Contents

1 Zk Stark

Zero-knowledge Succinct non-interactive ARguments of Knowledge (zk SNARK) è un ingegnoso algoritmo per provare che una determinata affermazione è vera, senza rivelare dettagli relativi all'affermazione stessa. Tanto per fare alcuni esempi, diventa possibile:

1) Dimostrare alcune affermazioni su dati personali del tipo:
- una determinata persona ha più di X euro nel suo conto bancario (senza rivelare il valore di X)
- il DNA di una determinate persona combacia (senza rivelarne i dettagli)

2) Autorizzazioni anonime:
- dimostrare che un richiedente R ha i diritti di accesso ad un'area riservata di un sito web senza rivelare username e password (questo probabilmente non è cosi diverso dalle catene di Single Sign-On, in cui comunque è presente una catena fiduciaria che qui invece non esiste)
- dimostrare la provenienza da una lista ben determinata di Paesi, senza rivelare il paese specifico

3) Pagamenti anonimi, assicurazioni:
- effettuare pagamenti senza rivelare la propria identità
- pagare le tasse senza rivelare nello specifico le proprie entrate
- stipulare una assicurazione senza rivelare i dettagli specifici, ma semplicemente il fatto che si ha una età compresa tra 30 e 40 anni, e uno stipendio tra 30 e 40 mila euro.

4) delegare esternamente i calcoli di funzionamento delle blockchain:

- eseguire esternamente un calcolo oneroso e validare che risulta corretto senza rifarlo in toto, cosa che apre ad una categoria di servizi di 'trustless' computing
- cambiare il modello di una blockchain da "tutti calcolano e concordano" a "tutti verificano i calcoli fatti da altri".

Nel caso 4 si parla di "Zk Rollup": è noto che Ethereum sia in sofferenza e che il suo 'gas' costi troppo (non solo il suo di questi tempi, purtroppo), per tale ragione risulta più economico rivolgersi ad altri competitor, che però potrebbero sfruttare comunque Ethereum per validare le prove della correttezza dei calcoli computazionali gestiti offline da 'parachain' o 'side-chain' parallele, come fa per esempio Polygon (che già utilizza questi "zk rollup"). Ethereum infatti rimane ad oggi la blockchain più decentralizzata ed anche la più sicura. La sua lentezza è legata principalmente al privilegiare questi due vertici del famoso triangolo, solitamente chi millanta prestazione elevatissime lo fa centralizzando e richiedendo requisiti da supercomputer e tanta banda, e non da laptop che chiunque potrebbe possedere. Poichè Ethereum passerà tra non molto da PoW a PoS, cosa che era nota già dall'inizio del suo sviluppo nel 2015, anche se ci sono voluti parecchi anni, i miners che in questo periodo di caro prezzi dell'energia vedono già intaccati i propri guadagni, potrebbero riciclare il proprio hardware dedicato per altre criptovalute (cosa che non risolve il caro prezzi energetico) oppure occuparsi in qualche modo delle 'parachains' e dei calcoli necessari per creare le prove di correttezza computazionale.

Come quasi sempre in questi casi, sotto algoritmi di questo tipo spiegati ad alto livello, ci sono elaborati e complessi algoritmi basati su matematica, gruppi aritmetici, crittografia, che mettono insieme (come nel caso di Bitcoin) 20 o 30 anni di ricerca pura sul campo. Tanto per fare un esempio il primo lavoro sulle "prove di corretta computazione" risale alla fine degli anni '80 al MIT e tra le varie menti dietro la pubblicazione c'è anche quella dell'italiano Silvio Micali.
In ogni sistema di "prova a conoscenza zero", esiste un dimostrante (o 'prover' in inglese) ed un controllore (o 'verifier'). Il dimostrante non vuole rivelare i dettagli di ciò che vuole dimostrare, ci sono quindi essenzialmente 3 caratteristiche per un protocollo di questo tipo:

- Completeness — se l'affermazione del dimostrante è vera ed il dimostrante è onesto, lo stesso può convincere il controllore della veridicità della sua affermazione
- Soundness — un dimostrante disonesto NON potrà convincere il controllore che sta asserendo qualcosa di falso
- Zero-knowledge — l'interazione tra dimostrante e controllore è tale per cui non vengono rivelati i dettagli più privati dell'affermazione stessa

1.1 Documentazione

La documentazione sull'argomento, a parte i White Paper illeggibili e incomprensibili ai più, è riportata di seguito. E' comunque complessa e richiede spesso basi non banali di matematica, arrivando ad essere materia veramente per pochi specialisti volendo essere formalmente corretti in tutte le affermazioni che si fanno. In generale, dopo aver letto tutto il materiale sottostante varie volte ed aver visto i video su Youtube, ho deciso di tradurre in Italiano i 3 post di Vitalik Buterin, aggiungendo ove possibile dettagli per chiarire ed ampliare passaggi potenzialmente complessi che sono stati magari riassunti in poche parole o poche righe. In caso di dubbi sulla paternità delle affermazioni, fare sempre riferimento ai post originali in inglese.

https://vitalik.ca/general/2017/11/09/starks_part_1.html

https://vitalik.ca/general/2017/11/22/starks_part_2.html

https://vitalik.ca/general/2018/07/21/starks_part_3.html

Un altro sito importante con svariati post interessanti è il seguente (ho però riportato solamente un paio di link relativi a Stark):

https://kitten-finance.medium.com/low-degree-testing-in-zk-stark-part-1-c0ac6ef0de3c

https://kitten-finance.medium.com/low-degree-testing-in-zk-stark-part-2-7c729118d10c

Il sito seguente è straordinariamente completo e ricco di contenuti su tutto il mondo delle blockchain ed i loro costrutti matematici e crittografici, e vanno estremamente nel dettaglio delle cose:

https://zeroknowledgedefi.com/

https://zeroknowledgedefi.com/2021/12/13/starks-i-polynomials-everywhere/

https://zeroknowledgedefi.com/2021/12/26/starks-ii-is-that-a-polynomial/

sempre dello stesso autore anche il seguente (meno focalizzato sul ramo crypto):

https://almostsuremath.com/

Anche questo è un link riassuntivo sull'argomento, non particolarmente dettagliato ma che messo insieme a tutti gli altri può riempire alcune piccole lacune:

https://medium.com/unitychain/reveal-mysterious-zk-starks-42d00679c05b

Starkware è l'azienda che fornisce soluzioni di questo tipo per applicazioni legate alle blockchain, ma probabilmente non solo e non necessariamente in modo esclusivo.

https://starkware.co/stark/

https://medium.com/starkware/stark-math-the-journey-begins-51bd2b063c71

https://medium.com/starkware/arithmetization-i-15c046390862

https://medium.com/starkware/arithmetization-ii-403c3b3f4355

https://medium.com/starkware/low-degree-testing-f7614f5172db

https://medium.com/starkware/a-framework-for-efficient-starks-19608ba06fbe

https://www.youtube.com/watch?v=Y0uJz9VL3Fo (Parte1)

https://www.youtube.com/watch?v=fg3mFPXEYQY (Parte 2)

https://www.youtube.com/watch?v=gd1NbKUOJwA (Parte 3)

https://www.youtube.com/watch?v=CxP28qM4tAc (Parte 4)

https://www.youtube.com/watch?v=iuNbrTkH2ik (Parte 5)

Il codice Python di cui si parla in questi video, come riportato su Youtube stesso nei commenti, si trova qui:

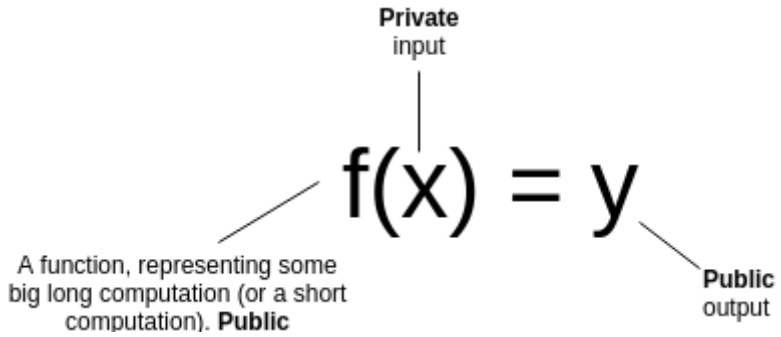
https://github.com/starkware-industries/stark101/

1.2 Zero Knowledge Stark parte 1

Un grazie speciale ad Eli Ben-Sasson per l'aiuto, le spiegazioni e la review, per alcuni degli esempi in questo post, e soprattutto per avere inventato la gran parte di tutto ciò. Grazie a Hsiao-wei Wang per la revisione.

Speranzosamente molti hanno già sentito parlare di ZK-Snark, l'algoritmo generale di prova a conoscenza zero che può essere utilizzato in molti casi reali. Ora ZK-SNARK ha un nuovo cugino: ZK STARK, con la T che sta per Trasparente, risolve uno dei problemi più spinosi di ZK-Snark, ossia la necessità di un "setup fidato", cui spesso si fa riferimento anche come 'cerimoniale' in cui dei partecipanti devono generare dei numeri casuali e poi distruggerli onde evitare di compromettere potenzialmente la sicurezza del sistema. Inoltre sfrutta algoritmi crittografici più semplici, non utilizza le curve ellittiche, le associazioni omomorfe (il 'pairing') e si basa invece puramente su hash e teoria dell'informazione, cosa che lo rende resistente anche agli attacchi di ipotetici computer quantici.

Come fa quindi a funzionare questo algoritmo di prova a conoscenza zero ? ricordiamo prima che cosa fa un algoritmo "a prova breve" ('succinct' in inglese) Zero Knowledge. Si supponga di avere una funzione pubblica f, un ingresso privato x ed un output pubblico y. Si vuole sostanzialmente provare che si conosce un x tale per cui f(x)=y, ma senza rivelare a che cosa corrisponda x. Inoltre, perchè si possa considerare la prova come 'breve', la verifica deve richiedere molto meno tempo di quanto non sia richiesto calcolarla.



Facciamo alcuni esempi di applicazioni pratiche:

- F è un calcolo che richiede due settimane in un normale laptop domestico, ma un paio d'ore in un data center in cloud. Si invia al data center la funziona calcolo, e ci si aspetta non solo un risultato ma anche la prova che lo stesso sia corretto, una prova semplice e verificabile con il laptop domestico in pochi millisecondi.
- Hai una transazione criptata nella forma "X1 è il mio vecchio saldo, X2 è il tuo vecchio saldo. X3 è il mio nuovo salvo, X4 è il tuo nuovo saldo". Vuoi creare una prova che questa sia una transazione valida, ossia i vecchi e nuovi saldi siano non negativi, e la decrescita del mio saldo corrisponda alla crescita del tuo (più eventuali commissioni). x può essere costituito da un paio di chiavi pubbliche, ed f può essere la funzione che ha come input la transazione, le chiavi, decripta la transazione, esegue i controlli di congruenza e restituisce 1 se è tutto ok, 0 altrimenti.
- hai una blockchain come Ethereum, ed esegui il download del blocco più recente. Vuoi una prova che il blocco sia valido, e che questo blocco sia l'ultimo di una catena in cui OGNI blocco è valido. Chiedi pertanto ad un 'full node' (ossia un nodo che possiede l'intera blockchain memorizzata localmente) di fornire tale prova, dato che non hai tempo di scaricare l'intera blockchain e verificarla, cosa che potrebbe richiedere parecchi giorni, banda e spazio nel tuo hard disk.

