle risorse. In particolare, nei sistemi su larga scala, la limitazione della frequenza rappresenta uno strumento essenziale per garantire la disponibilità e l'integrità delle risorse e dei servizi.

Un altro campo di interesse in cui si sono fatti molti progressi negli ultimi anni, grazie soprattutto all'avvento delle tecnologie blockchain, è stato quello dell'anonimato in rete. Con il termine anonimato in rete ci si riferisce alla condizione in cui, sulla base di una conoscenza parziale o totale delle interazioni di un utente in una rete, non è possibile risalire all'identità dell'utente stesso; permettendo un'interazione con i servizi offerti dalla rete in assoluta riservatezza. L'anonimato in rete presenta molteplici vantaggi, soprattutto in contesti in cui la privacy costituisce un requisito fondamentale, come ad esempio nelle votazioni o nelle transazioni finanziarie. Inoltre, la possibilità di mantenere l'anonimato può rivelarsi altrettanto utile in ambiti più diffusi, come le conversazioni online o i social network, garantendo la libertà di espressione e la tutela della propria sfera privata.

Tuttavia, l'anonimato in rete presenta anche alcune criticità, tra cui la principale è rappresentata dalla difficoltà di controllo. La ragione di questa difficoltà è da individuare nel fatto che le metodologie di sicurezza a livello applicazione (pila ISO/OSI) generalmente si basano sull' analisi del comportamento degli utenti o dei dispositivi, nel corso del tempo, al fine di rilevare i pattern di attività che potrebbero suggerire la presenza di un attacco. In un contesto anonimo, in cui le identità degli utenti non sono tracciabili, ciò diviene notevolmente più arduo. Esistono strumenti di rate-limiting efficaci a livelli inferiori della pila ISO/OSI, come a livello di trasporto o di sessione. Tuttavia, la loro attuazione comporta alcuni svantaggi, come il rischio di limitare o bloccare numerosi indirizzi IP tradotti sotto una NAT, minare la privacy degli utenti disattivando protezioni utili come TLS o ancora essere elusi da tecniche di mascheramento, come l'IP spoofing. Per tali motivazioni, per un servizio diffuso in ambiente anonimo, non è possibile fare affidamento esclusivamente su questi strumenti.

La presente tesi affronta la discussione di un protocollo chiamato **RLN (Rate-Limiting Nullifier)**[1] basato sulla tecnologia **zk-SNARK**, che permette di attuare un rate-limiting in ambiente anonimo. Il protocollo è composto da tre parti generali, le quali si differenziano, nei dettagli, a seconda del dominio applicativo. Fasi del protocollo:

- **Registrazione**: Durante questa fase, gli utenti che desiderano accedere al servizio devono registrarsi fornendo una prova del possesso di determinati requisiti. Questa prova di possesso è denominata "identity commitment" e viene conservata e utilizzata nelle fasi successive del protocollo. I dati personali che soddisfano i requisiti sono definiti "stake" e possono assumere forme diverse a seconda del contesto applicativo. Ad esempio, possono essere costituiti da un profilo su un social network, dall'indirizzo di un portafoglio di criptovalute o da un'identità digitale come lo SPID o la CIE.

  Gli stake non vengono conservati dal protocollo. Alla fine della fase di registrazione, il sistma è a conoscenza unicamente nel fatto che un nuovo utente anonimo che soddisfa le specifiche della registrazione è stato aggiunto al servizio. La presenza di uno stake non è strettamente necessaria per la registrazione. In effetti, è possibile registrarsi semplicemente creando un codice univoco e fornendolo come identity commitment. Tuttavia, l'utilizzo di uno stake risulta essere molto utile al fine di prevenire attacchi Sybil, ovvero la generazione di numerosi utenti malevoli da parte di un attaccante.

- **Interazione**: Dopo essersi registrati, gli utenti hanno la possibilità di interagire con il servizio attraverso l'invio di richieste. Ad ogni richiesta, gli utenti sono tenuti a fornire una prova di appartenenza al sistema. Tale prova è generata tramite la tecnologia zk-SNARK, che consente di dimostrare al servizio che l'utente è effettivamente un membro legittimo senza rivelare alcuna informazione sulla sua identità. Inoltre, il protocollo consente di implementare una regola di rate-limiting, che, se non rispettata, porta l'utente a rivelare la propria identità. Ciò consente di scoprire e gestire chi attua comportamenti di spam o DoS, e di procedere alla fase successiva.

- **Punizione**: La tipologia e il grado di punizione dipendono molto dal contesto applicativo. Ad esempio, alcune punizioni o misure di "slashing" possono comportare la rimozione del membro dal gruppo, con conseguente impossibilità di interagire con il sistema e perdita dell'anonimato. In presenza di una stake, è possibile prevedere sanzioni più articolate, come l'inserimento dell'identità virtuale in un registro di esclusione per altri servizi, o la rivelazione dell'indirizzo del portafoglio criptovalute dell'utente e il sequestro dei fondi.

Di seguito si cercherà di guardare a tutto tondo le tecnologie e le idee alla base di zk-SNARK e del protocollo RLN. Inoltre si mostrerà l'implementazione di un piccolo prototipo che utilizza questo protocollo per evitare il sovraccarico di risorse in un servizio basato su API.

[1] `https://rate-limiting-nullifier.github.io/rln-docs/`

Capitolo 1

Stato dell'arte

1.1 Denial-of-Service (DoS)

Gli attacchi di tipo Denial-of-Service (DoS) sono una tipologia di attacchi informatici estremamente diffusi, la cui frequenza è cresciuta negli anni sia in termini quantitativi che qualitativi. Questo incremento è stato determinato dalla relativa facilità con cui è possibile condurre un attacco di questo tipo, nonché dalle sue conseguenze, che in base alle contromisure adottate e alla struttura del servizio attaccato possono essere più o meno gravi. Nella tesi ci concentreremo solamente, sull'analisi della metodologia di protezione denominate rate-limiting. Queste strategie possono risultare estremamente efficaci in alcune situazioni, come dimostrato dall'evento accaduto ad agosto 2022 a Google [1], la quale grazie all'attivazione di un rate-limiter è stata in grado di contrastare con successo il più grande attacco di tipo DDoS a livello applicativo mai registrato. Tale attacco, proveniente da oltre 5.256 sorgenti IP dislocate in 132 paesi differenti, ha raggiunto il picco di 46 milioni di richieste al secondo.

In generale, un una regola per il rate-limiting consiste in un semplice conteggio delle occorrenze delle richieste in un lasso di tempo. Tuttavia, esistono diverse tecniche per misurare e limitare la frequenza di tali richieste, ognuna con i propri usi e implicazioni.

- **Fixed window**: Tra tutte le strategie che vedremo è la più semplice da implementare, consiste nello stabilire un limite massimo al numero di richieste che possono essere inviate in un determinato intervallo di tempo, detto "finestra". Ad esempio, si potrebbe limitare il numero di richieste a 100 ogni minuto. Questo limite viene applicato in modo uniforme all'interno della finestra temporale. Ciò significa che, una volta raggiunto il limite massimo di richieste consentite, l'utente o l'applicazione deve attendere il termine della finestra temporale per poter inviare nuove richieste. Uno svantaggio significativo di questa strategia è la possibile concentrazione di richieste tutte in una porzione della finestra, rischiando un sovraccarico del servizio. Caso notevole è il caso in cui si concentrino tutte le richieste di una finestra al margine della fine e tutte le richieste della finestra successiva al margine dell'inizio questo comporta che il sistema in una durata equivalente a quella della finestra gestisca il doppio

delle richieste consentite.

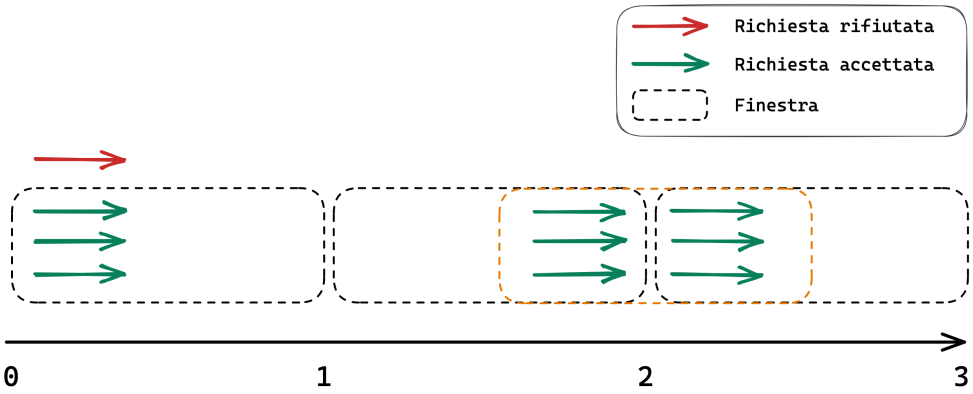


Figura 1.1: Diagramma funzionamento della strategia Fixed window

- **Token bucket**: Il funzionamento del token bucket prevede che non tutte le richieste di un servizio vengano mappate 1:1 con la richiesta di risorse, poiché alcune richieste potrebbero richiedere più risorse di altre. Per questo motivo, viene mantenuto un contatore di risorse, che viene scalato per ogni richiesta, del numero di token necessari per portare a termine il lavoro. Il contatore ha una frequenza di riempimento, ovvero a ogni unità di tempo viene reimpostato al suo valore massimo. Quando un utente o un'applicazione invia una richiesta al sistema, il sistema verifica se ci sono abbastanza token disponibili per soddisfare la richiesta. Se non ci sono abbastanza token, la richiesta viene respinta. In questo modo, le richieste vengono accettate solo se c'è abbastanza capacità per soddisfarle.

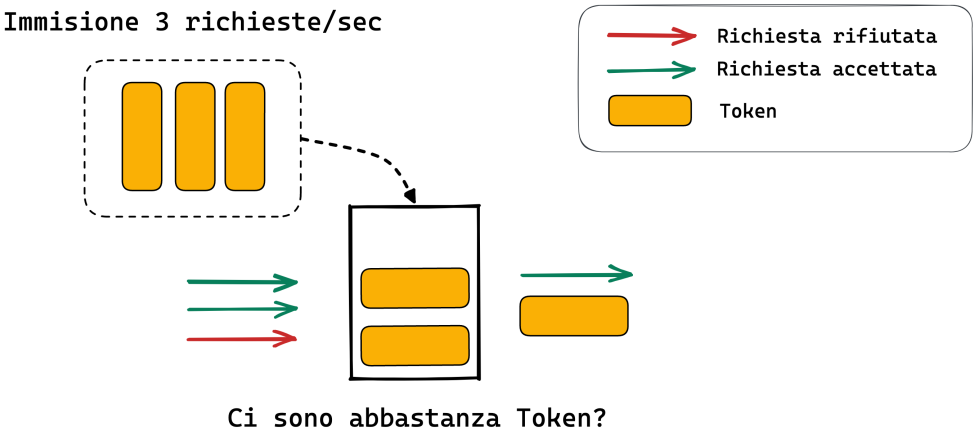


Figura 1.2: Diagramma funzionamento della strategia Token bucket

- **Leaky bucket**: L'idea di base è simile a quella del Token bucket ma invece di rispondere al numero di richieste liberando i token necessari fino a esaurimento, il tasso di elaborazione delle richieste viene regolato in modo uniforme. Questo significa che quando un pacchetto di dati arriva al sistema, se il secchio è già pieno, il pacchetto viene scartato. Nel mentre ad ogni unità di tempo viene elaborata una quantità di richieste corrispondente alla velocità di uscita del

sistema. In questo modo, il flusso di dati in uscita non supera mai una certa soglia.