- cosa c'è di cosi difficile in tutto questo ? la componente *zero knowledge* ossia la privacy è relativamente semplice da ottenere, come si vedrà
- la cosa molto più difficile da fare è rendere breve la verifica della prova fornita. Infatti qualunque sequenza di calcoli è particolarmente fragile, nel senso che basta alterare un bit in un qualunque passaggio, perchè il risultato finale ne risulti totalmente compromesso.
- ad alto livello l'idea è quella di sfruttare algoritmi di data fault tolerance. Se si dispone di un dato e lo si codifica rappresentandolo come una linea retta, si possono scegliere 4 punti su di essa. Due qualunque di questi punti sono sufficienti per ricostruire la linea originale, e da questi si possono quindi calcolare tutti gli altri punti della retta stessa. Si possono codificare altresì i dati a nostra disposizione con un polinomio di 1 milione di coefficienti, e memorizzare 2 milioni di punti su tale polinomio. Qualunque sequenza di 1 milione ed 1 punto è sufficiente per ricostruire tutti gli altri punti (che potrebbero essere stati persi, modificati, corrotti e compromessi), cosi come una qualunque modifica nel milione di punti originali avrebbe prodotto invece 2 milioni di punti da memorizzare completamente diversi. Non è quindi un caso che tali protocolli abbiano qualcosa in comune con i codici a correzione d'errore (Reed Solomon per esempio), che hanno la funzione di amplificare qualunque tipo di errore o deviazione dai dati originali, rendendo possibile anche la correzione di eventuali errori, oltre che la 'semplice' rilevazione degli stessi.

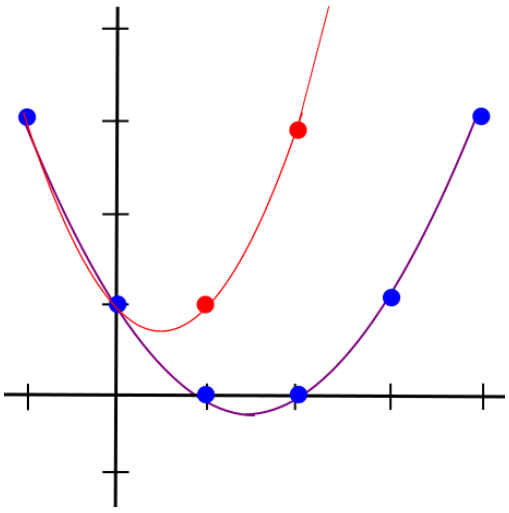


Figura 1 Punti che non rappresentano un polinomio.

1.2.1 Un semplice esempio

Si supponga che si voglia dimostrare la conoscenza di un polinomio P tale per cui P(x) sia un intero compreso tra 0 e 9 per qualunque x da 1 ad 1 milione. Questa è la traduzione polinomiale di un problema di controllo di range, ossia del fatto che per esempio una persona abbia una età compresa tra 40 e 50 anni (ma senza rivelare tale valore), oppure che tutti gli account su una blockchain siano positivi dopo aver applicato una serie di transazioni che coinvolgono gli account stessi. Nel caso in cui la relazione sia invece:

$$1 \leq P(x) \leq 9$$

il polinomio potrebbe rappresentare una soluzione del noto gioco Sudoku. Il metodo tradizionale con cui provare una tale affermazione, sarebbe quello di mostrare il 1,000,000 di punti assunti dalla funzione f(x) e verificare controllandone i valori. Ma a noi interessa evitare di dover eseguire un controllo estensivo di tutti i valori del polinomio nello spazio di lavoro considerato. Un controllo casuale non funzionerà perchè c'è sempre la possibilità che un dimostrante disonesto possa fornire un polinomio che soddisfa i requisiti in 999,999 punti ma non lo soddisfa nell'ultimo, ed un controllo casuale su un sottoinsieme di punti immancabilmente finirà per mancarlo. Che cosa possiamo fare quindi ?

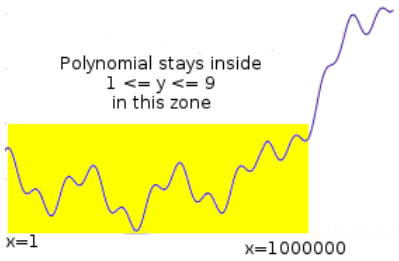


Figura 2 Polinomio di valore compreso tra 1 e 9 nel range voluto di valori di x.

Si tratta in ogni caso di trasformare matematicamente il problema. Definiamo C(x) come un polinomio che sia sempre 0 per x nel range di valori $[0, \dots, 9]$ e diverso da zero altrove. Per costruire un siffatto polinomio, è sufficiente procedere moltiplicando:

$$C(x) = x(x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(z-7)(x-8)(x-9)$$

Supponiamo di lavorare solo ed esclusivamente sui numeri interi, pertanto non dobbiamo preoccuparci dei valori del polinomio nei numeri reali compresi tra i vari interi. Nelle applicazioni pratiche, si lavora sempre in questo modo, ossia in spazi finiti di numeri interi.

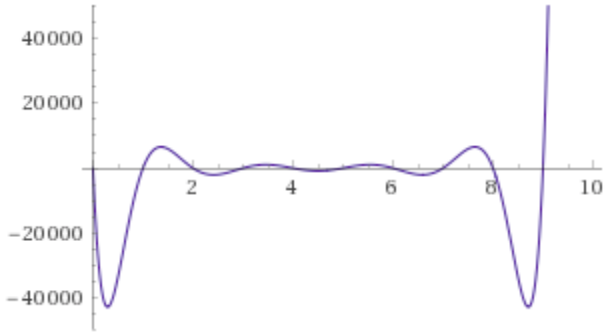


Figura 3 Polinomio a valore 0 per x intero tra 0 e 9 compresi.

Il problema precedente cosi formulato diventa il seguente: provare che si conosce un polinomio P tale per cui:

$$C\big(P(x)\big) = 0, \forall\, x \,\epsilon\, [1,2,\dots,1.000.000]$$

Definiamo quindi:

$$Z(x) = (x-1)(x-2)\dots(x-1.000.000)$$

E' piuttosto banale dimostrare che qualunque polinomio che valga 0 per ogni x compreso tra 1 e 1 milione, sia necessariamente un multiplo di $Z(x)$. Quindi il problema può essere nuovamente riformulato nel modo seguente: provare che si conoscono un polinomio $P(x)$ e $D(x)$ tali per cui $C\big(P(x)\big) = Z(x) \cdot D(x)$ per ogni x. Si noti che se si è a conoscenza di $P(x)$, trovare $D(x)$ eseguendo la divisione polinomiale di $P(x)$ per $Z(x)$ non è una operazione complessa. Ricordiamo che tale approccio (trasformare un problema computazionale in un polinomio che risolve tale problema nei punti in cui lo stesso vale zero) è generale per tutti gli algoritmi di 'Zero knowledge proof', non solo per quello che poi si declina nello ZK STARK.

Come si prova una tale affermazione ? Possiamo immaginare il protocollo di prova come un colloquio in 3 passi tra controllore e dimostrante: quest'ultimo invia delle informazioni, il controllore ne richiede altre, il dimostrante le invia. Come prima cosa, il dimostrante calcola il 'merkle tree root hash' di tutti i punti del polinomio tra 1 e 1 miliardo, con complessità algoritmica $O(N \cdot logN)$. 1 miliardo non è un errore, infatti è fondamentale che il numero di punti considerati sia MOLTO più grande del range o comunque del numero di punti sul quale abbiamo imposto dei vincoli. Quindi i punti saranno il milione di punti a valore a 0, e i 999 milioni di punti che (probabilmente) saranno diversi da zero. Per chi non sapesse che cosa sia un 'merkle tree', si rimanda ai seguenti post:

https://zeroknowledgedefi.com/2021/06/20/merkle-trees/

https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/

Un hash invece non è nient'altro che un modo per generare una sequenza di bit di lunghezza fissa a partire da una sequenza di dati di origine. Se anche un solo bit è diverso, l'output sarà completamente diverso. La probabilità di collisione dovrà essere necessariamente bassa, ciò fa parte del protocollo. Tale costrutto algoritmico si potrebbe per esempio usare per confrontare due file e capire se sono uguali, indipendentemente dal nome con cui sono stati salvati, ed una volta verificato preliminarmente che hanno la stessa lunghezza (cosa sfruttata anche da Emule per esempio).

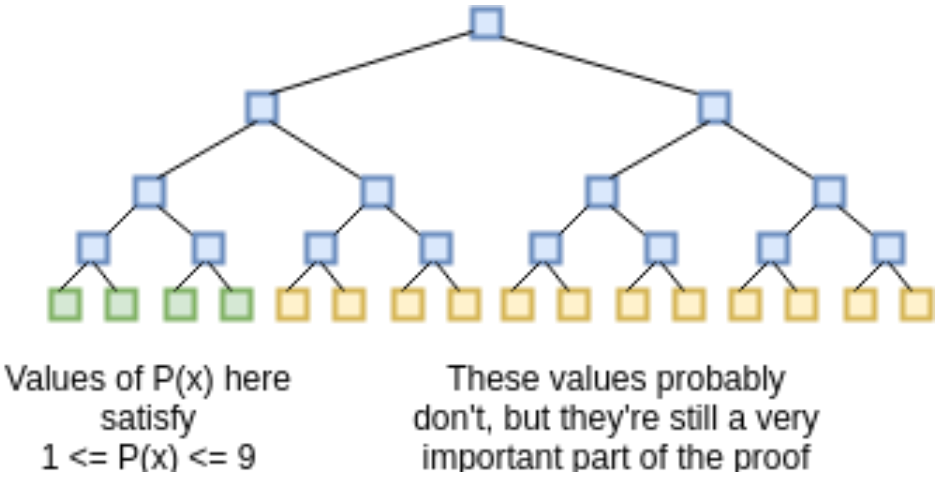


Figura 4 Merkle Tree, per ogni coppia di valori si calcola un hash e si prosegue fino ad avere un unico hash.

Si assume che al controllore sia noto il polinomio $Z(x)$ per tutti questi punti, in questo senso $Z(x)$ è come fosse una chiave pubblica di verifica per tale protocollo, e che tutti devono conoscere. Dopo aver ricevuto tale dato, il controllore sceglie 16 valori tra 1 e 1 miliardo e chiede al dimostrante di fornire i rami del 'Merkle tree' per i polinomi $P(x)$ e $D(x)$ in quei punti. Il controllore a questo punto verifica che:

1. $C\big(P(x)\big) = Z(x) \cdot D(x)$ su tutti e 16 i valori di x richiesti
2. I rami parziali del 'merkle tree' forniti sono corretti e portano, assieme ai punti di $P(x)$ calcolati al passo precedente, al medesimo root hash finale

Figura 5 Calcolo del root hash a partire da alcuni valori del polinomio.

Il punto 2 garantisce la privacy (sono stati rivelati solo alcuni punti del polinomio soluzione del nostro problema computazionale) e la correttezza dei valori del polinomio forniti preliminarmente dal dimostrante. Dalla seguente figura è più semplice capire come si ricava la radice da alcune foglie (per esempio dalla t_5):

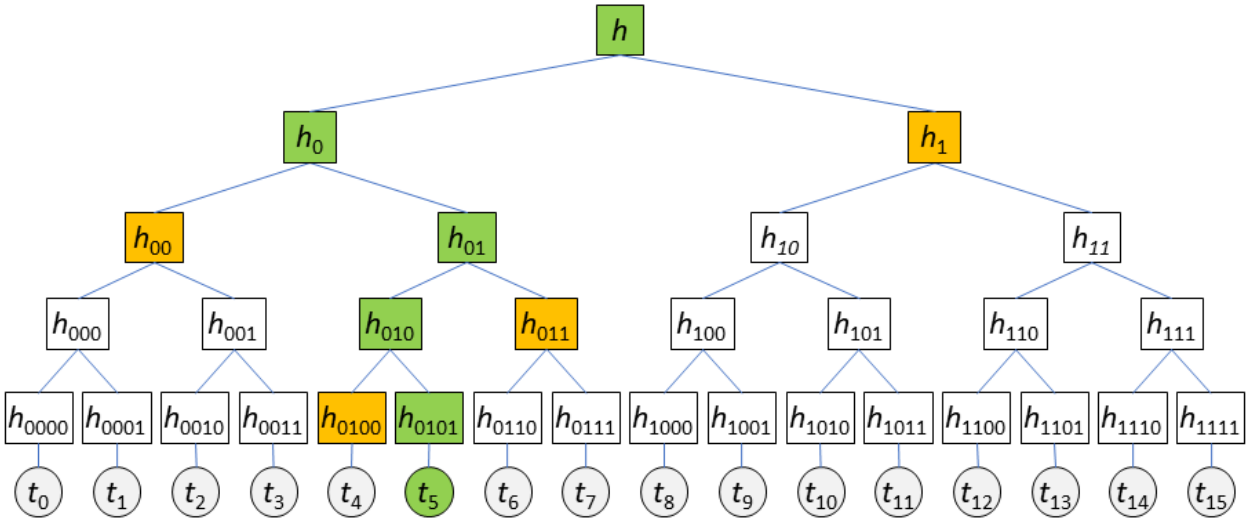


Figura 6 Calcolo del root hash a partire da alcuni valori del polinomio.

Il dimostrante fornisce i valori di $t_5, h_{0100}, h_{011}, h_{00}, h_1$, il controllore sfrutta tali valori per calcolare il root hash e verificare che coincide con quello fornito da dimostrante. La funzione di hash infatti non è (ovviamente) invertibile, ma soprattutto non è possibile 'condizionare' e pre-calcolare l'output, se non eseguendo una sorta di 'attacco in forza bruta' che poi è la base dei miners dei Bitcoin e del loro algoritmo di Proof of Work (modificare un 'nonce' parametrico affinchè l'hash sia inferiore ad un certo numero, o parimenti cominci con un certo numero di zeri).

Sappiamo che questa rappresenta una prova di 'completeness', ossia un dimostrante onesto sarà in grado di eseguire gli step come descritti, e di passare il test del controllore. Ma cosa possiamo invece affermare riguarda la 'soundness' ? che cosa è in grado di fare un dimostrante disonesto ? come possiamo essere sicuri che non sia in grado di produrre una falsa prova in grado di convincere il controllore ? Poichè $C(P(x))$ è un polinomio di grado 10, ossia il grado di $Z(x)$, composto con un polinomio di grado 1.000.000 il suo grado composto sarà di almeno 10 milioni. Lo stesso sarà quindi diverso da $Z(x) \cdot D(x)$ in almeno 990.000.000 di punti e quindi la probabilità che un $P(x)$ falso sia rilevato dal controllore al primo step è già del 99%. Con 16 controlli la probabilità di essere 'beccato' sale indicativamente ad $1 - 10^{-32}$ (ossia 1/100 elevato alla 16). In sostanza la probabilità di essere colti in flagranza di reato è la stessa di avere una collisione tra due hash partendo da sorgenti dati diverse, qualcosa di trascurabile.

Ciò che si è fatto, è stato sostanzialmente usare dei polinomi per amplificare l'errore in una soluzione non corretta, in modo tale da poter verificare nel 99% dei casi con un solo passaggio se un dimostrante è disonesto o meno. L'unica cosa che un dimostrante non onesto può cercare di fare senza possedere un valido $P(x)$, è essere estremamente fortunato con i rami del merkle root selezionati, ma la probabilità che ciò avvenga è talmente bassa che ci vorrebbero miliardi di anni perchè possa accadere.

Il tutto potrebbe essere reso non interattivo, se la scelta dei punti diventasse dipendente in modo deterministico dal valore del root hash: ad esempio il primo byte del route hash seleziona un valore di x ove fornire quelli dei polinomi P(x) e D(x). Poichè non si può condizionare o modificare il valore di un root hash, il dimostrante non può modificare arbitrariamente il valore del polinomio, perchè anche la modifica del polinomio in un solo punto, avrebbe come conseguenza la modifica del root hash calcolato.

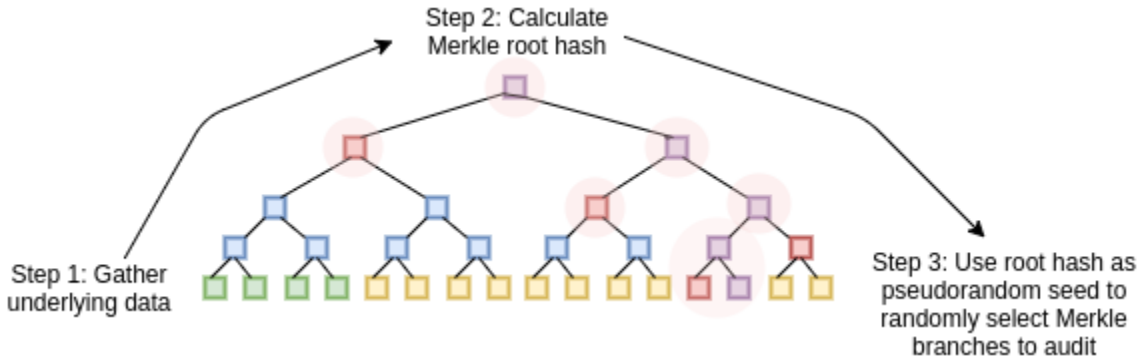


Figura 7 Calcolo del root hash a partire da alcuni valori del polinomio.

1.2.2 Proseguendo ulteriormente

Per illustrare la potenza di questa tecnica, utilizziamola per fare qualcosa di meno semplice: vogliamo dimostrare che conosciamo il milionesimo numero della sequenza di Fibonacci, chiaramente senza rivelarlo nello specifico. Per assolvere a questo compito, dobbiamo dimostrare la conoscenza di un polinomio che rappresenta una 'sequenza computazionale', con P(x) che rappresenta lo x-esimo valore della sequenza di Fibonacci. Per chi non lo sapesse, la sequenza è:

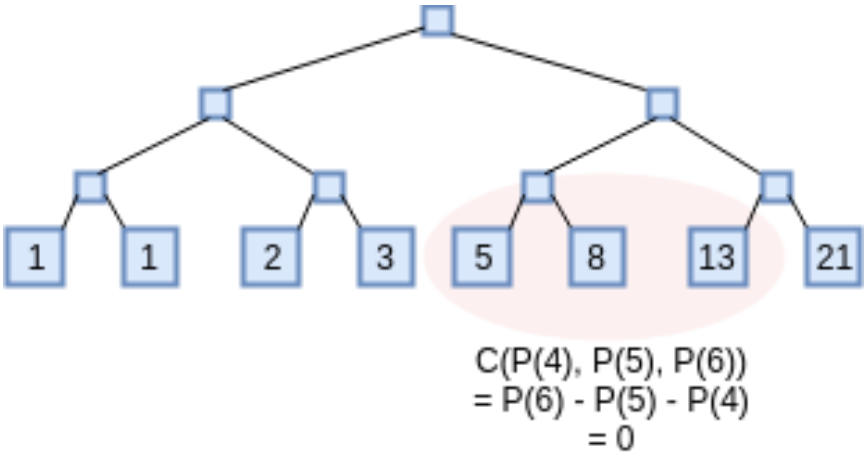
1,1,2,3,5,8,13,21,34,...

ossia ogni valore n-esimo è il risultato della somma dei due precedenti. Chiaramente i primi due step vanno 'inizializzati' non esistendo alcun valore precedente. Il polinomio di check dei vincoli coinvolgerà a questo punto 3 coordinate di x e quindi:

$$(x_1, x_2, x_3) = x_3 - x_2 - x_1$$

$$C\big(P(x), P(x+1), P(x+2)\big) = 0, \ \forall\, x\epsilon[1,2,\ldots,1.000.000]$$

gli altri valori possono essere qualunque fino a (per esempio) 10^9, infatti ricordiamo che il campo di lavoro deve essere molto più grande (in questo esempio è 1000 volte più grande) di quello della sequenza di calcolo.



Il problema traslato diventa il seguente: si provi la conoscenza di un polinomio $P(x)$ e $D(x)$ tale per cui:

$$C\big(P(x), P(x+1), P(x+2)\big) = Z(x) \cdot D(x)$$

con:

$$Z(x) = (x-1)(x-2)(x-2) \ldots . (x - 1.000.000)$$

Il polinomio $Z(x)$ è un parametro pubblico ed i suoi coefficienti possono essere calcolati da chiunque. Per ciascuno dei 16 indici indicati nella prova, il dimostrante dovrà fornire gli hash dei rami opportuni per $P(x), P(x+1), P(x+2), D(x)$ inoltre dovrà fornire i rami del Merkle tree per dimostrare che P(0)=P(1)=1. Per il resto, il processo precedentemente descritto rimane lo stesso.

Per ottenere quanto descritto nella realtà, ci sono 2 problemi che dobbiamo risolvere. Il primo problema è che lavorare con i numeri naturali non è possibile nè efficiente in pratica, perchè gli stessi diventerebbero troppo grandi. Il milionesimo numero della sequenza di Fibonacci per esempio ha 208988 cifre. Se vogliamo raggiungere la brevità della prova, dobbiamo usare i campi finiti (che descriveremo in seguito), e lavorare su questo nuovo insieme.

Il più semplice esempio possibile di campo finito è l'aritmetica modulare (descritto in 1.3.2)

Secondo, potreste aver notato che nella prova di 'veridicità' ci siamo dimenticati di coprire un attacco specifico: cosa accade se invece di utilizzare un polinomio di grado 1.000.000 per P(x) e 9.000.000 per D(x), il dimostrante disonesto utilizzasse valori che non si trovano su nessun polinomio ? In questo caso l'argomentazione che un polinomio sbagliato C(P(x)) debba differire da uno valido in almeno 990 milioni di punti non si applica, per cui sono possibili molti diversi tipi di attacchi. Per esempio, un disonesto potrebbe generare un valore a caso P(x) per ogni x, e calcolare di conseguenza:

$$D(x) = \frac{C(x)}{Z(x)}$$

Successivamente può inviare questi valori al posto di P(x) e D(x), valori che non si trovano su alcun polinomio di basso grado, ma passerebbero il test del controllore.

E' possibile effettivamente difendersi da tale possibilitò, sebbene i tool per farlo sono piuttosto complessi, e si può dire costituiscano la base di innovazione matematica che sta alla base dell'algoritmo Stark. Inoltre la soluzione ha una limitazione: si possono rilevare prove che siano molto lontane da un polinomio di grado 1.000.000, ma non che prove che differiscano da questo in solo 1 o 2 punti. Quindi ciò che con tale algoritmo

si dimostra è una cosiddetta “prova di prossimità”, ossia una prova che la gran parte dei punti di P e D corrispondano effettivamente ad un polinomio corretto di basso grado.

Si può dimostrare che ciò è sufficiente a generare una prova, sebbene ci siano 2 casi particolari. Primo, il controllore ha bisogno verificare qualche valore in più dello stretto indispensabile per ridurre la probabilità di errore dell’algoritmo. Secondo, se si necessita anche l’esecuzione di vincoli specifici di range o valori precisi (per esempio verificare che i primi due valori della sequenza di Fibonacci siano 1), è necessario estendere la prova di prossimità in modo da non provare solamente che la gran parte dei punti stanno sullo stesso polinomio, ma che quei due specifici punti (o N generici) stanno sul polinomio.

1.3 Zero Knowledge Stark parte 2

Grazie ad Eli Ben-Sasson per il continuo aiuto e le spiegazioni, ed a Justin Drake per la review del post.