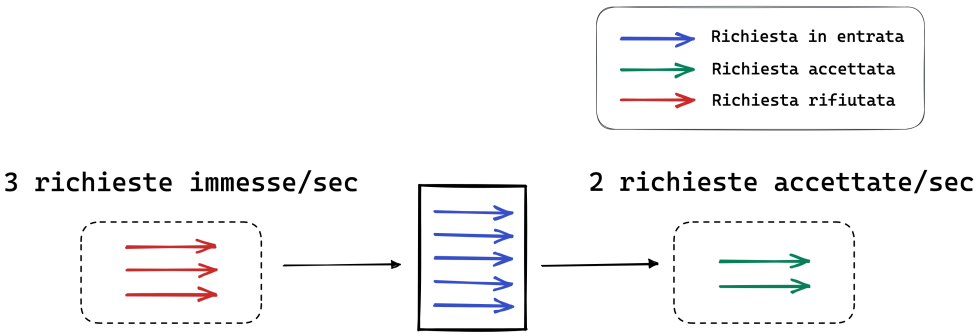


Figura 1.3: Diagramma funzionamento della strategia Leaky bucket

- **Sliding window**: La strategia di sliding window adotta un approccio simile alla Fixed Window, ma con una differenza fondamentale: esamina il tasso di richieste effettuate in un periodo di tempo continuo piuttosto che in intervalli fissi. Ad esempio, se il limite di richieste è fissato a 100 al minuto, la strategia prevede di controllare il numero di richieste effettuate nell'ultimo minuto e, nel caso superi il limite, rifiutare la richiesta. Questo approccio evita il potenziale sovraccarico del sistema alla fine di una finestra con un tempo prefissato, ma richiede la gestione di una finestra scorrevole, aumentando la complessità di implementazione.

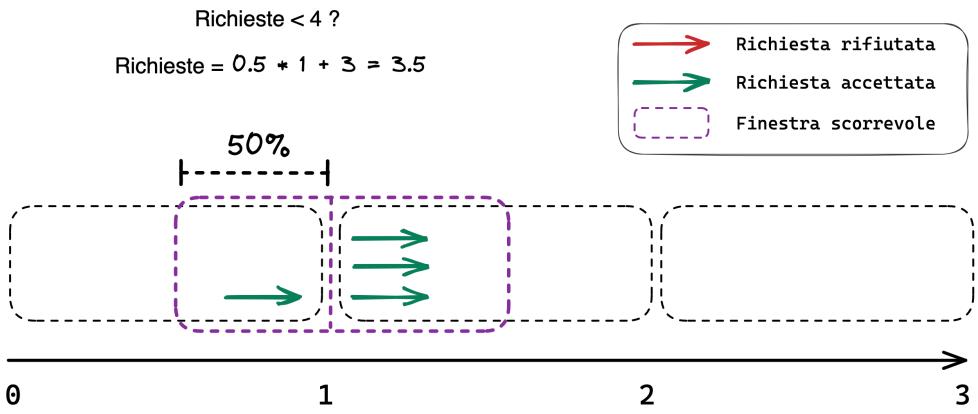


Figura 1.4: Diagramma funzionamento della strategia Sliding window

Vale la pena notare che pur essendo un metodo ampiamente utilizzato il rate-limiting da solo potrebbe risultare insufficiente per garantire un livello di sicurezza adeguato contro gli attacchi DoS, pertanto, è necessario adottare contromisure di diversa natura al fine di garantire una protezione completa.

1.2 zk-SNARK

L'acronimo zk-SNARK rappresenta l'espressione completa "Zero-Knowled Succinct Non-Interactive Argument of Knowledge". Questa tecnologia è stata formalizzata per la prima volta in un articolo scientifico pubblicato nel 2014 intitolato "Succinct non-interactive Zero Knowledge for a von Neumann architecture"[2]. Da allora, sono state apportate diverse migliorie alla tecnologia e sono emerse numerose applicazioni in svariati settori. Le prime applicazioni significative di zk-SNARK sono state implementate nel contesto della Blockchain, che rimane il settore dove la tecnologia è più conosciuta e applicata, per fare alcuni esempi Ethereum[1] una delle Blockchain più accreditate ha iniziato ad implementare la tecnologia nella sua rete dal 2016, e il suo fondatore Vitalik Buterin ha scritto in un articolo sull'argomento: "Perhaps the most powerful cryptographic technology to come out of the last decade is general-purpose succinct Zero Knowledge proofs, usually called zk-SNARKs" [3] (Forse la tecnologia crittografica più potente emersa nell'ultimo decennio è quella delle prove succinte a conoscenza zero di uso generale, solitamente chiamate zk-SNARK.).

Zk-SNARK, come suggerisce il nome, rappresenta una tecnologia fondata sul protocollo crittografico zero-knowledge proof, il quale consente a una parte (il dimostratore) di dimostrare a un'altra (il verificatore) la veridicità di un'affermazione senza rivelare nessuna informazione ulteriore. Inoltre questo processo di dimostrazione, viene effettuato in modo da ottenere una prova, in cui sia la dimensione della prova, che il tempo necessario per verificarla crescono molto più lentamente rispetto al calcolo da verificare e senza la necessità di interazione bidirezionale tra le due parti coinvolte.

Di seguito esamineremo meglio le singole parti che compongono la tecnologia zk-SNARK, analizzando i suoi componenti chiave per ottenerne una panoramica completa.

Notare: Nei prossimi paragrafi lavoreremo nel dominio algebrico dei campi finiti.

1.2.1 Zero-Knowledge

Come accennato precedentemente, Zero-Knowledge proof è un protocollo crittografico in cui una parte dimostra di conoscere una determinata informazione a un'altra parte, senza rivelare alcuna informazione aggiuntiva su di essa. Per esempio, se Peggy vuole dimostrare a Victor di conoscere la password del suo account, senza rivelargli la password stessa, può utilizzare un protocollo di tipo Zero-Knowledge. In questo modo, Peggy può dimostrare a Victor di sapere quale è la password corretta senza rivelarla, proteggendo così la sua privacy e la sicurezza del suo account.

Formalmente le prove di tipo Zero-Knowledge non sono dimostrazioni di carattere matematico, ma probabilistiche, il che significa che c'è sempre una probabilità che un dimostratore disonesto riesca a dimostrare la veridicità di un'affermazione a un verificatore onesto. Esistono tuttavia tecniche per ridurre questa probabilità a valori piccoli a piacere.

[1] `https://ethereum.org`

Il primo articolo che definisce il costrutto è "The Knowledge Complexity of Interactive Proof-Systems"[4] pubblicato nel 1985, dove gli autori introducono il concetto di "Zero-Knowledge proof" come un tipo di "interactive proof systems" [5] (un modello computazionale che simula lo scabio di messagi tra due individui) in cui il verificatore non apprende nulla oltre alla verità dell'affermazione che viene dimostrata.

Per poter ottenere un sistema Zero-Knowledge proof per una particolare affermazione abbiamo bisogno che la prova generata soddisfi le seguenti prorpietà :

- **Completezza**: Se l'affermazione da dimostrare è vera, allora un verificatore onesto (cioè che segue correttamente le regole del protocollo) verrà convinto della veridicità da un dimostratore onesto.
- **Correttezza**: Se l'affermazione è falsa, nessun dimostatore disonesto può convincere un verificatore onesto che essa è vera, se non con una piccola probabilità.
- **Zero-knowledge**: se l'affermazione è vera, nessun verificatore apprende altro se non il fatto che l'affermazione è vera.

Le tre proprietà appena viste descrivono un sistema Zero-Knowledge proof da un punto di vista formale ma per avere una visione più intutiva del protocollo è utile vedere il funzionamento del protocollo attraverso un esempio. L'esempio in questione è tratto da un famoso articolo di Jean-Jacques Quisquater "How to Explain Zero-Knowledge Protocols to Your Children" [6] ne estrapolerò solo le parti esenziali alla tratazione, ma ne consiglio la lettura.

L'esempio riguarda una caverna a forma di anello, nella quale è posta a metà del percoso una porta che impedisce il passaggio, a meno di non conoscere una parola segreta. Supponiamo l'esistenza di due parti, Peggy - che conosce il segreto della porta e agirà da dimostratrice - e Victor - che invece non conosce il segreto e sarà il verificatore. Peggy vuole dimostrare a Victor di sapere come superare la barriera senza però rivelare il segreto. Per fare ciò, Peggy propone a Victor di seguire una strategia: dapprima stabiliscono un nome per identificare i due percorsi (ad esempio, A e B), poi Victor rimane fuori dalla caverna mentre Peggy sceglie uno dei due percorsi.

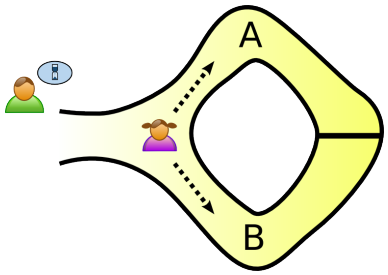


Figura 1.5: Disegno della caverna

Dopo qualche minuto, Victor entra nella caverna e chiama ad alta voce il nome di uno dei due percorsi, a quel punto Peggy dovrà uscire dal percorso chiamato da Victor. Victor accetta la proposta, ma con una condizione: il processo dovrà

Figura 1.6: Interzione tra Peggy e Victor

essereBob ripetuto più volte. Dopo diversi tentativi, Victor si convince che Peggy conosca effettivamente il segreto per superare la barriera. Possiamo calcolare questa probabilità utilizzando la distribuzione binomiale, dove la probabilità di successo p è del 50% e il numero totale di prove n è 10. La probabilità di indovinare correttamente tutte le 10 scelte di Victor è data dalla seguente formula: $P = p^n = 0,5^{10} = 0,0009765625$

L'esempio proposto è molto efficace nel far comprendere come sia possibile dimostrare il possesso o la conoscenza di un informazione senza rivelarla. Nell'articolo citato inoltre si prevendono diversi scenari in cui una o entrambe le parti potrebbero essere disoneste. Per gestire tali situazioni, è possibile applicare una procedura denominata "trusted setup", che verrà approfondita nelle sezioni relativa alle prove non interattive [Non-Interactive].

1.2.2 Succinct

Una dimostrazione di tipo succinct che potremmo tradurre con la parola concisa, è una prova in cui sia la dimensione della prova che il tempo necessario per verificarla crescono molto più lentamente rispetto alla computazione da verificare. Ad esempio se Peggy volesse provare a Victor di possedere un array composto da un milione di elementi, dove ogni elemento è uguale all'indice dell'array più uno.

$$a = [1, 2, 3, ..., 1000000] \tag{1.1}$$

Una prova di tipo succinct, così come definita, non può essere realizzata attraverso un processo di controllo "uno per uno" degli elementi dell'array. Questo perché, se così fosse, si otterrebbe un processo che richiederebbe tanto tempo e spazio di memoria quanto il calcolo effettuato per generare l'array. Una possibile miglioria rispetto al calcolo puntuale potrebbe essere quella di campionare l'array in un punto casuale e controllare se l'elemento selezionato rispetta la regola.
Dopo un singolo campionamento, il grado di fiducia di Victor rispetto ad Peggy sarebbe di soli 0,0001%, ovvero un milionesimo. Se in uno qualsiasi dei campionamenti effettuati Victor dovesse prelevare un numero non congruo, la dimostrazione verrebbe invalidata completamente senza bisogno di ulteriori controlli. È possibile aumentare il grado di fiducia del verificatore eseguendo più controlli, ma anche in questo caso, le prove generate da dimostratori disonesti in cui uno o pochi elementi sono errati all'interno dell'intera prova richiederebbero un numero eccessivo di passaggi per raggiungere un grado di fiducia accettabile, rendendo cosi questo secondo approccio fragile e non attuabile.