Nella prima parte abbiamo spiegato come sia possibile fornire alcune interessanti prove brevi di corretta computazione, come per esempio dimostrare che si è calcolato ed è noto il milionesimo numero della sequenza di Fibonacci, utilizzando dei polinomi che hanno degli zeri in corrispondenza della soluzione che si dovrebbe fornire. Rimane comunque da dimostrare una cosa fondamentale, ossia che la gran parte dei punti presi all’interno di un insieme molto più ampio si trova sullo stesso polinomio “di basso grado”. Il concetto di basso grado è relativo al fatto che il grado del polinomio che risolve il determinato problema è noto, ma potrebbero esserci infiniti polinomi di grado maggiore con gli stessi zeri in corrispondenza agli stessi valori di x, dato che ricordiamo:

$t(x) = Z(x) \cdot H(x)$

Il suddetto problema è chiamato “low degree testing” ed è la parte più complessa del protocollo, dato che nulla viene gratis e questa è anche la parte che permette di raggiungere complessità computazioni migliori rispetto a “ZK Snark”.

Inizieremo dal ribadire il problema che deve essere risolto. Si supponga di avere una serie di punti che si trovano tutti nello stesso polinomio con grado minore di ‘D’ (per esempio se D=2 significa che tali punti stanno tutti su una retta del tipo $y = ax + b$, se D=3 i punti stanno tutti su una parabola del tipo $y = ax^2 + bx + c$). Si vuole creare una ‘breve’ prova probabilistica che tale affermazione sia vera. Breve in questo caso è la traduzione di ‘succinct’, va inteso come il fatto che se il polinomio è per esempio nello spazio $Z_p$dei numeri interi [0,1,2,...,p-1] tale prova non può tradursi nel fornire il valore del polinomio in tutti i punti, in quanto questo non è fattibile nelle applicazioni pratiche, oltre ad essere contrario al principio della “conoscenza zero”.

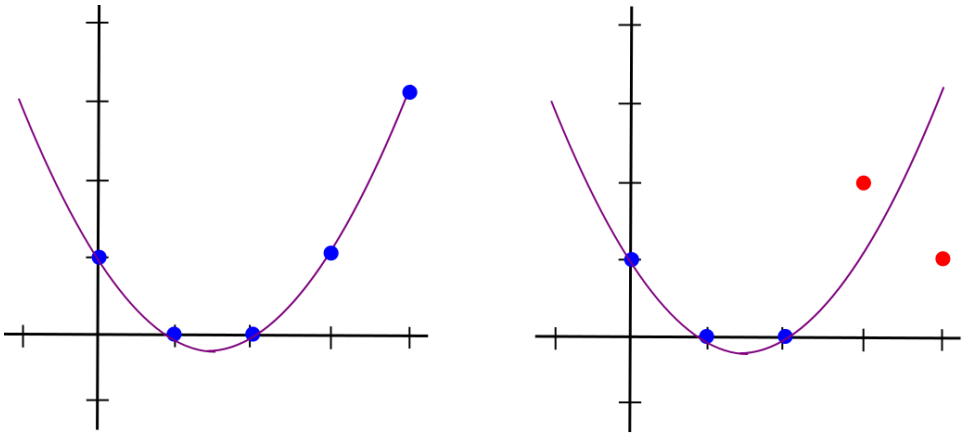


Figura 8 A sinistra: punti sullo stesso polinomio di grado minore di 3 (una parabola). A destra: punti che non si trovano tutti sullo stesso polinomio di grado minore di 3.

Se si vuole verificare che gli N punti forniti stiano tutti sullo stesso polinomio di grado minore di D, è necessario verificarli senza eccezioni, dato che saltandone uno non si avrà mai la certezza dell'affermazione. Ma ciò che si può alternativamente pensare di fare, è verificare che almeno una frazione di questi (per esempio il 90%) si trovino sullo stesso polinomio.

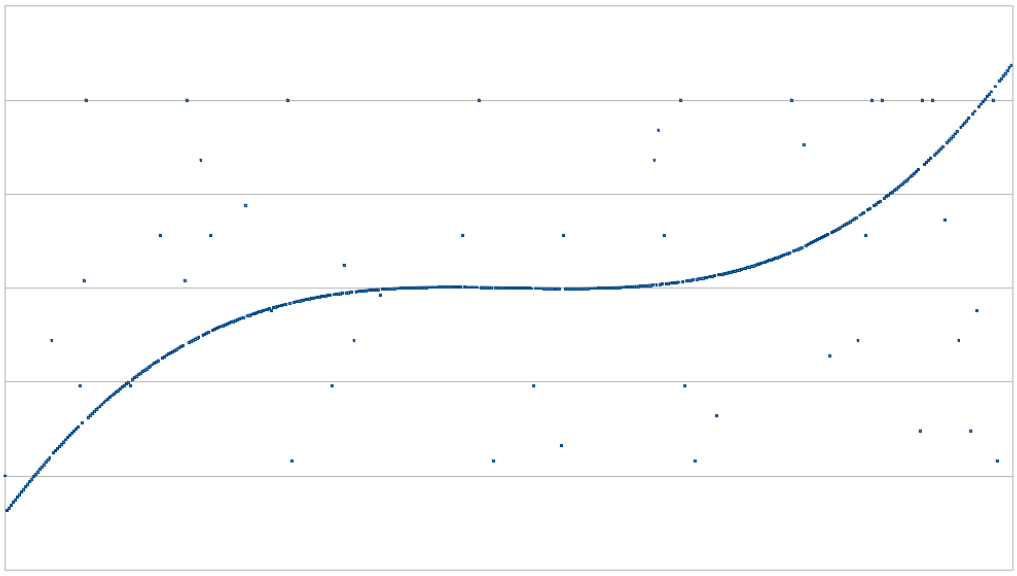


Figura 9 Punti che con buona probabilità rappresentano un polinomio.

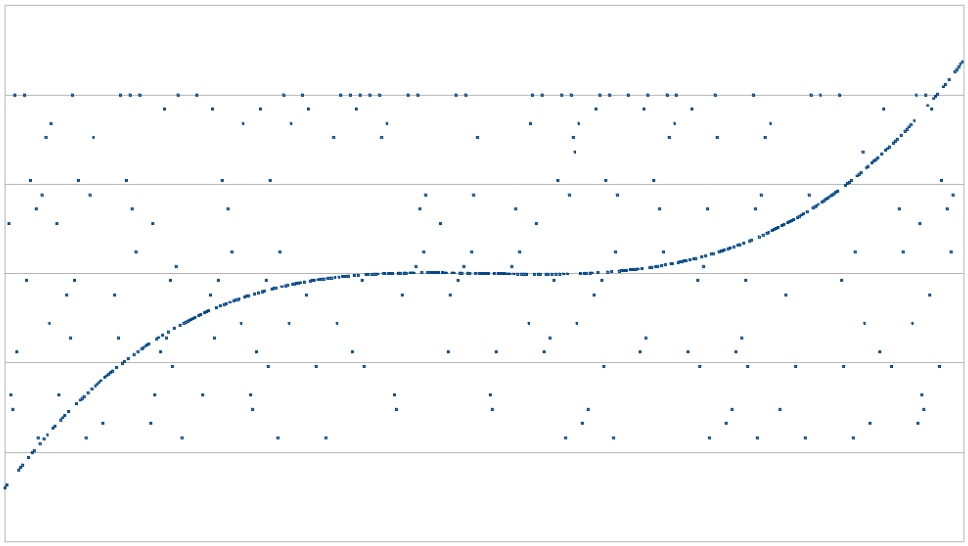


Figura 10 Punti che non rappresentano un polinomio con sufficiente probabilità

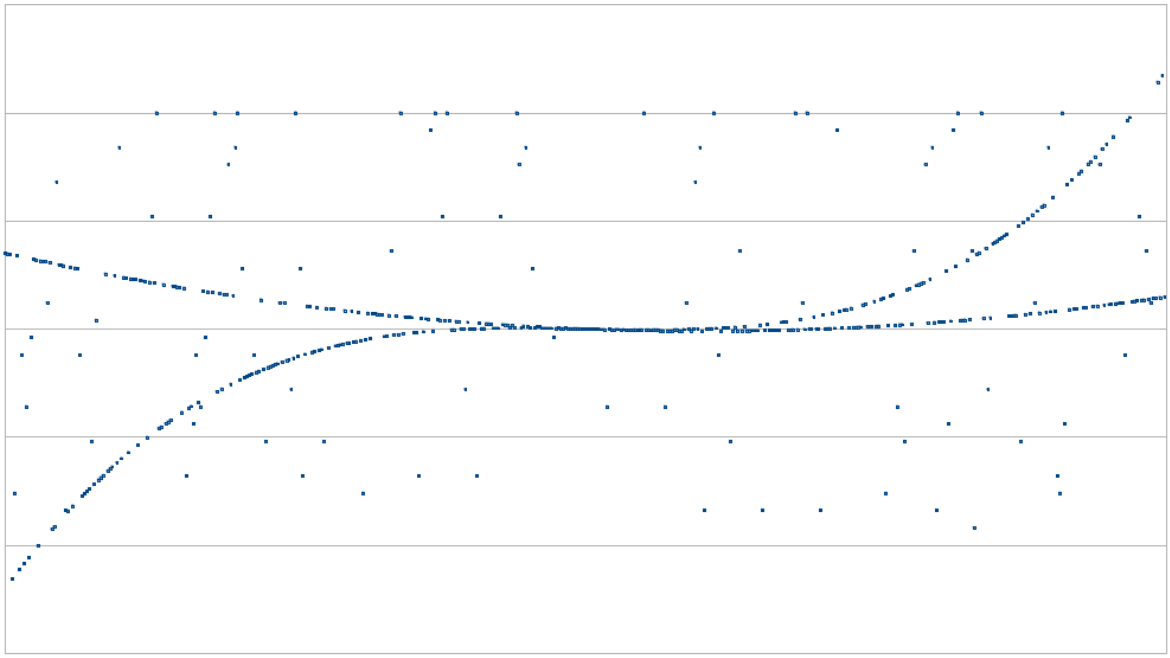


Figura 11 Punti che non rappresentano un polinomio singolo, ma probabilmente due diversi.

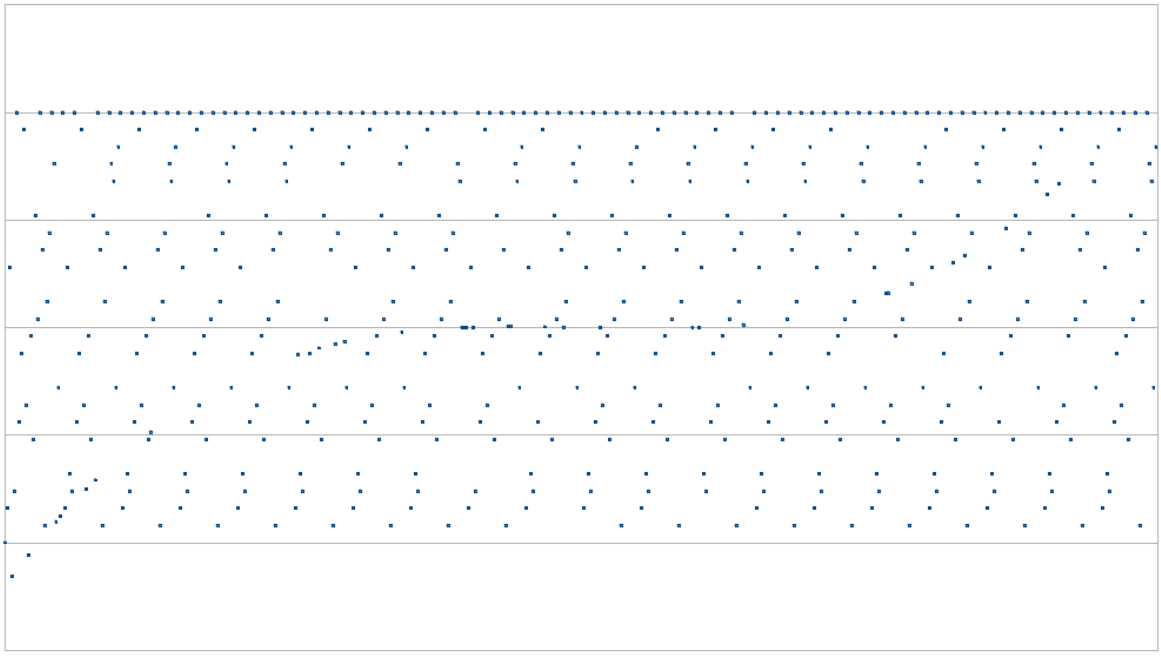


Figura 12 Punti che non rappresentano un polinomio.

Se si ha l'abilità o la possibilità di verificare OGNI punto di un polinomio, il problema è semplice. Ma cosa accade se invece si ha la possibilità di richiedere il valore del polinomio:

$$f(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \cdots + a_1 \cdot x + a_0$$

solo in un numero di punti superiormente limitato ? quanti punti è necessario richiedere per un polinomio di grado minore di un D noto, per avere la desiderata certezza che sia effettivamente un polinomio di grado minore di D ? Chiaramente 'D' punti non sono sufficienti in quanto tale quantità individua univocamente un polinomio di grado minore di D (pensiamo sempre che servono 2 punti per individuare una retta ossia un polinomio di grado 1, 3 per una parabola ossia un polinomio di grado 2, e cosi via ...), ma non ci dà la garanzia che anche tutti gli altri punti che non abbiamo verificato obbediscano a tale regola. Potenzialmente il polinomio potrebbe essere di grado uguale o superiore a D, e quindi la nostra verifica non sarebbe sufficientemente accurata. Tuttavia se vengono richiesti 'D+1' punti allora le cose cambiano, ed abbiamo qualche indicazione in più.

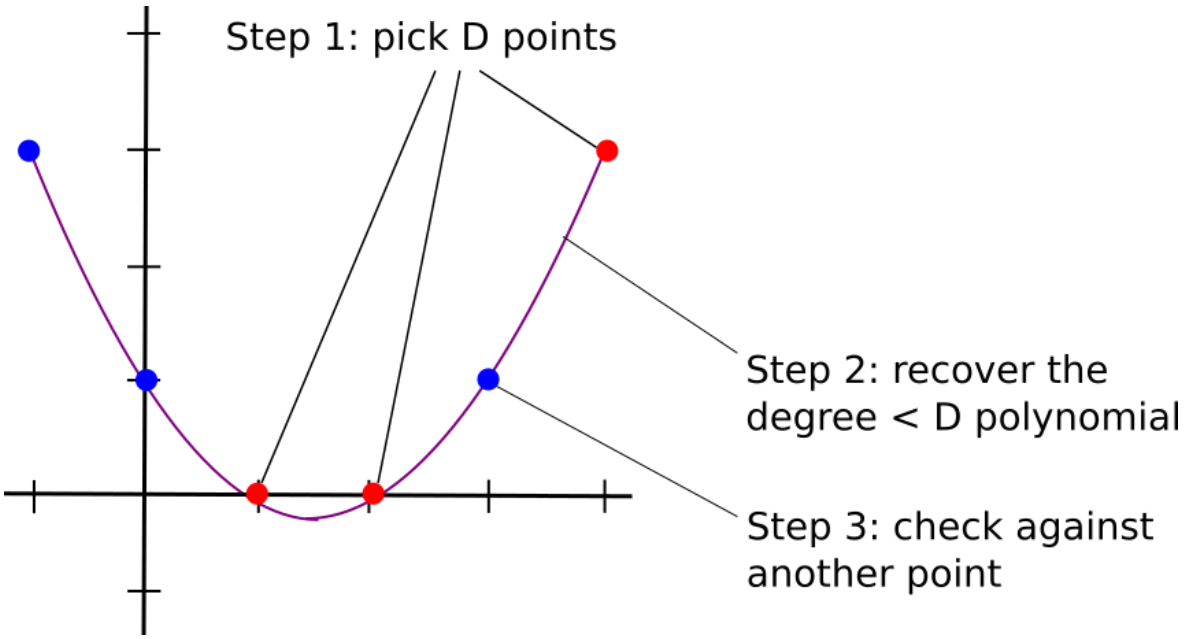


Figura 13 Esempio di controllo che dei punti a campione stanno su una parabola e quindi D<3.

L'algoritmo di controllo diventa in questo caso relativamente semplice: seleziona casualmente D punti e sfrutta uno dei vari algoritmi disponibili, per esempio quello di Lagrande, per trovare il polinomio di grado 'D-1' che li attraversa tutti. Successivamente seleziona casualmente un ulteriore punto (ricordiamoci che ci sono sempre un controllore che richiede il valore y in corrispondenza di x, ed un dimostrante non necessariamente onesto che fornisce la risposta) e verifica che lo stesso si trovi effettivamente sul medesimo polinomio precedentemente calcolato di grado D-1.

Facciamo notare che questo è un test di "prossimità", perchè c'è sempre la possibilità che la gran parte dei punti si trovi sullo stesso polinomio di grado minore di D, ma alcuni non lo siano e i D+1 punti che sono stati scelti casualmente siano capitati al di fuori di questi. Si può comunque concludere che se meno del 90% dei punti si trova su un polinomio di grado minore di D, il test da parte del controllore riuscirà a rilevarlo e considererà non valida la prova fornita. Aumentando il numero di richieste a D+k, la probabilità che il dimostrante passi il test per errore diminuirà esponenzialmente come $(1-p)^k = (1-0.9)^k = 0.1^k$.

Ma che cosa accade nel caso in cui D sia molto alto, e si voglia eseguire la verifica su meno di D punti ? è impossibile fare questo direttamente, per quanto detto precedentemente. Infatti non c'è modo di eseguire verifiche sufficienti che possano portare ad una conclusione (per esempio, non possono essere sufficienti 2 punti per capire se si tratta di una retta o di una parabola). Tuttavia è possibile fare questo (ci sono circa 20 anni di ricerca sull'argomento) indirettamente chiedendo al dimostrante di fornire dei dati ausiliari, raggiungendo una altissima efficienza in termini di complessità computazione. Questo è ciò che si ottiene tramite il protocollo FRI o "Fast RS IOPP" che sta per "Fast Reed Solomon Interactive Oracle Proofs of

Proximity". Reed Solomon sono noti da tempo per la teoria sui codici a correzione d'errore, che usano anch'essi i polinomi. La ragione intuitivamente è che dati 5 punti qualsiasi che sappiamo essere disposti su una parabola, se per un qualunque motivo se ne perdono 2 qualunque, i valori degli stessi sono recuperabili dagli altri 3 (sufficienti ad individuare la parabola in ogni suo punto).

1.3.1 Un primo sguardo al concetto di 'sub linearità'

Per provare che quanto sopra è possibile, partiremo da un protocollo relativamente semplice e con poche complicazioni, ma che ottiene l'obiettivo di verifiche sub-lineari, ossia provare la prossimità ad un polinomio di grado minore di D con un numero di richieste inferiore a D.

L'idea è la seguente: supponiamo di avere N punti, con N pari a circa 1 miliardo (1.000.000.000 o 10^9), e che gli stessi si trovino su un polinomio di grado minore di 1 milione (1.000.000 o 10^6). Si tratta di trovare un polinomio basato su due variabili x,y ossia una funzione $z = f(x, y)$ del tipo seguente:

$$1 + x + xy + x^5 \cdot y^3 + x^{12} + x \cdot y^{11}$$

tale per cui $f(x) = g(x, x^{1000})$. Ciò potrebbe essere fatto come segue: il termine di f(x) di grado k ossia per esempio:

$1744 \cdot x^{185423}$

viene scomposto nel modo seguente:

$$1744 \cdot x^{185423} = 1744 \cdot x^{185423\%1000} \cdot y^{\frac{185423}{1000}} = x^{423} \cdot y^{185}$$

E' semplice verificare che se $y = x^{1000}$, l'espressione di cui sopra è equivalente. In sostanza si è ottenuto un polinomio che differentemente da quello di partenza, ha grado 'composto' minore di 10^3 (in altre parole, considerando x costante o y costante, il polinomio è di grado minore di 1000). Infatti avremmo per esempio che:

$$1744 \cdot x^{999.999} = 1744 \cdot x^{999.999\%1000} \cdot y^{\frac{999.999}{1000}} = x^{999} \cdot y^{999}$$

Nella primo passaggio del protocollo, il dimostrante presenta il valore di g(x,y) sull'intero quadrato i cui punti sono:

[1 ... N] x {x^{1000}, 1≤ x ≤ N}

Il che corrisponde ad un miliardo di coordinate per le colonne (ricordiamo che N=10^9), e ad un miliardo di coordinate per le righe che corrispondono ai valori di $y = x^{1000}$. In realtà abbiamo spiegato che per ragioni di privacy si presenta un 'merkle root hash' in cui le foglie sono i valori di tale funzione, che verrà verificata a campione dal controllore (con meccanismi simili ai 'lightweight clients' dei bitcoin per i 'payment verification checks'). La diagonale di tale quadrato, seguendo la quale si ottiene che x=y, rappresenta invece i valori:

$$g(x, y) = g(x, x^{1000}) = f(x)$$

In altre parole, si è trasformata una funzione ad una singola variabile x in una funzione a due variabili, in cui ci si muove su un'area piana e per ciascuna coppia di variabili si ottiene una determinata 'altezza' della superficie discreta o continua (dipendentemente dal campo di numeri in cui si lavora), vedi l'esempio di Figura 14.

Il controllore poi sceglie casualmente alcune dozzine di righe e di colonne, e per ciascuna riga o colonna chiede al dimostrante di produrre il valore di (per esempio) 1010 punti (sul miliardo disponibili) con il vincolo in ogni caso che ALMENO UNO di essi (per ciascuna riga/colonna) si trovi sulla diagonale. Ricordiamo che vogliamo dimostrare che il polinomio ha grado inferiore a 1000, per cui 1000 punti ci servono ad individuare il polinomio ed altri k punti (in questo caso k=10) per verificare che gli stessi si trovano sul medesimo

polinomio. Il dimostrante deve rispondere con tali punti, assieme eventualmente ai rami del Merkle tree che dimostrino che gli stessi fanno parte degli stessi punti inizialmente prodotti sempre dal dimostrante (e quindi il polinomio non è cambiato). A questo punto il controllore verifica i rami del Merkle tree, e che i punti forniti dal dimostrante si trovino effettivamente su un polinomio di grado 1000.

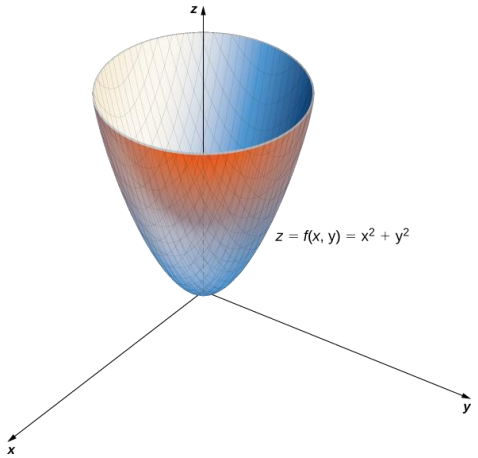


Figura 14 Esempio di funzione tridimensionale, nel caso specifico un paraboloide.