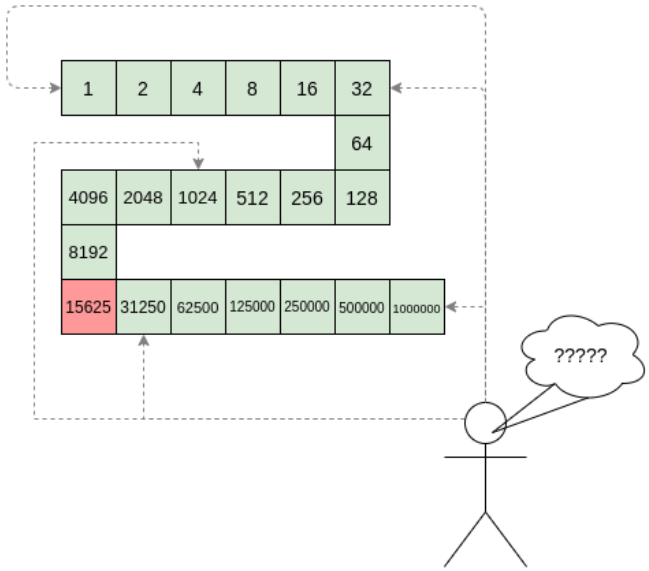


Figura 1.7: Disegno esempio di valutazione a tentativi fallita

Polinomi

Per trovare una soluzione al problema, è necessario fare riferimento ai polinomi, in particolare alla tecnica crittografica nota come "polynomial commitments". Questa tecnica ci consente di generare delle funzioni Hash per i polinomi, chiamate "polynomial commitment", sulle quali è ancora possibile effettuare operazioni algebriche. Ciò significa che possiamo eseguire operazioni sui polinomi senza conoscerli, attraverso i commitment. L'utilizzo di questa tecnica ci fornisce notevoli vantaggi durante la fase di verifica, Infatti la possibilità di verificare le informazioni del dimostratore senza dover operare un controllo puntuale è possibile grazie a una proprietà dei polinomi descritta dal lemma di Schwartz-Zippel; un risultato importante in teoria della complessità computazionale e della teoria degli algoritmi che stabilisce una condizione sufficiente per determinare se un polinomio multivariato non nullo ha radici in un campo finito.

Per comprendere come questo strumento possa esserci utile, partiamo da un equivalenza intuitiva: ipotizzando di essere in possesso di due polinomi $f(x_1, ..., x_n)$ e $g(x_1, ..., x_n)$, chiedersi se $f \equiv g$ è equivalente a chiedersi se

$$p(x_1, ...x_n) = f(x_1, ...x_n) - g(x_1, ...x_n) \equiv 0 \tag{1.2}$$

L'intuizione su cui possiamo basarci per capire l'utilità del lemma per i nostri fini, è che Victor (il verificatore) abbia il polynomial commitment $g(x_1, ..., x_n)$ del polinomio corretto $f(x_1, ..., x_n)$ e voglia verificare che Peggy (il dimostratore) lo conosca.

Sotto quete ipotesi possiamo immaginare un semplice protocollo di esempio dove:

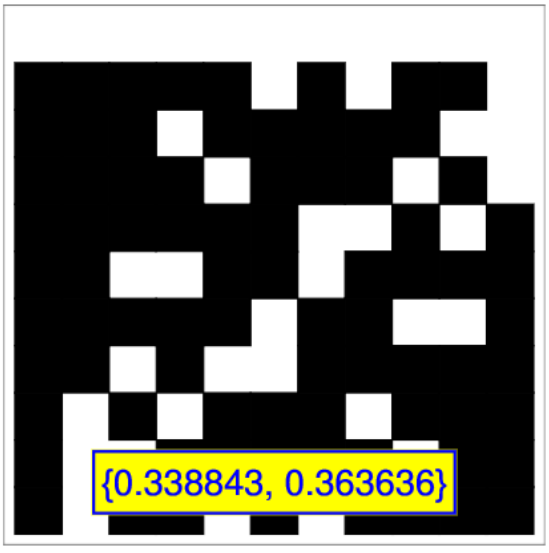
1. Victor sceglie un punto qualsiasi s e valuta il suo polynomial commitment in s, e ottiene $g(s_1, ..., s_n)$.
2. Victor invia s a Peggy che provvederà a valutare il suo polinomio in s, e ottiene $f(s_1, ..., s_n)$.
3. Peggy invia il risultato della sua valutazione a Victor che la confronta con il suo valore, se i valori sono uguali Victor si convince che nel punto s Peggy consce il corretto polinomio.

In questa fase ci potremmo chiedere quale beneficio abbiamo ottenuto passando dalla formulazione dell'array in cui venivano fatti campionamenti casuali, alla formulazione dei polinomi in cui vengono fatte valutazioni del polinomio in variabili casuali. Il beneficio è dovuto al lemma di Schwartz-Zippel, che permette di dimostrare che:

Prendendo un polinomio $p(x_1, ..., x_n)$ di grado $d > 1$ con coefficienti in un campo finto $\mathbb{K}$, prendendo $S \subset \mathbb{K}$ e $r_1, ..., r_n \in S$ scelti in modo arbitrario, allora se $p(x_1, ..., x_n)$ non è il polinomio nullo abbiamo che $p(r_1, ..., r_n) = 0$ con probabilità $\leq \ d/|S|$

Nell'immagine sottostante si può osservare che prendendo un polinomio qualunque nel campo finito $\mathbb{K}$, la probabilità che il polinomio valutato in un punto casuale sia uguale a 0 differisce di poco se calcolata con la definizione classica di probabilità o con il lemma di Schwartz-Zippel, tuttavia il calcolo attraverso la seconda formula è molto più immediato.

$f(x, y) = \prod_{i=0}^{d-1}(x + i \cdot y)$ and its roots in $\mathbb{F}_p^2$, for $p = 11, d = 4$:



Left number: percentage of roots; right number: d/p

Figura 1.8: Esempio applicazione del lemma di Schwartz-Zippel, tratta da [7]

La conseguenza del lemma è che la probabilità di trovare una radice del polinomio in un gruppo di valori appartenenti al campo è inferiore o uguale al rapporto tra il grado del polinomio e la cardinalità del gruppo di elementi selezionati, operazione molto più veloce da calcolare di un controllo puntuale. Inoltre se decidessimo di ripetere il processo di selezione degli $r_1, ..., r_n \in S$ un numero k di volte otterremo che la probabilità che $p(r_1^k, ..., r_n^k) = 0$ sarebbe obbligatoriamente $\leq \ (d/|S|)^k$ e per valori di S abbastanza grandi il il rapporto tende molto velocemente a 0. Intuitivamente il lemma ci dimostra che se un'equazione che coinvolge alcuni polinomi è vera in una coordinata selezionata arbitrariamente, allora è quasi certamente vera per i polinomio nel suo insieme.

Come calcolare i polinomi

Una volta compreso il vantaggio nell'affrontare il problema mediante l'utilizzo di polinomi, possiamo procedere alla descrizione del processo che ci permette di ottenere tali polinomi.

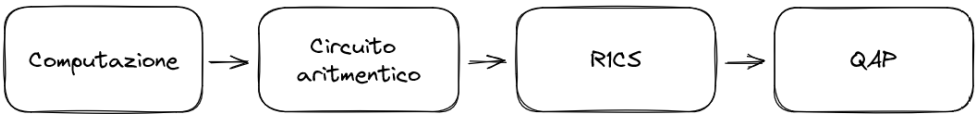


Figura 1.9: Passi da seguire per ottenere un QAP

Per illustrare in dettaglio i passaggi che conducono dalla computazione alla rappresentazione polinomiale desiderata, ovvero la QAP (Quadratic Arithmetic Programs), utilizzeremo un compilatore scritto in Rust[2] chiamato circom[3]

1. La prima fase, ovvero il passaggio dalla computazione al circuito algebrico, può essere un processo non immediato, soprattutto quando si trattano computazioni articolate. A titolo di esempio, consideriamo la computazione $2 * x^2 = 18$ con l'ovvia soluzione da dimostrare $x = 3$. Per ottenere un circuito algebrico a partire da questa computazione, si può procedere come segue:

[2] `https://www.rust-lang.org`
[3] `https://docs.circom.io/`

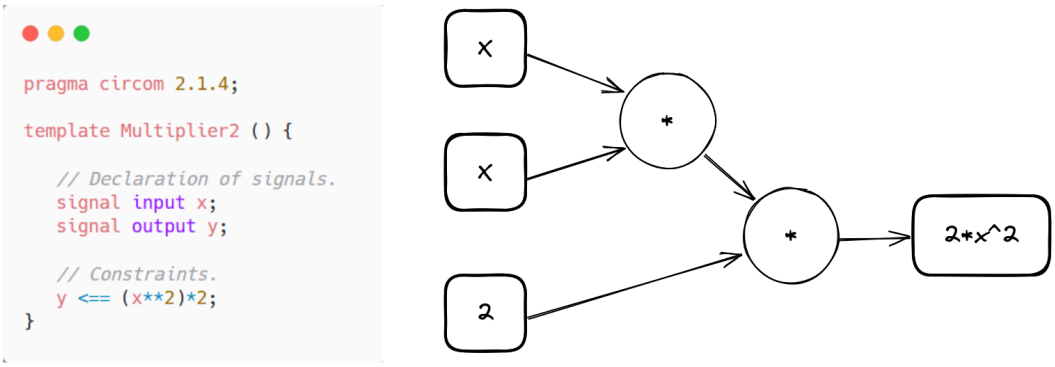


Figura 1.10: Codice esemplificativo per la creazione di un circuito algebrico

2. Ora per proseguire dobbiamo trasformare il nostro circuito aritmetico in un formato chiamato R1CS. R1CS è un formato in cui ogni vincolo (gate del circuito) viene espresso tramite in una terna di vettori (a,b,c) e sul quale viene calcolato un vettore chiamo s che rappresenta la soluzione del sistema di vincoli R1CS. Il vettore s è costruito nel seguente modo:

$$s \cdot a * s \cdot b - s \cdot c = 0$$

dove con il simbolo $\cdot$ si rappresenta il prodotto scalare. Continuando con il nostro esempio dal circuito precedente possiamo calcolare le terne di vettori (a, b, c) per i due gate

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Possiamo osservare che per ognuno dei due gate del circuito, sono stati creati una terna di vettori (a, b, c) di dimensione 4, la dimensione dei vettori è dovuta al numero di variabili del circuito.
Per quanto riguarda il vettore s, esso viene calcolato a partire dal vettore delle variabili del circuito, inserendo al posto di ogni variabile il valore assunto durante la valutazione del circuito. In questo modo, il vettore s rappresenta la soluzione del sistema R1CS.

$$\text{Vettore variabili} = \begin{bmatrix} one, & x, & out, & sym_1 \end{bmatrix}$$

$$\text{Vettore } s = \begin{bmatrix} 1 & 3 & 18 & 9 \end{bmatrix}$$

Possaimo vedere che sym_1 è una variabile creata per spezzare il calcolo di $2 * x^2$ in due diversi vincoli più semplici questa procedura viene chiamata "Flattening" (appiattimento), mentre *one* è una variabile di sistema utilizzata per effettuare operazioni algebriche come la moltiplicazione o la somma per gli elementi del campo. Per verificare che il vettore s sia effettivamente una soluzione del sistema di vincoli R1CS, è possibile calcolare la formula precedente per ogni gate del circuito.