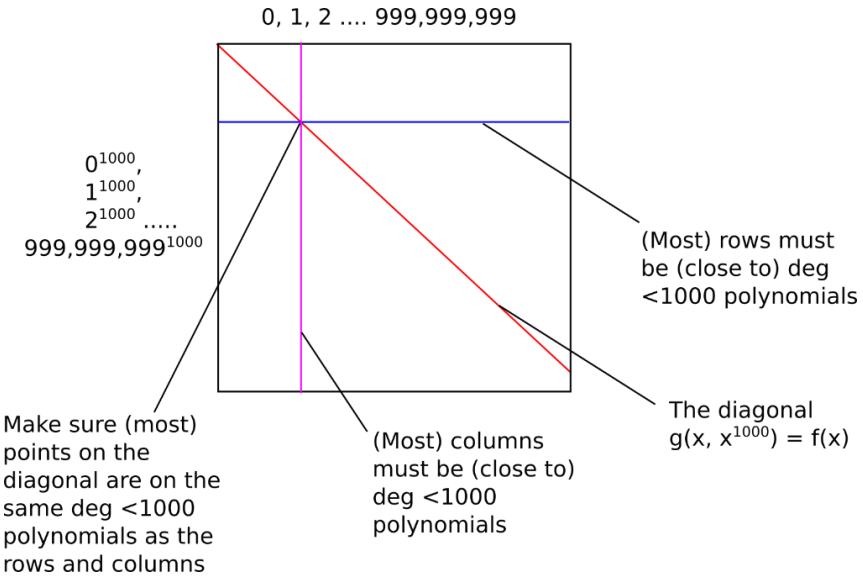


Figura 15 Punti della funzione tridimensionale f(x), sul piano delle coordinate piane (x,y) = $(\boldsymbol{x, x^{1000}})$.

Si noti in **Figura 15** che quando ci si muove lungo una riga o una colonna, stiamo sostanzialmente trasformando un polinomio del tipo seguente:

$$f(x,y) = f(x,x^{1000}) = 1 + x + xy + x^5 \cdot y^3 + x^{12} + x \cdot y^{11} + \cdots + x^{980} \cdot y^{995}$$

in un polinomio del tipo seguente:

$$f(x = k, y) = 1 + x + xy + x^5 \cdot y^3 + x^{12} + x \cdot y^{11} + \cdots + x^{980} \cdot y^{995}$$

quando ci muoviamo lungo una colonna, ossia in altre parole **x=k**, mentre quando ci muoviamo lungo una riga diventa costante $y = x^{1000}$, per cui il polinomio diventa:

$$f(x, y = k) = 1 + x + xk + x^5 \cdot k^3 + x^{12} + x \cdot k^{11} + \cdots + x^{980} \cdot k^{995}$$

Si noti che sempre in Figura 15 le ordinate sono rappresentate in scala logaritmica, in quanto sono i valori $\boldsymbol{x^{1000}}$ rappresentati su scala lineare. In realtà, pensando come esempio più semplice alla funzione $\boldsymbol{(x, x^2)}$, ciò che si sta facendo è considerare una funzione bidimensionale e quindi 'in un mondo più ampio' rispetto a prima, i punti in cui la funzione di partenza e la seconda coincidono sono i punti del piano in cui $\boldsymbol{y = x^2}$ e sarebbero quindi i punti di una parabola, non di una retta.

Tale approccio fornisce al controllore una prova statistica che la gran parte delle righe sono popolate da polinomi di grado inferiore a 1000, la gran parte delle colonne sono popolate da polinomi di grado inferiore a 1000, e che la diagonale invece si trova sul prodotto di due polinomi di grado inferiore a 1000 e sia quindi un polinomio di grado inferiore a $1000 \cdot 1000 = 10^6$.

Il controllore che richiede 1010 punti a caso (quindi 9 punti in più del minimo indispensabile che è 1001) per ciascuna delle 30 righe e ciascuna delle 30 colonne, in totale riceverà circa 1010*(30+30) = 60600 punti ossia molti meno del milione che avrebbe invece dovuto richiedere in partenza. Il dimostrante invece dovrà calcolare e fornire i valori per TUTTA la matrice in questione, che sono invece $10^9 \cdot 10^9 = 10^{18}$. In applicazione reali in cui il campo di lavoro è enormemente più grande di questo esempio, ciò è infattibile sia per il controllore che per il dimostrante, ma l'esempio fa comunque capire la direzione in cui ci stiamo muovendo.

In tutti gli algoritmi 'zero knowledge' dimostrare la correttezza di un calcolo senza riverlarne i dettagli deve essere invariabilmente più impegnativo rispetto a verificarne la correttezza. Solitamente i protocolli bilanciano tra:

- complessità computazione del dimostrante e del controllore
- numero di interazioni richieste tra le due parti
- lunghezza della prova prodotta per dimostrare la correttezza del calcolo

1.3.2 Matematica modulare, campi finiti di interi

Prima di proseguire aggiungendo complessità al protocollo, avremo bisogno di una piccola digressione nel mondo dell'aritmetica modulare. In particolare ci sono gruppi finiti di numeri in cui possono essere definite le operazioni di addizione, sottrazione, moltiplicazione e divisione, ma a differenza dei numeri naturali, interi, reali o immaginari, sono un elenco FINITO per quanto eventualmente grande. Ciò è di fondamentale importanza nel campo della crittografia e degli algoritmi perchè elimina lo spazio per errori legati ai problemi di arrotondamento o di numeri che diventerebbero troppo grandi (migliaia di cifre per intenderci) per poter essere correttamente gestiti.

Consideriamo per esempio un numero primo p=11, e ridefiniamo le 4 operazioni per il gruppo di numeri seguente:

$Z_{11}, x \in [0,1,2,3,4,5,6,7,8,9,10]$

In questo mondo si applicano tutte le regole che già conosciamo, con la sola differenza che i risultati delle 4 operazioni non possono mai superare il valore di p del gruppo. Quando il risultato di una somma supera p, il risultato è il resto della divisione intera per p. L'analogia come vedremo è quella degli orologi, se sono le 11 e si dice '3 ore dopo', si intendono le 2 del pomeriggio lavorando nello spazio dei numeri con p=12. Si procede analogamente per le moltiplicazioni, per esempio:

$(5 + 9)_{mod\ 11} = 14_{mod\ 11} = 3$
$(5 + 15)_{mod\ 11} = 20_{mod\ 11} = 9$
$(13 + 15)_{mod\ 11} = 13_{mod\ 11} + 15_{mod\ 11} = 2_{mod\ 11} + 4_{mod\ 11} = 6_{mod\ 11}$
$(13 + 15)_{mod\ 11} = 28_{mod\ 11} = 6_{mod\ 11}$

$(5 \cdot 9)_{mod\ 11} = 45_{mod\ 11} = 1$
$(5 \cdot 15)_{mod\ 11} = 75_{mod\ 11} = 9$
$(13 \cdot 15)_{mod\ 11} = 13_{mod\ 11} \cdot 15_{mod\ 11} = 2_{mod\ 11} \cdot 4_{mod\ 11} = 8_{mod\ 11}$
$(13 \cdot 15)_{mod\ 11} = 195_{mod\ 11} = (17 \cdot 11 + 8)_{mod\ 11} = (17 \cdot 11)_{mod\ 11} + 8_{mod\ 11} = 8_{mod\ 11}$

Ma possiamo anche definire l'operazione di **divisione** nel modo intuitivo seguente. Per esempio supponiamo si debba trovare l'inverso di 2 nel mondo dei numeri in Z_{11}:

$2^{-1} = \frac{1}{2}$

L'inverso di 2 è un numero $x \in Z_{11}$, tale per cui:

$(x \cdot 2)_{mod\ 11} = 1$

Possiamo facilmente trovare questo numero procedendo per tentativi, essendo lo spazio Z_{11} molto piccolo:

$(6 \cdot 2)_{mod\ 11} = 12_{mod\ 11} = 1$

Allo stesso modo, questi sono gli inversi per tutti i valori nello spazio finito Z_{11}:

$1^{-1}_{mod\ 11} = 1, \quad 2^{-1}_{mod\ 11} = 6, \quad 3^{-1}_{mod\ 11} = 4, \quad 4^{-1}_{mod\ 11} = 3, \quad 5^{-1}_{mod\ 11} = 9$

$6^{-1}_{mod\ 11} = 2, \quad 7^{-1}_{mod\ 11} = 8, \quad 8^{-1}_{mod\ 11} = 7, \quad 9^{-1}_{mod\ 11} = 5, \quad 10^{-1}_{mod\ 11} = 10$

E per quanto riguarda l'operazione di sottrazione ? ragionando analogamente a prima, si supponga di voler trovare il negativo del numero 2 in Z_{11}, è quel numero x tale per cui:

$(-2 + x)_{mod\ p} = 0$

Possiamo facilmente trovare questo numero procedendo per tentativi, essendo lo spazio Z_{11} molto piccolo:

$(2 + 9)_{mod\ p} = 11_{mod\ p} = 0$

Allo stesso modo, questi sono i numeri negativi per tutti i valori nello spazio finito Z_{11}:

$-1_{mod\ p} = 10, -2_{mod\ p} = 9, \quad -3_{mod\ p} = 8, \quad -4_{mod\ p} = 7, \quad -5_{mod\ p} = 6$
$-6_{mod\ p} = 5, \quad -7_{mod\ p} = 4, \quad -8_{mod\ p} = 3, \quad -9_{mod\ p} = 2, \quad -10_{mod\ p} = 1$

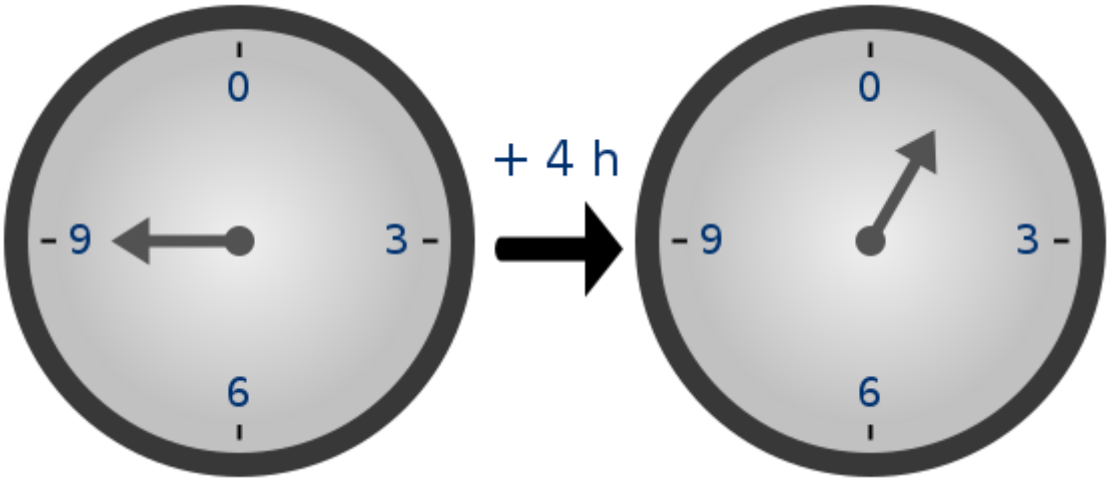


Figura 16 L'aritmetica modulare è spesso definita come la matematica degli orologi.

Il teorema di Fermat applicato in questi ambiti ha una interessante conseguenza. Se p-1 è multiplo di un determinato numero k, allora la funzione $y = f(x) = x^k$ ha una immagine piccola, ossia una quantità limitata di possibili valori di uscita. Per esempio con:

$p = 17, \;\; k = 2$

Ci sono solamente:

$\frac{p-1}{k} + 1 = 9$

possibili valori dell'immagine $f(x) = x^2$, come si deduce graficamente da qui:

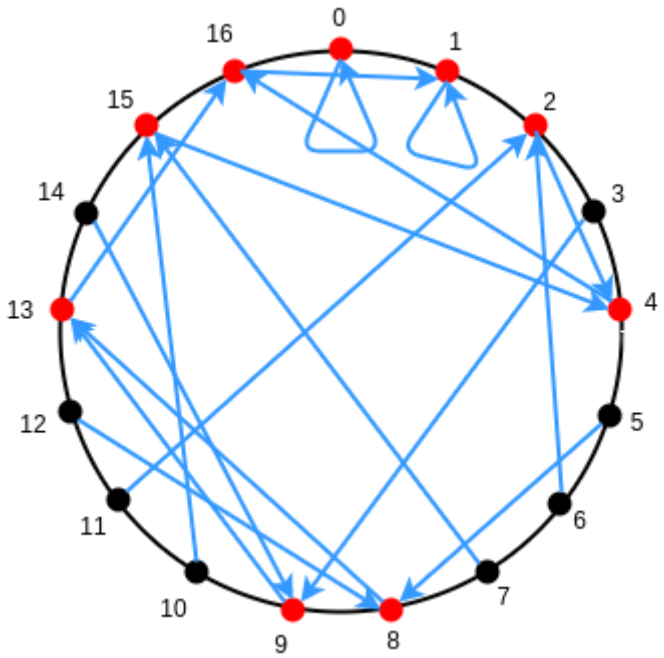


Figura 17 Valori al quadrato nel campo dei numeri Z16.

Per p=17 e k=16 ci sono solamente 2 possibili valori: 0 e 1. Tali proprietà sono sfruttate anche nell'algoritmo di ZK Stark per ottimizzare i calcoli dei polinomi, sfruttando i campi finiti di Galois in cui i coefficienti dei polinomi possono assumere solo valori binari, in cui k=4 e p è un numero di 256 bit.

Per un ripasso sulla teoria dei **gruppi** di numeri, rimandiamo al link seguente:

https://zeroknowledgedefi.com/2021/07/20/group-theory/

1.3.3 Un po' di efficienza in più

Ora ci muoviamo in direzione di una versione più complicata del protocollo, con il modesto obiettivo di ridurre la complessità del dimostrante da 10^{18} a 10^{15}, per poi ridurla a 10^{9} (il che inizia a rendere le cose gestibili, in quanto la complessità è lineare e non quadratica). Invece di lavorare su numeri 'reali' e potenzialmente infiniti, come detto si lavorerà su "campi finiti" e quindi superiormente limitati di numeri interi positivi.

Sceglieremo quindi di lavorare su Z_p con p= 1,000,005,001 in quanto:

3. tale valore è maggiore di 1 miliardo, in quanto l'obiettivo è avere uno spazio potenziale dei punti pari ad 1 miliardo
4. è un numero primo
5. p-1 è un multiplo dispari di 1000

$y = x^{1000}$ avrà quindi una dimensione massima di 1.000.006 valori. La diagonale dell'area bidimensionale precedentemente considerata, avrà quindi solamente 1.000.006 righe. Conseguentemente abbiamo ridotto il numero totale di punti nell'area (una sorta di griglia) a $10^9(colonne) \cdot 10^6(righe) = 10^{15}$ elementi. In altre parole:

$$f(x,y) = f(x,y+p) = f(x,y+2p) = f(x,y+3p) = \cdots$$

$$f(x,y+1) = f(x,y+1+p) = f(x,y+1+2p) = f(x,y+1+3p) = \cdots$$

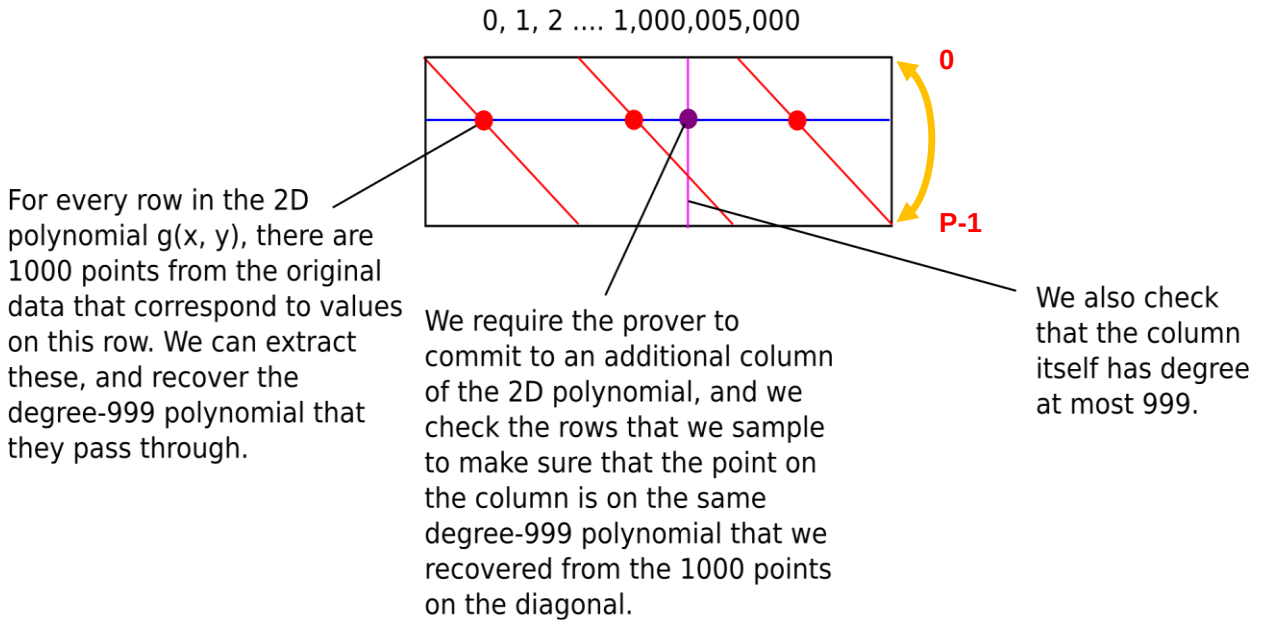


Figura 18 Nuova area limitata nel numero di righe.

Ma possiamo spingerci oltre, in particolare il dimostrante può inviare al controllore i valori di g su una singola colonna, questo perchè la funzione di partenza contiene già oltre 1000 punti su qualunque riga, quindi è possibile ricavare da essi il polinomio di grado minore di 1000, e verificare poi che lo stesso sulla colonna

casualmente scelta abbia il valore inviato dal dimostrante. Infine si verifica anche che la colonna stessa abbia valori di g compatibili con un polinomio di grado inferiore a 1000.

Nota del traduttore: quest'ultimo passaggio è totalmente fumoso e incomprensibile, andrebbe sicuramente spiegato in modo più esaustivo ed approfondito.

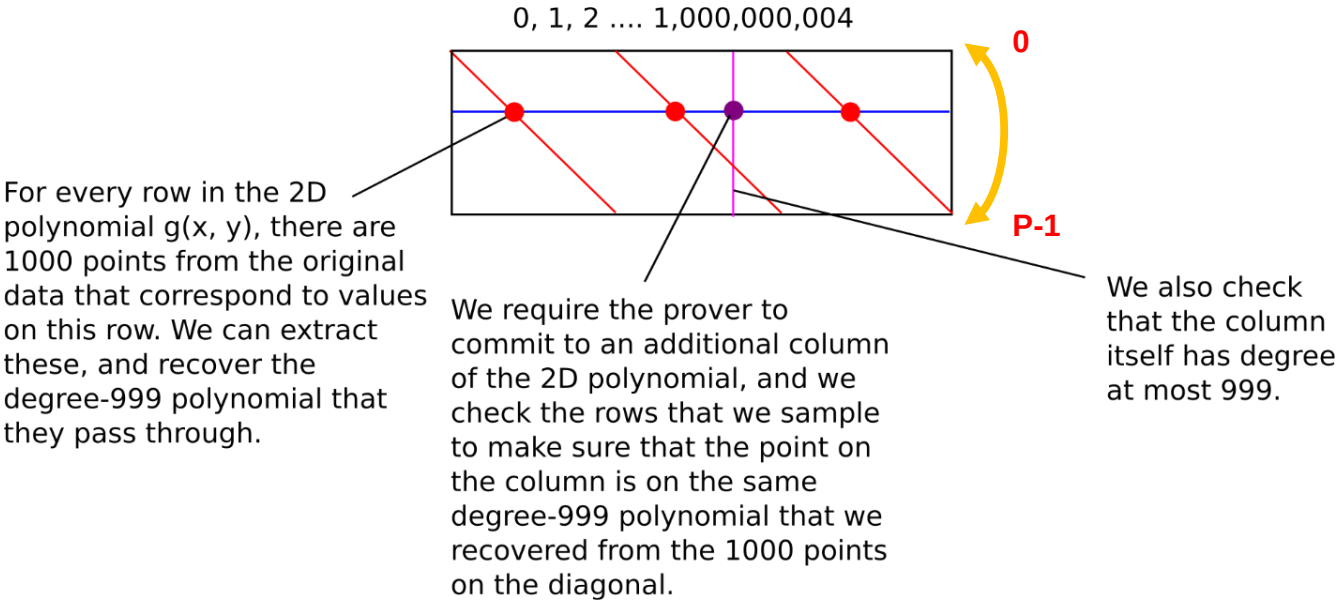


Figura 19 Nuova area in cui il dimostrante fornisce valori solo su una colonna.

La complessità del controllore è sublineare, mentre quella del controllore è decresciuta a 10^9, rendendola quindi lineare rispetto allo spazio o alla dimensione del gruppo aritmetico in cui si opera.

1.3.4 Ancora più efficienza

La complessità del dimostrante a questo punto è diventata la più bassa possibile, ma si può lavorare sulla complessità del controllore fino a ridurla da quadratica a logaritmica. Ciò si fa elaborando un algoritmo ricorsivo, in cui quanto descritto allo step precedente non si fa più considerando una funzione in due variabili in cui il grado massimo rispetto ad x ed y è il medesimo, ma si considera invece una funzione:

$$f(x) = g(x, x^2)$$

In tal modo, si 'scompone' il problema in uno di complessità molto più semplice e gestibile: vista in questo modo, considerando sempre l'area bidimensionale precedente, fissando il valore sulla colonna e muovendoci sulle righe, la funziona che si ottiene deve essere una retta in quanto è funzione di x che è di grado 1. Sarà quindi sufficiente in una riga verificare 3 valori: due che si trovano sulla diagonale, ed uno che si trova sulla colonna scelta in modalità casuale. Invece muovendoci lungo le colonne abbiamo una funzione di grado N/2 e quindi vanno controllati N/2+k punti. A titolo di esempio si prenda il polinomio seguente:

$$P_0(x) = 5x^5 + 3x^4 + 7x^3 + 2x^2 + x + 3 = (3x^4 + 2x^2 + 3) + x \cdot (5x^4 + 7x^2 + 1) = g(x^2) + x \cdot h(x^2)$$
$$P_0(-x) = g(x^2) - x \cdot h(x^2)$$