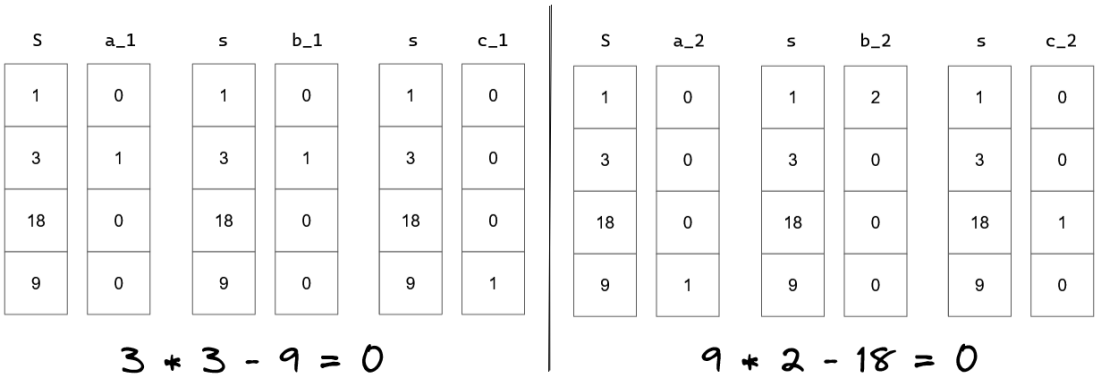


Figura 1.11: Contrllo dei vincoli R1CS

Visto che il risultato dell'operazione è 0 possiamo dire che s è una soluzione del sistema R1CS, e quindi che il nostro valore $x = 3$ che compone il vettore soluzione è corretto. Al pari del calcolo puntuale delgli esempi precedenti questa operazione di verifica per circuiti con molti gate non può essere percorribile.

3. Il processo di trasformazione dal formato R1CS al formato QAP è il più complesso e richiede l'utilizzo di una tecnioca chiamata interpolazione di Lagrange. L'idea principale è quella di costruire dei polinomi tali che se valutati nelle coordinate relative ai vincoli, restituiscano i valori dei vettori corrispondenti. Il processo per ottenere i polinomi si basa su una procedura iterativa su i vettori delle matrici A, B e C. Applicando il processo al nostro esempio otteniamo i seguenti polinomi:

$$\text{Ap} = \begin{bmatrix} 0x^2 + 0x \\ -1x^2 + 2x \\ 0x^2 + 0x \\ 1x^2 - 1x \end{bmatrix} \qquad \text{Bp} = \begin{bmatrix} 2x^2 - 2x \\ -1x^2 + 2x \\ 0x^2 + 0x \\ 0x^2 + 0x \end{bmatrix} \qquad \text{Cp} = \begin{bmatrix} 0x^2 + 0x \\ 0x^2 + 0x \\ 1x^2 - 1x \\ -1x^2 + 2x \end{bmatrix}$$

Questi polinomi sono costruiti in modo tale che valutandoli in x = 1 (coordinata del primo vincolo) otteniamo i valori corrispondenti ai vettori del primo vincolo

$$\text{Ap} = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \qquad \text{Bp} = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \qquad \text{Cp} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$$

grazie a questa formato ora se volessimo verificando i vincoli tramite la formula precedente avremmo

$$A(x) * B(x) - C(x) = P(x)$$

dove $P(x)$ non è obbligatoriamente il polinomio nullo. Tuttavia Se la dimostrazione è corretta $P(x)$ deve avere delle radici in tutti le coordinate corrispondenti ai vincoli. Per verificarlo non abbiamo bisogno di controllare tutti i vincoli perché lavorando con i polinomi possiamo sfruttare il lemma di Schwartz-Zippel per valutare la condizione molto più velocemente.

Valutazioni sul protocollo

Fino ad ora abbiamo descritto un processo che ci consente di trasformare i nostri vincoli numerici in polinomi, permettendoci di calcolare con grande velocità e un elevato grado di certezza la validità dei vincoli imposti. Tuttavia, il protocollo spiegato in precedenza presenta due falle:

1. **Arbitrarietà**: Nel secondo passaggio del protocollo, Victor condivide con Peggy un punto s, che è stato scelto casualemten per valutare il polinomio. Tuttavia, passando s in chiaro ad Peggy ci si espone al rischio che Peggy costruisca un polinomio ad hoc, che soddisfi i vincoli solo nel punto specifico scelto, permettendole così di convincere Victor di pur non conoscendo il polinomio. Ciò violerebbe il principio di correttezza dei protocolli Zero-Knowledge proof.
2. **Verificabilità**: la capacità di Peggy di valutare correttamente il polinomio in s non garantisce che essa abbia utilizzato effettivamente il polinomio p per restituire il risultato a Victor, In altre parole, non è garantito che Peggy abbia utilizzato il polinomio descritto dal polynomial commitment per raggiungere il risultato atteso.

Per affrontare questi due problemi, vengono utilizzate altre tecniche matematiche, denominate Homomorphic Encryption (HH) per risolvere il primo problema e Knowledge of Coefficient (KC) per risolvere il secondo. In questa sede, si accennerà brevemente a come è possibile ottenere un Homomorphic Encryption in grado di soddisfare la proprietà di correttezza, mentre la trattazione del KC non verrà approfondita. È possibile trovare una introduzione ad entrambe le tecniche in questa serie di articoli [8]

Homomorphic Encryption

L'idea alla base dell'Homomorphic Encryption è quella di costruire un'operazione simile all'operazione di hashing con la proprietà di poter applicare delle operazioni aritmetiche ai valori criptati senza decifrarli, ovvero ottenere una funzione $E(x)$ che soddisfi tali proprietà :

1. **Difficoltà di inversione**: Dato un $E(x)$ sia complesso risalire a x
2. **Resistenza alle collisioni**: Se $x \neq y \Rightarrow E(x) \neq E(y)$
3. **Preserva operazioni aritmetiche**: Se si conoscono E(x) ed E(y), si può generare delle espressioni aritmetiche in x e y. Per esempio, si può calcolare E(x+y) da E(x) ed E(y)

1.2.3 Non-Interactive

Tutti i protocolli Zero-Knowledge proof esaminati fino ad ora erano di tipo interattivo. Questo significa che per generare e verificare la prova, le due parti coinvolte, ovvero il verificatore e il dimostratore, devono interagire in modo bidirezionale. Ad esempio, nel caso della "grotta di Alibaba", Victor, il verificatore, indicava il passaggio da intraprendere e Peggy, il dimostratore, che "rispondeva" uscendo dalla giusta direzione. Mentre nel caso dei polinomi, il verificatore forniva le coordinate del punto in cui valutare il polinomio e il dimostratore rispondeva con la valutazione corretta. In alcune situazioni tuttavia, può risultare vantaggioso utilizzare le cosiddette NIZKP, ovvero le Non-Interactive Zero-Knowledge proof, in cui il dimostratore trasmette al verificatore una prova contenente tutte le informazioni necessarie per la verifica, senza richiedere ulteriori interazioni tra le parti.

Questa metodologia presenta numerosi vantaggi sia in termini di prestazioni, poiché consente di costruire le prove più velocemente senza dover attendere la risposta e il tempo di comunicazione, sia in termini di sicurezza, in quanto l'assenza di uno scambio di messaggi elimina il rischio di intercettazioni da parte di individui malintenzionati. Inoltre, l'utilizzo di prove NIZKP potrebbe risultare obbligato in quei contesti applicativi in cui la comunicazione in tempo reale non è possibile. L'adozione di questo tipo di prove comporta però un aumento della complessità, poiché, essendo privi di interazione, tutto il lavoro di generazione e preparazione alla verifica viene svolto dal dimostratore e affinché il dimostratore non sia arbitrariamente libero di agire, generando prove scorrette, è necessario vincolarlo al rispetto delle regole.

A tal fine, si utilizza una tecnica nota come "trusted set-up" che consente di generare una Common Reference String (CRS), ovvero un dato pubblico conosciuto sia dal dimostratore che dal verificatore usato per aggiungere casualità all'interno della generazione della prova. Sempre riferendoci ai due esempi precedenti, la CRS ottenuta tramite un trusted set-up consentirebbe di rendere le scelte di Victor nella caverna il più casuali possibile, impedendo ad Peggy di individuare dei pattern o a entrambi di collaborare per ingannare un terzo osservatore e nel caso dei polinomi, invece, consentirebbe di rendere aleatoria la scelta del punto in cui valutare il polinomio.

La procedura di costruzione della CRS rappresenta un passaggio critico per la robustezza del protocollo e dipende fortemente dal tipo di cerimonia di trusted set-up scelta.

Modello di affidabilità

Una cerimonia di trusted set-up è una procedura che viene eseguita una sola volta per generare un CRS che poi potrà essere utilizzata ogni volta che viene eseguito il protocollo. Il termine "trusted" deriva dal fatto che una o più persone devono partecipare alla cerimonia in modo "fidato" contribuendo alla creazione della CRS con dei segreti anche chiamati "toxic waste" che non appena inseriti nella procedura dovranno essere eliminati.

Una volta completata l'elaborazione e dopo aver eliminato i segreti, il risultato dell'elaborazione (la CRS) diventa irreversibile e non può essere riprodotta volontariamente, il che garantisce la sicurezza del protocollo.

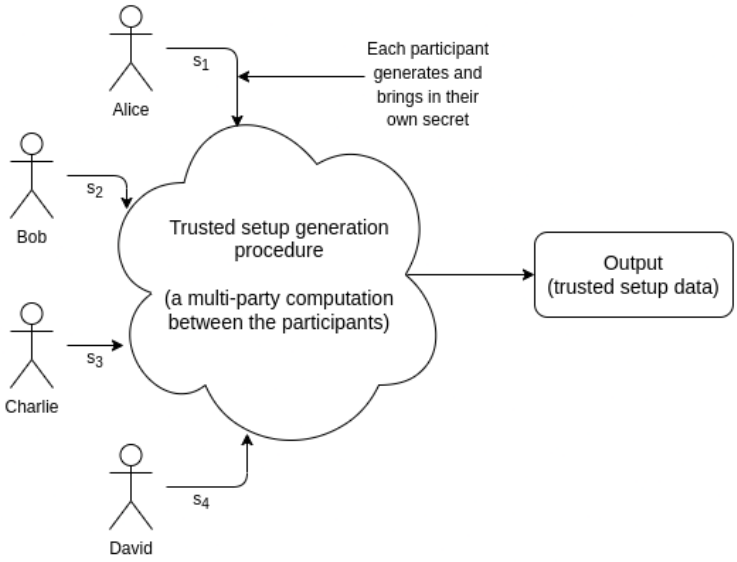


Figura 1.12: immagine generica di una cerimonia di trusted set-up. tratta da [9]

Se le parti coinvolte non si comportano in modo leale e conservano o pubblicano i loro segreti, la validità della cerimonia dipenderà dal modello di affidabilità utilizzato dall'algoritmo di trusted set-up. Infatti poiché i protocolli operano in diversi domini applicativi, non tutti utilizzano lo stesso modello di affidabilità.

Una trattazione completa dei vari modelli di affidabilità esula dai miei scopi, tuttavia, riducendo la classificazione a due sole proprietà degli algoritmi, è possibile descrivere brevemente alcune tipologie utili per la trattazione successiva.

Le due proprietà su cui si basa la classificazione sono il numero di parti fidate che partecipano alla cerimonia e il numero di parti che devono comportarsi "lealmente" per garantire la segretezza della CRS.

In base a queste proprietà, si possono individuare tre gruppi di interesse (non sono gli unici):

- **1 - 1**: Questo modello è simile a quello utilizzato nei servizi centralizzati, in cui ci si affida a un singolo individuo fidato per la gestione dei dati.

- **N/2 - 1**: La maggior parte delle blockchain utilizza questo modello, in cui il processo viene considerato valido se la maggioranza dei partecipanti rimane onesta.

- **1 - N**: In questo modello, è sufficiente che almeno una delle parti rimanga onesta durante la cerimonia per garantire la segretezza della CRS.

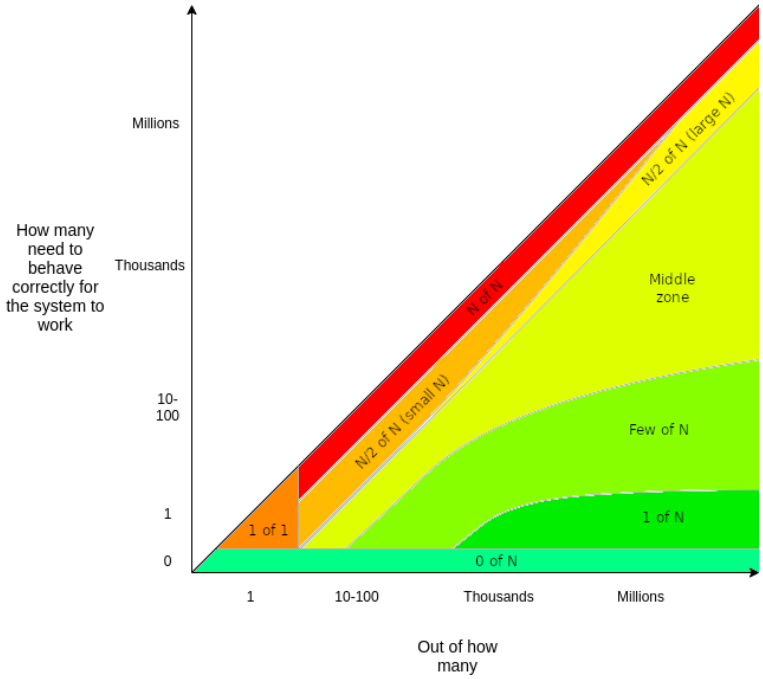


Figura 1.13: immagine modelli di di affidabilità, tratta da [10]

MPC (multi-party computation)

La tecnologia MPC (multi-party computation), anche se non è stata ancora esplicitamente menzionata, è alla base dei trusted set-up. Infatti per generare una CRS sicura e non riproducibile, è necessario coinvolgere molte parti e garantire che nessuno possa ricavare informazioni sui segreti dei partecipanti o sulla CRS stessa. Per questo motivo la procedura non si limita a una semplice combinazione dei segreti, ma può essere vista come una "black box" che applica elaborazioni ai segreti in input e restituisce la CRS in output.

Ogni algoritmo di MPC deve soddisfare due proprietà fondamentali:

1. Nel caso in cui uno o più partecipanti disonesti decidano di rivelare il loro segreto, il protocollo MPC deve impedire loro di obbligare i partecipanti onesti a rivelare le loro informazioni riservate o influenzare il risultato finale.

2. Nessuno dei partecipanti deve essere in grado di dedurre i segreti degli altri partecipanti dagli elaborati del protocollo. In altre parole, il calcolo effettuato dall'algoritmo non deve fornire alcun indizio sui segreti che hanno portato al risultato.

Groth16

Groth16 è un protocollo Zero-knowledge proof proposto da Jens Groth nell'articolo "On the Size of Pairing-based Non-interactive Arguments" [11] pubblicato nel 2016, questo protocollo è diventato molto diffuso grazie alla sua grande efficienza dal punto di vista computazionale, che permette di creare zk-SNARK estremamente

performanti sia in termini di tempo che di memoria. Tale tecnologia è alla base di progetti come Z-cash[4].

Il trusted set-up su cui si basa Groth16 richiede due fasi distinte:

1. Nella prima fase si applica una procedura di trusted set-up chiamata "power of tau" che consiste in una algoritmo MPC basato sul modello di affidabilità 1-N che utilizzando le curve ellitice permette di generare una CRS. Il processo viene chiamato powers of tau o Pot perchè duarnte le sue fasi vengono usate le potenze di un numero tao generalmente il 2 assime ai contributi dei partecipanti per generare punti casuali e ottener una CRS. il numero N di partecipati per una cerimonia di tipo power of tau puo essere dell'ordine del migliaio di partecipanti rendendo cosi la cerimonia molto sicura, in quanto basta che almento uno di essi mantenga il segreto per far si che la CRS rimanga valdia. Questo procedimento deve essere svolto una volta sola e deve essere combinato con il processo della fase successiva.

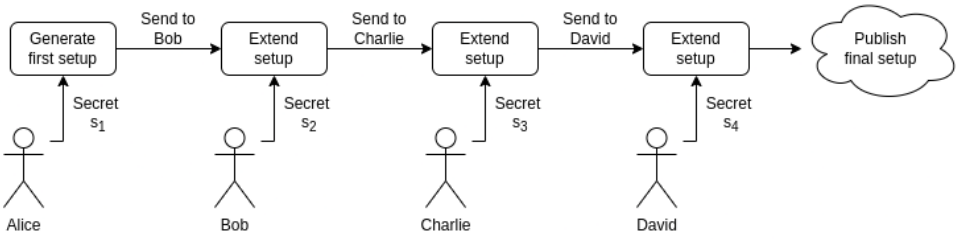


Figura 1.14: immagine generica di una di power of tau. tratta da [9]

2. Nella seconda fase invece viene elaborato un trusted set-up dipendente dalla computazione che si vuole provare (o meglio dipendete dal circuito). Quindi questa fase deve essere riprodotta dal dimostratore ogni volta che la dimostrazione cambia. Non entrerò nei dettagli implementativi, ma in questa fase viene utilizzata l' euristica di Fiat-Shamir , che permette di scegliere il punto in cui valutare il polinomio, che descrive la computazione, senza che il dimostratore possa interferire. Il punto viene scelto sulla base dei calcoli fatti per ottenere il polinomio stesso. È intuitivo comprendere che se il dimostratore volesse falsificare la prova, non potrebbe farlo perché per scegliere arbitrariamente un polinomio pere soddisfare un determinato vincolo, dovrebbe conoscere il valore di s ma essendo s derivato dal polinomio stesso questo non è possibile.

Indubbiamente il fatto che la robustezze della tecnologia zk-SNARK sia subordinata a una procedura cosi elaborata è uno svantaggio, infatti esistono, protocolli di Zero-knowledge proof che non necessitano di una fase di trust set-up iniziale. Bulletproof e le STARK (Scalable Transparent Arguments of Knowledge), ad esempio, non richiedono alcuna configurazione di questo tipo, ciò nonostante, le tecnologie zk-SNARK basate su Groth16 risultano essere più performati in termini di efficienza il che le rende molto più diffuse delle altre tecnologie.

[4] `https://z.cash/`

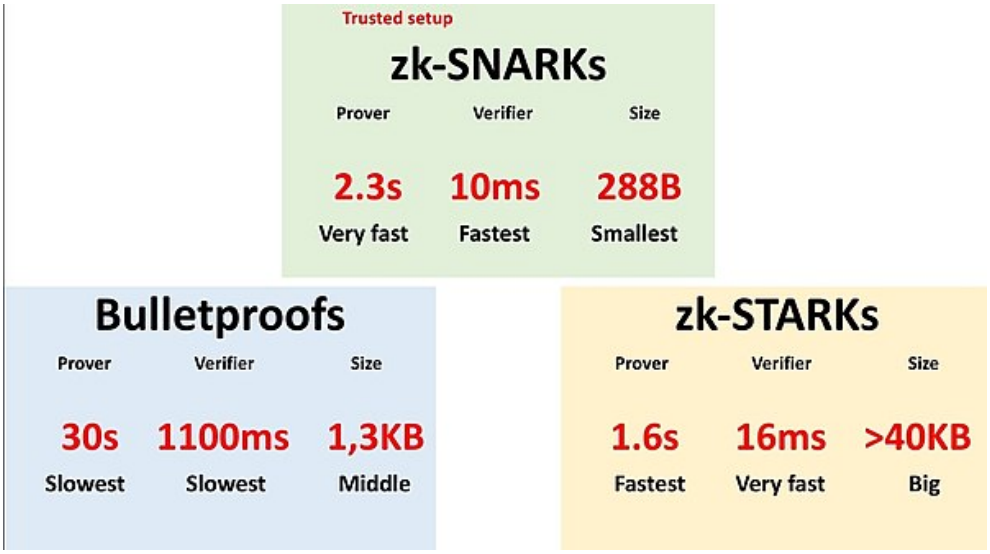


Figura 1.15: immagine prestazioni di tecnologie NIZKP a confronto. tratta da [12]

Capitolo 2

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 utilizzando la tecnologia zk-SANRK. 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" [13]. Attualmente, RLN fa parte di (PSE) Privacy & Scaling Explorations [14], 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.

Nelle prossime sezioni, verranno dettagliate le fasi del protocollo e le tecnologie utilizzate. Tuttavia, prima di addentrarci, potrebbe essere utile fornire una descrizione generale e concisa del protocollo per comprendere meglio dove e perché vengono impiegate le tecnologie che andremo a esaminare. Il protocollo RLN è suddiviso in tre fasi. Descriveremo brevemente le prime due fasi, in quanto la terza diventa ovvia una volta compreso il funzionamento delle prime due.

Regitrazione: Questa fase consente agli utenti di registrarsi al servizio che utilizza il protocollo RLN. In particolare, agli utenti viene richiesto di generare una chiave privata che rappresenta il loro segreto. Le strategie per la generazione della chiave possono variare a seconda del contesto applicativo e possono essere correlate alla presenza o meno di una stake. Una volta ottenuta la chiave privata, a questa viene applicata una funzione di hash per generare un "identity commitment", ovvero un dato che può essere reso pubblico in quanto non consente di ricostruire la chiave privata, ma che ne garantisce l'identificazione univoca. Durante questa fase, l'identity commitment generato viene inviato al servizio, che lo inserisce in una particolare struttura dati chiamata albero di Merkle. In questo albero, il sistema salva e mantiene tutti gli identity commitment degli utenti registrati. La registrazione rappresenta una fase cruciale del protocollo, in quanto solo dopo di essa gli utenti

saranno in grado di interagire con il sistema, generando delle Zero Knowledge proof che attestino la loro appartenenza al servizio questa prova è anche chiamata "proof of membership". Al fine di limitare gli attacchi DoS o Sybil, la fase di registrazione sarà molto più efficace se sottoposta al rispetto di determinate specifiche di difficile riproducibilità da parte degli utenti. Per questo, in alcuni casi, è possibile legare la chiave privata a una stake.

Interazione: Questa fase rappresenta sia quella che consente l'interazione tra gli utenti registrati e il sistema, sia quella che implementa la regola di rate limiting. La chiave privata dell'utente, prima che interagisca con il sistema, viene suddivisa in n parti in modo che ad ogni interazione il sistema riveli all'utente solo una porzione della chiave. Il sistema non sarà in grado di ricostruire la chiave privata dell'utente dalle singole porzioni, a meno che queste non raggiungano un numero k stabilito dalla regola di rate limiting. In caso contrario, se il sistema dispone di k parti della chiave, è in grado di ricavare l'identità dell'utente. Questo processo è attuabile grazie ad un algoritmo basato sull'interpolazione di Lagrange, chiamato Shamir Secret Sharing (SSS).

2.1 Strumenti

2.1.1 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 composta dalle foglie, che rappresentano i dati a cui è stata applicata una funzione di hash e dai nodi, che sono ottenuti uno dopo l'altro 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:

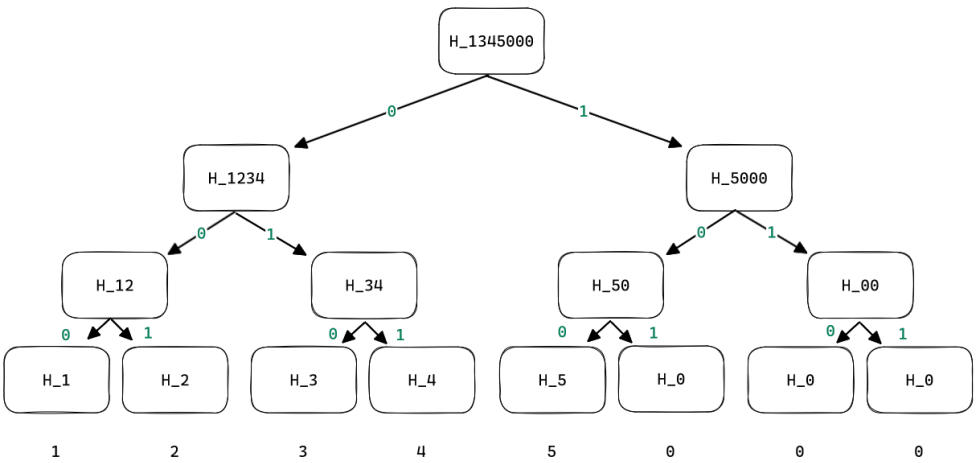


Figura 2.1: Esempio struttura albero di Merkle binario

Possiamo notare che la radice degli alberi di Merkle 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 necessità 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. Una volta ottenuta la radice basterà confrontarla con la radice corretta per convicersi della presenza dell'elemento o meno nella struttura.

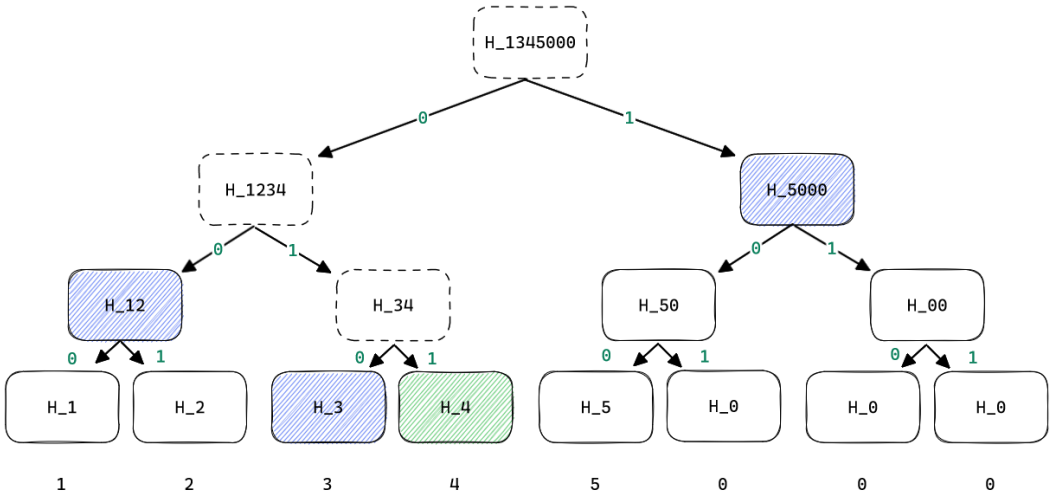


Figura 2.2: Esempio Merkle proof

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.

2.1.2 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, sfruttano metodologie come la ripetizione di operazioni bit-wise, che tendono se trasforamte con R1CS ad aumentare considerevolmente la dimensione dei vincoli del circuito. 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 Hash come Pedersen, Rescue-x^5 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 verrà utilizzata nella costruzione del circuito per il protocollo RLN. Di seguito vengono mostrate delle tabelle contenteti un confronto delle prestazioni tra le più note funzione di Hash per le tecnologie Zero Knowledge e la funzione Sha-256, tabelle tratte da "POSEIDON: A New Hash Function for Zero-Knowledge Proof Systems" [15]. 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 2.3: Confronto con altri algoritmi Hash

2.1.3 Nullifier

I "nullifier" in crittografia 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 elettore voti più di una volta, mentre nel campo delle criptovalute è fondamentale evitare che la stessa moneta, venga spesa più volte. Il problema si risolve utilizzando 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.

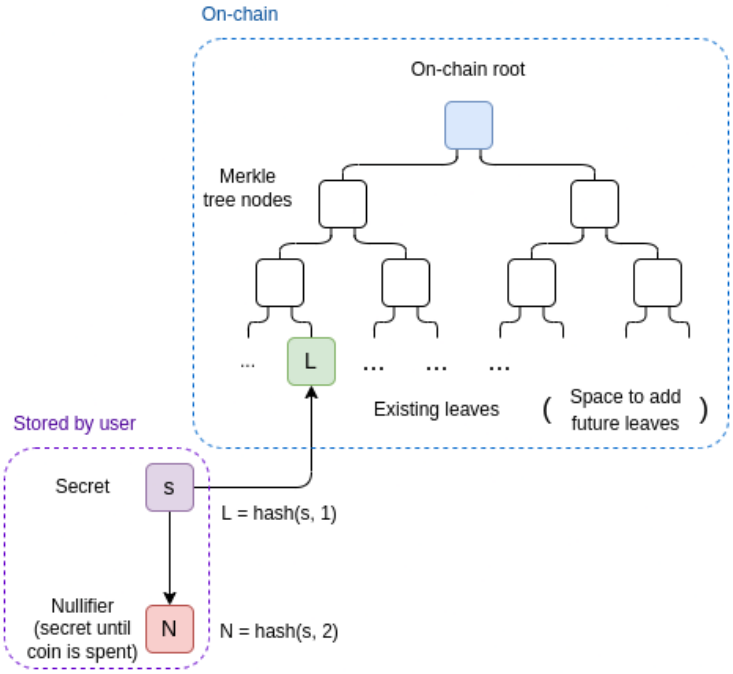


Figura 2.4: Rappresentazione della creazione di un nullifier, tratta da [16]

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

Dopo aver visto le tecnologie utilizzate nel protocollo RLN, è possibile procedere con la descrizione delle sue fasi.

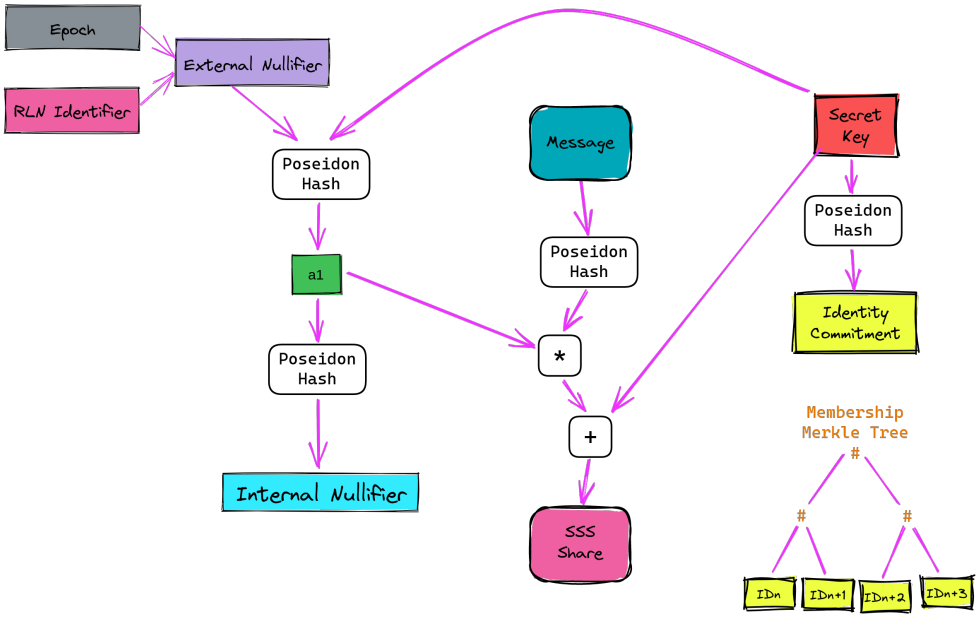


Figura 2.5: Diagramma funzionametno RLN, tratto da [17]

2.2 Registrazione

La registrazione consiste nella prima fase del protocollo RLN, in questa fase gli utenti, generano un identity commitment partendo dalla propria chiave privata, attraverso l'utilizzo dell 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 Merkle 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 requisiti, come il posseso di identità digitali (CIE o SPID), 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 identity commitment a lui associato all'inetrno di alcuni gruppi di reputazione. Questi gruppi (alberi di Merkle) sono divisi in base al grado di reputazione degli utenti. Attualmente il progetto Interep[1] permette di estrapolare la reputazione di un utente dagli account social di Github, Twitter e Reddit.

2.3 Interazione

Passata la fase di registrazione, gli utenti avranno la possibilità di effettuare richieste, provando di appartenere ai membri del servizio, senza rivelare la loro identità utilizzando zk-SNARK. Ora ci chiediamo come possiamo 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 in questione.

La procedura per dividere e ricostruire il segreto si basa ancora una volta sull'utilizzo dei 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 ($k + 1$ richieste).

[1] urlhttps://interep.link/

Vediamo un esempio di funzionamento dell'algoritmo, 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, questo valore viene utilizzato per proteggere gli utenti che utilizzino la stessa chiave private in più servizi che apllicano RLN, infatti grazie a questo paramentro anche se si usasse la stessa chiave privata per costruire il polinomio si otterrbbero valori differenti in fase di valutazione. Proseguiamo con l'identificazione del polinomio che dovrà essere ricostruito dal verificatore (il sitema), il polinomio in questione dovra essere di primo grado ($k = 1$) in quanto voglaimos che con due richieste ($k + 1$ punti) si possibile ricostruirlo

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

Notiamo che il polinomio valutato in 0 vale a_0 ovvero la nostra chiave privata, mentre a_1 è definito come $a_1 = Poseidon(a_0, externalNullifier)$ che permette di variare il polinomio in base all'epoca in cui viene fatta la richietsa, in questo caso specifico a_1 è il coefficente angolare della retta identificata dal polinomio. Quando un utente invia una richiesta al sistema vengono calcolate due coordinate: $x = Poseidon(richiesta)$ e $y = A(x)$ che identificano un punto sulla retta. Se un utente malitenzionato inviasse un altro nella setta epoca otterrebbe le nuove cordinate $x_2 = Poseidon(richiseta_2)$ e $y_2 = A(x_2)$ apprtenenti alla stessa retta e il sitema sarebbe in grado di ricostruire il polinomio interpolando i due punti. Nel caso di un polinomio di primo grado, la procedura di interpolazione è immediata

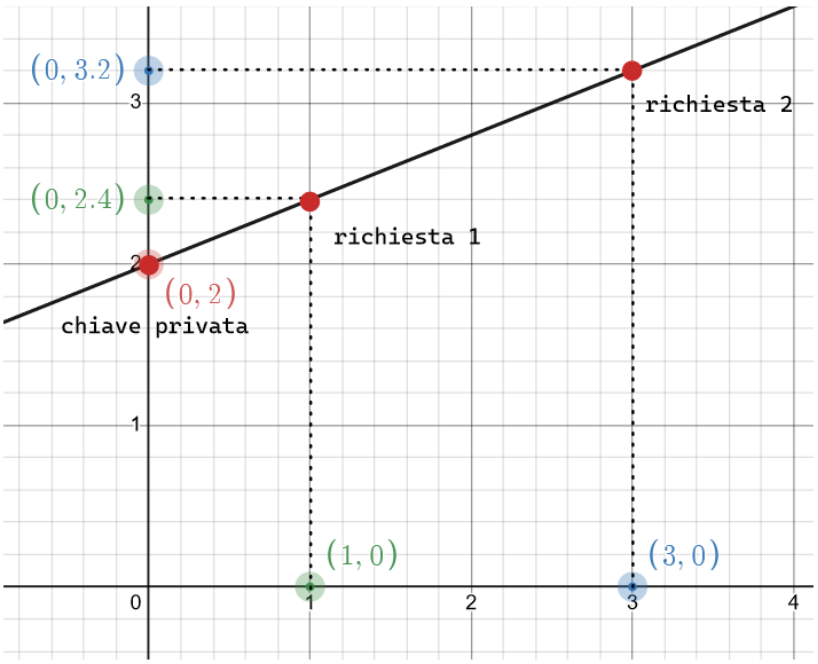


Figura 2.6: Grafico SSS pilinomio primo grado

2.4 Punizione

L'ultima fase del protocollo consiste nella punizione dell'utente malintenzionato. Questa procedura è fortemente legata al contesto applicativo. L'idea generale è che se sono presenti k porzioni diverse per ricostruire a_0 dalle cordinate x e y, allora l'identità dell'utente può essere rivelata e l'utente può essere rimosso dall'albero di Merkle ricostruendo il suo l'identity commitment. Questo comporta l'azzeramento della foglia che contiene l'identity commitment dell'utente all'interno dell'albero. Inoltre, a seconda dell'applicazione, la chiave privata può anche essere usata per sequestrare la stake fornita dall'utente.

Prototipo

In questo capitolo, presenteremo le parti significative di un prototipo di applicazione che utilizza il protocollo RLN e la tecnologia zk_SNARK per applicare una regola di limitazione della velocità in un sistema centralizzato, dove Client e Server interagiscono in completo anonimato.

Al fine di concentrarci sulle dinamiche del protocollo per comprenderne il funzionamento, ho scelto di semplificare la parte strutturale del prototipo, che è composta da un Server che espone un servizio di registrazione e di interazione, e alcuni Client che comunicano con il Server attraverso un canale di comunicazione basato su socket.

Si tenga presente che il protocollo non ha pretese di essere un progetto realmente funzionante, ma piuttosto di essere un utile strumento per coloro che desiderano acquisire familiarità con le tecnologie zero knowledge e applicarle concretamente. Ciò nonostante sarà possibile valutare i benefici e le criticità che queste tecnologie portano in campo, mediante il calcolo delle prestazioni e l'analisi del codice.

2.5 Tecnologie e struttura

Per lo sviluppo del prototipo, ho attuato la scelta di utilizzare il linguaggio di programmazione TypeScript. Questa decisione è stata presa in considerazione delle specifiche necessità del progetto, e in particolare delle librerie sviluppate e attive per la tecnologia zk-SNARK e per il protocollo RLN. La struttura del progetto è un monorepo (unica repository con molteplici progetti) che contiene sia i dati e il codice del Server che quelli reltiva al Client. Entrambi i componenti, utilizzano Node.js come ambiente di esecuzione. Infatti dopo la compilazione del codice TypeScript in JavaScript, il codice viene eseguito tramite il runtime Node.js. Le librerie utilizzate sono le seguenti:

- **Circom 2**: usata per la generazione e la compilazione dei circuiti del progetto. Circom consiste di un compilatore Rust per i circuiti scritti l'inguaggio circom (omonimo della libreria). Inotre comprende anche diversi template di circuiti che assolvono specifiche funzioni molto utli come l'implementazione della funzione di hash Poseidon.
- **Socket.IO**: è una libreria JavaScript che consente di creare e gestire connessioni in tempo reale tra il Client e il Server. Si basa sul protocollo WebSocket e permette di creare una comunicazione bidirezionale tra il Client e il Server implementando anche un sistema basato sugli eventi, ispirato alla classe EventEmitter di Node.js

- **RLN Circuits**: in questa libreria sono preseti i circuiti che implementano le funioni principarli del protocollo RLN. In particolare, sono presenti i circuiti per la registrazione, la verifica, la punizione e alcuni circuiti di supporto.

- **RLNjs**: è una libreria scritta in TypeScript che implementa la logica del protocollo RLN. Sfrutta i circuiti messi a disposizione dalla libreria precedente per implementare la logica di verifica e generazione delle prove.

Inoltre ho utilizzato il package manager ufficiale di Node.js, npm, per installare rapidamente e facilmente le librerie e i moduli di cui hai avuto bisogno e Eslint uno strumento di analisi del codice statico che aiuta a individuare e correggere errori e eventuali vulnerabilità del codice.

2.6 Circuiti

Come spiegato nella sezione relativa alla proprietà succint di zk-SNARK per potre creare un programma che usi questa tecnologia, abbiamo bisogno di formulare le porzioni di logica del programma che vogliamo provare (dimostrare) attraverso dei circuiti algebrici. Le parti significative del codice di cui vogliamo creare una Zero Knowledge Proof sono l'appartenenza all'albero di Merkle e la corretta costruzione delle porzione di segreto che ogni Client rilascia al Server durante l'interazione. Per questo motivo abbiamo bisogno di due circuiti:

1. **proof of membership**:

```
template MerkleTreeInclusionProof(n_levels) {
    signal input leaf;
    signal input path_index[n_levels];
    signal input path_elements[n_levels][1];
    signal output root;
    component hashers[n_levels];
    component mux[n_levels];
    signal levelHashes[n_levels + 1];
    levelHashes[0] <== leaf;

    for (var i = 0; i < n_levels; i++) {
        // Should be 0 or 1
        path_index[i] * (1 - path_index[i]) === 0;

        hashers[i] = HashLeftRight();
        mux[i] = MultiMux1(2);

        mux[i].c[0][0] <== levelHashes[i];
        mux[i].c[0][1] <== path_elements[i][0];
        mux[i].c[1][0] <== path_elements[i][0];
        mux[i].c[1][1] <== levelHashes[i];

        mux[i].s <== path_index[i];
        hashers[i].left <== mux[i].out[0];
        hashers[i].right <== mux[i].out[1];
        levelHashes[i + 1] <== hashers[i].hash;
    }
    root <== levelHashes[n_levels];
}
```

Figura 2.7: codice circuito proof of membership

Nel codice in figura possiamo vedere il circuito scritto in codice circom che implementa la ricerca di un elemento all'interno di un albro di Merkle, il circuito prende in ingresso tre paramentri: la prfonditò dell' albero, il valore della foglia da cercare, il percorso binario, dove 0 indica il ramo di sinistra e 1 indica il ramo di destra, che bisogna seguire partendo dalla radice per raggiungere la foglia e le altre foglie dell'albero. Il circuito restiturisce la radice dell'albero. I componenti $HashLeftRight()$ e $MultiMux1()$ sono due componenti ausiliari che servono rispettivamente a apllicare la funzione hash ai nodi figli e a selezionare i nodi corretti in base al livello dell'albero in cui ci troviamo.

2. **verifica delle porzioni del segreto**:

```
template CalculateIdentityCommitment() {
    signal input identity_secret;
    signal output out;

    component hasher = Poseidon(1);
    hasher.inputs[0] <== identity_secret;

    out <== hasher.out;
}

template CalculateExternalNullifier() {
    signal input epoch;
    signal input rln_identifier;

    signal output out;

    component hasher = Poseidon(2);
    hasher.inputs[0] <== epoch;
    hasher.inputs[1] <== rln_identifier;

    out <== hasher.out;
}
```

Figura 2.8: codice circuito proof of membership

In questa prima porzione di codice possiamo vedere due funzioni fondamentali del protocollo RLN, ovvero la funzione $CalculateIdentityCommitment()$ che permette di verificare che l'identity commitment venga effettivamente calcolato a partire dalla chiave privata e $CalculateExternalNullifier()$ che è quel circuito ci permette di ottenere un nullifiere diverso per singolo mesaggio.

```
21 template CalculateA1() {
22     signal input a_0;
23     signal input external_nullifier;
24
25     signal output out;
26
27     component hasher = Poseidon(2);
28     hasher.inputs[0] <== a_0;
29     hasher.inputs[1] <== external_nullifier;
30
31     out <== hasher.out;
32 }
33
34 ...
35
36     component external_nullifier = CalculateExternalNullifier();
37     external_nullifier.epoch <== epoch;
38     external_nullifier.rln_identifier <== rln_identifier;
39
40     component a_1 = CalculateA1();
41     a_1.a_0 <== identity_secret;
42     a_1.external_nullifier <== external_nullifier.out;
43
44     y <== identity_secret + a_1.out * x;
45
46 ...
```

Figura 2.9: codice circuito proof of membership

Mentre in questa porzione vediamo ancora una funzione atomica ovvero CalculateA1() che permette di ottenere il valore a1 che fa variare il polinomio di verifica a seconda dell' external_nullifier e della chiave privata seguendo la formula vista in precedenza $a_1 = Poseidon(a_0, externalNullifier)$ e in fine vediamo una porzione di codice dove tutte le funzioni precedenti vengono utilizzate per verificare che il segreto sia stato costruito correttamente.

Ricordo che il codice visto fino ad adesso non rappresenta la logica del protocollo ma solamente, i circuiti algebrici che verranno trasformati in vincoli per la verifica che il Client soddisfi e rispetti pur rimanendo anonimo le regole del servizio.

Per ottenere i file che verranno utilizzati da Server e Client per provare e verificare le dimostazione, bisogna compilare i circuiti utilizzando circom. Alla fine della fase di compilazione che si divide in trasformazione da circuito a QAP e successiva fase di trustedSetup che ovviamente in questo caso è stat svolta con un unico partecipantem, otteniamo i seguenti file:

- **rln_final.zkey**: file che contiene i parametri crittografici e i paramentri privati che consentono a un verificatore di controllare la validità di una prova senza rivelare alcuna informazione sui dati sottostanti.
- **rln.wasm**: è una versione compilata del circuito che può essere eseguita in un browser web utilizzando WebAssembly. Questo file è generato dal codice Circom e contiene la logica del circuito in un formato binario che può essere eseguito.
- **verification_key.json**: contiene una chiave pubblica relativa al circuito, che può essere usata da un verificatore per controllare la validità di una prova.

2.7 Server

La struttura del Server è semplice: si tratta di un progetto Node.js che utilizza la libreria Socket.IO per gestire la comunicazione con i Client. Il Server è composto da due classi principali: "server.ts" e "type.ts". La prima classe contiene la logica applicativa e implementa le primitive della libreria RLNjs per verificare le prove dei Client. La seconda classe è un file che definisce degli enum, utilizzati per identificare lo stato e gli eventi di comunicazione tra il Server e i Client. Questo file, insieme ai file di configurazione del circuito, è presente sia nel progetto Server che in quello Client.

Il Server attende la comunicazione con un Client sulla porta 3000 e rimane in attesa fino a quando non riceve un evento. Gli eventi possibili sono definiti nel file "type.ts" e quelli accettati dal Server sono EventType.REGISTER e EventType.INTERACTION. Questi eventi sono utilizzati rispettivamente per gestire la registrazione e l'interazione con gli utenti. Durante la fase di registrazione, il Server inserisce gli utenti nell'albero di Merkle, chiamato "registry" nel programma, e notifica lo stato della registrazione, che può essere uno dei seguenti: ALREADY_REGISTERED, BANNED o VALID.

```
socket.on(EventType.INTERACTION, async (data, callback) => {
...

if (registry.root !== BigInt(proof.publicSignals.merkleRoot)) {
  ...
}

const proofResult = await RLN.verifyProof(vKey, proof);
if (!proofResult) {
  ...
}

// raccolta di meta-dati per iedntificare velocemente
// chi supera il numero di richieste, utilizza il concetto
// di internal_nullifier
const res = cache.addProof(proof);
if (res.status === 'breach') {
  ...
}
// stesso messaggio invaito nella stessa epoca, non è considerato
// spam ma non viene trattato neanche come una richiesta
if (res.status === 'invalid') {
  ...
}

callback({
  header: 'ok',
  body: SignalVerificationStatus.VALID,
});
...
```

Figura 2.10: codice Server gestione registrazione

Mentre per quanto riguarda la fase di interazione, il Server si occupa prima di tutto di controllare che l'utente abbia inviato una prova valida. Questo controllo avviene verificando che la radice del suo registry collimi con quella del Server e che la generazione della prova abbia seguito i vincoli specificati dal circuito. Successivamente, il Server controlla se il messaggio inviato rispetta le regole anti-spam. Se ciò non avviene, il Server rimuove l'utente e sincronizza i registry dei suoi Client inviando

un messaggio di broadcast a tutti i membri. Nelle fasi successive, approfondiremo i tempi necessari per lo svolgimento di queste funzioni.

```
socket.on(EventType.REGISTER, (data, callback) => {
const identityCommitment = BigInt(data);
...
try {
    registry.addMember(identityCommitment);
} catch (error) {
    onError(callback, UserRegistrationStatus.BANNED);
    return;
}
socket.broadcast.emit(
  EventType.USER_REGISTERED,
  serializeLeaves(registry.members)
);

callback({
    header: 'ok',
    body: {
        status: UserRegistrationStatus.VALID,
        leaves: serializeLeaves(registry.members),
    },
});

...
```

Figura 2.11: codice Server gestione interazione

2.8 Client

Il Client presenta una struttura molto simile a quella del server, utilizza la libreria Socket.IO-client per gestire la comunicazione. Inoltre, dispone di un file "type.ts" analogo a quello del server per gestire gli eventi di comunicazione e conserva all'interno del progetto i file relativi alla generazione del circuito. Tuttavia, a differenza del server, il Client utilizza tali file per generare le prove e non per verificarle. Gli eventi rilevanti per il Client sono EventType.REGISTER, EventType.INTERACTION, EventType.USER_REGISTERED e EventType.USER_SLASHED. Gli ultimi due sono generati dal Server e servono a sincronizzare gli alberi di merkle degli utenti a seguito della registrazione o rimozione di un nuovo membro, infatti per poter generare prove valide, gli utenti devono possedere la stessa versione della struttura posseduta dal Server altrimenti il primo controllo sulla radice dei registri, non andrebbe a buon fine.

```
const refreshRegistry = (data) => {
  if (state.isRegistered) {
    for (let i = 0; i < data.length; i++) {
        leaves[i] = BigInt(data[i]);
    }

    merkleProof = Registry.generateMerkleProof(
        treeDepth, // (20)
        zeroValue, // BigInt(0)
        leaves, // deserializzate da data
        identityCommitment
    );
  }
};

socket.on(EventType.USER_REGISTERED, (data) => {
    console.log('New user registered!');
    refreshRegistry(data);
});

socket.on(EventType.USER_SLASHED, (data) => {
    console.log('User was slashed!');
    refreshRegistry(data);
});
```

Figura 2.12: codice Client sincronizzazione registri

Per ottimizzare la fase di interazione la generazione della prova di appartenenza al registro viene effettuata ad ogni sincronizzazione. La fase di registrazione è abbastanza banale in quanto è costituita solamente dall'invio dell'identity commitment al server a dall'attesa del relativo aknowledgement. Mentre la fase di interazione è più articolata in quanto è durante questa fase che il Client genera la prova di appartenere all'albero e di avere generato le porzioni di chiave da rilascaire in modo corretto. In questa fase, per testare le funzionalità di rate limiting, ho incluso la possibilità per l'utente di selezionare l'opzione "dos". Grazie a questa opzione, l'utente potrà provare ad inviare 100 messaggi nello stesso intervallo di tempo. Prima d'interagire con il sistema l'utente genera una prova utilizzando il segnale, che una volta applicata la funzione hash rappresenterà la nostra cordinata x, la merkleProof e l'externalNullifier. Le altre infomazioni necessarie come la chiave privata o l'rln_identifier vengono ricavate da un istanza della libreria RLNjs creata durante le fasi iniziali del Client.

```
rlnInstance = new RLN(
    wasmFilePath, // percorso file wasm
    finalZkeyPath, // percorso file paramentri privati del circuito
    vKey, // stringa ottenuta dal parsing del file verification_key.json
    rln_identifier,
    secret
);

...

const options = async () => {
...
  const n = signal === 'dos' ? 100 : 1; // signal viene inserito dall'utente
  const epoch = BigInt(Math.floor(Date.now() / 10000));
  const externalNullifier = genExternalNullifier(
      signal === 'dos' ? 'same' : epoch.toString()
  );

  for (let i = 0; i < n; i++) {
      const signalVerificationStatus = await interact(
          'This is a DoS attack'+i,
          externalNullifier
      );
  }
};

const interact = async (signal,externalNullifier
): Promise<SignalVerificationStatus> => {
  const proof = await rlnInstance.generateProof(
      signal,
      merkleProof,
      externalNullifier
  );

  const res: SignalVerificationStatus =
    await new Promise((resolve, reject) => {
      socket.emit(EventType.INTERACTION, proof, async (response) => {
          const status = response.header;
          if (status === 'error') {
              if (response.body === SignalVerificationStatus.BREACH) {
                  reject('User was slashed!');
              }
              reject(response.body);
          } else {
              resolve(response.body);
          }
      });
  });

  return res;
};
```

Figura 2.13: codice Client interazione con il sistema

2.9 Prestazioni

Di seguito riporto alcuni dati significativi ottenuti dall'esecuzione del prototipo. I test sono stati effettuati su un computer con le seguenti caratteristiche:

- CPU: Apple silicon M1
- RAM: 8GB
- OS: MacOS Ventura 13.0

Inizieremo analizzando il numero di vincoli che costituiscono i circuiti per la generazione delle prove e le loro dimensioni. È importante ricordare che i circuiti sono stati generati utilizzando la libreria Circom 2. Per i test che verranno presentati, sono state utilizzate tre diverse profondità degli alberi di Merkle: 16, 24 e 32, che corrispondono rispettivamente ad un massimo di 2^{16}, 2^{24} e 2^{32} utenti.

Funzione di Hash	Profondità	Numero vincoli	Parametri privati generati	Collegamenti tra gate	Alias	Dimensione file zkey	Dimensione file wasm
Poseidon (3, 8, 56)	16	4795	33	4816	15275	2.82 mb	1.94 mb
Poseidon (3, 8, 56)	24	6739	49	6768	21555	3.76 mb	1.95 mb
Poseidon (3, 8, 56)	32	8683	65	8720	27835	5.21 mb	1.93 mb

Figura 2.14: codice Client interazione con il sistema

I dati più significativi sono il numero di vincoli, le dimensioni dei file rln_final.zkey e rln.wasm. Infatti, se a un primo impatto i numeri dei vincoli generati dai circuiti possono sembrare molto elevati, in realtà le dimensioni dei file sono notevolemente ridotte, se pensiamo che un albero di Merkle di profondità 30 utilizzando una funzione di hash con output 128 bit, può arrivare pesare decine di GB. Di seguito invece mostriamo i tempi di esecuzione:

Profondità	Merkle Proof	Generazione prova	Verifica radice dell'albero	Verifica della prova
16	0.007 s	0.789 s	0 s	0.237 s
24	0.01 s	0.811 s	0 s	0.236 s
32	0.015 s	1.031 s	0 s	0.235 s

Figura 2.15: codice Client interazione con il sistema

qui i dati più significativi sono i tempi di generazione e verifica delle prove, in particolare la verifica che rimane praticamente costante. La colonna con indice "Verifica radice dell'albero" rappresenta la velocità con cui il sistema riesce a verificare se la radice dell'albero con cui è stata generata la prova è uguale a quello correto.

Ringraziamenti

Ringrazio il mio relatore per il supporto e per avermi dato l'opportunità di approfondire questa tema.

Altrettanto ringrazio la mia famiglia e tutti coloro che mi sono stati vicini in questo percorso e hanno creduto in me. A loro è dedicata questa tesi.

Bibliografia

[1] Cloud Armor Emil Kiner Senior Product Manager. *Google blocked largest Layer 7 DDoS attack*. URL: https://cloud.google.com/blog/products/identity-security/how-google-cloud-blocked-largest-layer-7-ddos-attack-at-46-million-rps.

[2] Eli Ben-Sasson et al. «Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture». In: *Proceedings of the 23rd USENIX Conference on Security Symposium*. SEC'14. San Diego, CA: USENIX Association, 2014, 781–796. ISBN: 9781931971157.

[3] Vitalik Buterin. «An approximate introduction to how zk-SNARKs are possible». In: (). URL: https://vitalik.ca/general/2021/01/26/snarks.html.

[4] S Goldwasser, S Micali e C Rackoff. «The Knowledge Complexity of Interactive Proof-Systems». In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. STOC '85. Providence, Rhode Island, USA: Association for Computing Machinery, 1985, 291–304. ISBN: 0897911512. DOI: 10.1145/22145.22178. URL: https://doi.org/10.1145/22145.22178.

[5] *Interactive proof system*. URL: https://en.wikipedia.org/wiki/Interactive_proof_system.

[6] Jean-Jacques Quisquater et al. «How to Explain Zero-Knowledge Protocols to Your Children». In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. A cura di Gilles Brassard. New York, NY: Springer New York, 1990, pp. 628–631. ISBN: 978-0-387-34805-6.

[7] dtubbenhauer. «Slide on Schwartz-Zippel Lemma». In: (). URL: https://www.dtubbenhauer.com/slides/my-favorite-theorems/23-schwartz-zippel.pdf.

[8] Ariel Gabizon. «Explaining SNARKs». In: (). URL: https://electriccoin.co/blog/snark-explain/.

[9] Vitalik Buterin. «How do trusted setups work?» In: (). URL: https://vitalik.ca/general/2022/03/14/trustedsetup.html.

[10] Vitalik Buterin. «trusted models». In: (). URL: https://vitalik.ca/general/2020/08/20/trust.html.

[11] Jens Groth. «On the Size of Pairing-Based Non-interactive Arguments». In: *Advances in Cryptology – EUROCRYPT 2016*. A cura di Marc Fischlin e Jean-Sébastien Coron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 305–326. ISBN: 978-3-662-49896-5.

[12] *Interactive proof system*. URL: https://en.wikipedia.org/wiki/Non-interactive_zero-knowledge_proof.

[13] barryWhiteHat. *Semaphore RLN, rate limiting nullifier for spam prevention in anonymous p2p setting*. URL: https://ethresear.ch/t/semaphore-rln-rate-limiting-nullifier-for-spam-prevention-in-anonymous-p2p-setting/5009.

[14] *Privacy and Scaling Exploration*. URL: https://appliedzkp.org/.

[15] 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.

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

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