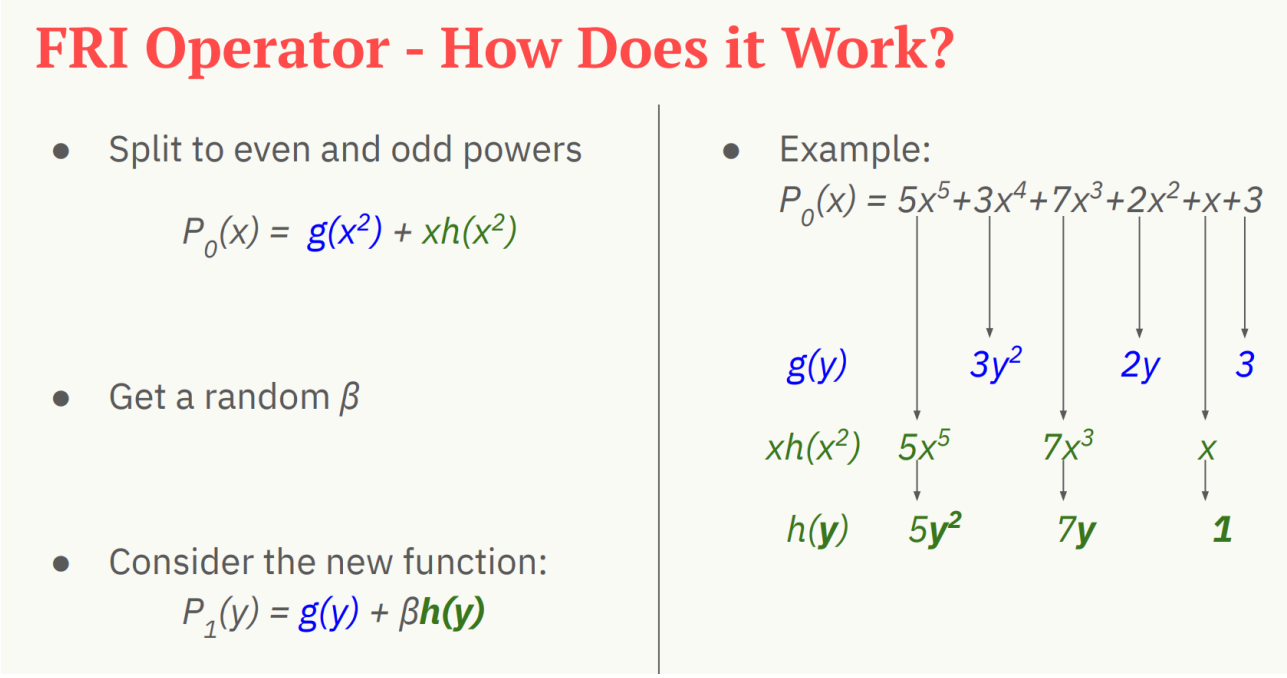


Figura 20 Esempio di scomposizione polinomiale.

Precedentemente avevamo estratto una funzione per ridurre di un fattore 1000 il grado del polinomio:

$$f(x,y) = f(x,x^{1000})$$

mentre qui viene ridotto di un fattore 2:

$$P_1(x) = g(x^2) + x \cdot h(x^2) = f(x,x^2)$$

Se sommiamo membro a membro le due equazioni di cui sopra, valide per ogni x nello spazio Z_p:

$$P_0(x) + P_0(-x) = 2g(x^2)$$

$$g(x^2) = \frac{P_0(x)+P_0(-x)}{2}$$

Sottraendo membro a membro invece otteniamo:

$$P_0(x) - P_0(-x) = 2x \cdot h(x^2)$$

$$h(x^2) = \frac{P_0(x)-P_0(-x)}{2x}$$

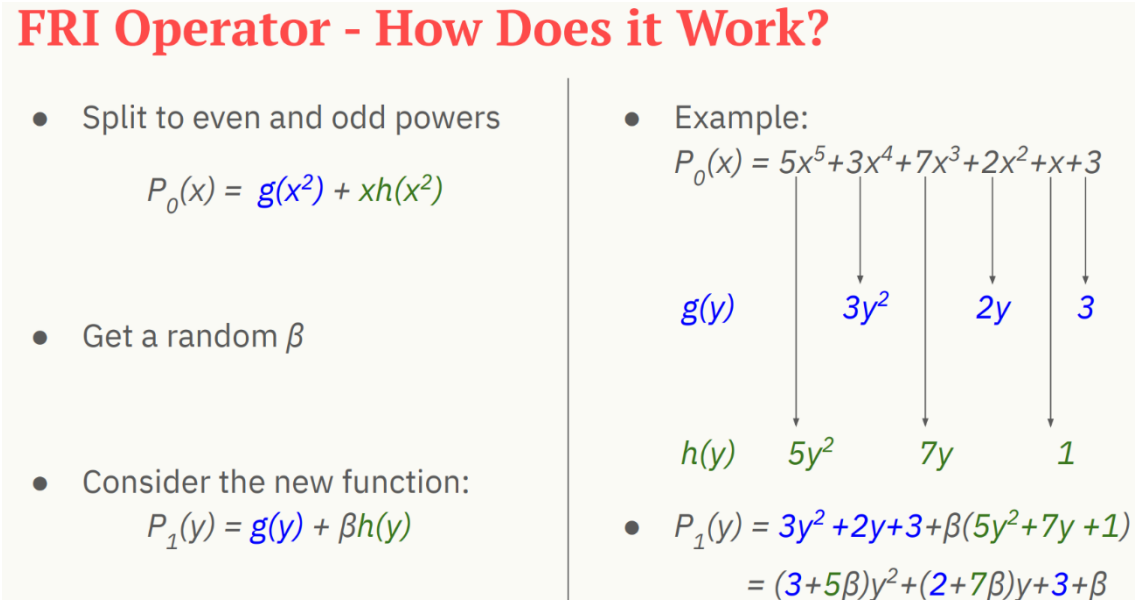


Figura 21 Esempio di scomposizione polinomiale (slide di Starkware).

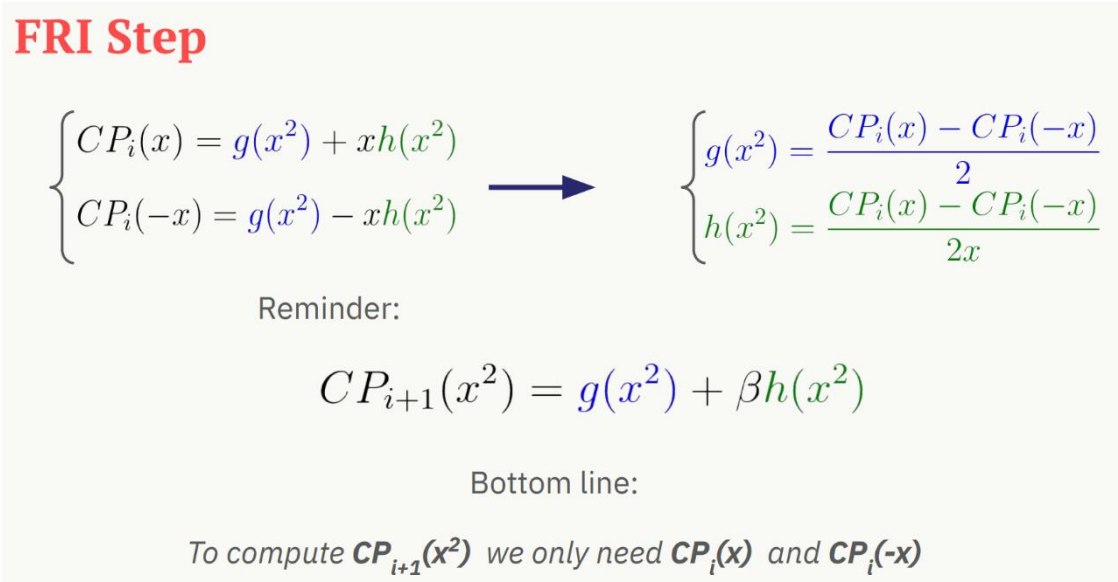


Figura 22 Relazioni tra polinomi ai vari step (slide ufficiali Starkware).

(nella figura sopra tratta dalle presentazioni di Starkware nei video di youtube c'è un errore nella funzione $g(x^2)$ in alto a destra)

Si noti che il denominatore è un numero pari, in uno spazio opportuno finito Z_p esiste sempre l'inverso di un valore del tipo $2x$ appartenente allo stesso spazio Z_p, cosa che non è invece vera in generale (riprenderemo questo punto più avanti). L'inverso di un numero $x \epsilon Z_p$ è un numero $y \epsilon Z_p = x^{-1}$ tale per cui $x \cdot y = x \cdot x^{-1} = 1$.

Ad ogni iterazione, abbiamo in ogni caso una relazione ben definita tra i valori del polinomio all'iterazione (i+1)-esima, e i valori dei due polinomi all'iterazione i-esima, cosa che consente di controllare ad ogni step che il dimostrante non stia inventando dei valori di sana pianta, ma che gli stessi facciamo riferimento sempre ad un polinomio, ed in particolare allo stesso polinomio di partenza che è la soluzione computazionale in corrispondenza dei suoi zeri:

$$P_{i+1}(x^2) = g_i(x^2) + \beta \cdot h_i(x^2) = \frac{P_i(x)+P_i(-x)}{2} + \beta \frac{P_i(x)-P_i(-x)}{2x}$$

β è un simbolo random scelto dal controllore ed è importante per rendere la vita difficile ad un dimostrante non onesto. Scendere nei dettagli in questo caso è complicato e non sono stati trovati nei link forniti a livello di documentazione. Ad ogni step inoltre, si passa a dimostrare che un polinomio è di grado dimezzato rispetto al precedente, e viene dimezzato anche lo spazio finito in cui si opera Z_p, come si deduce dalla slide seguente:

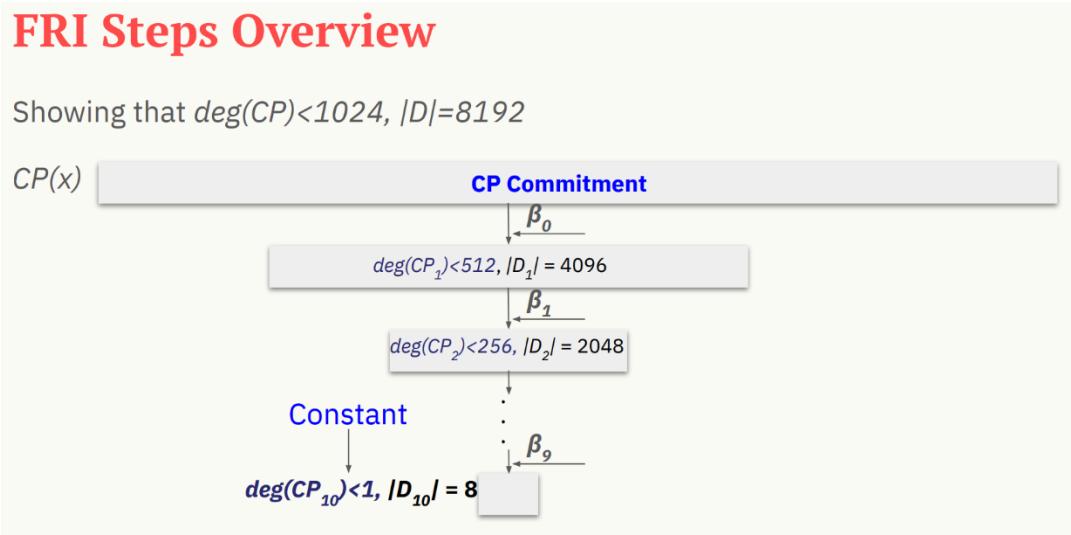


Figura 23 Rappresentazione dell'algoritmo FRI (slide ufficiali Starkware).

In questa invece si cerca di evidenziare la relazione tra i valori del polinomio allo step (i+1) con quelli calcolati e forniti allo step precedente.

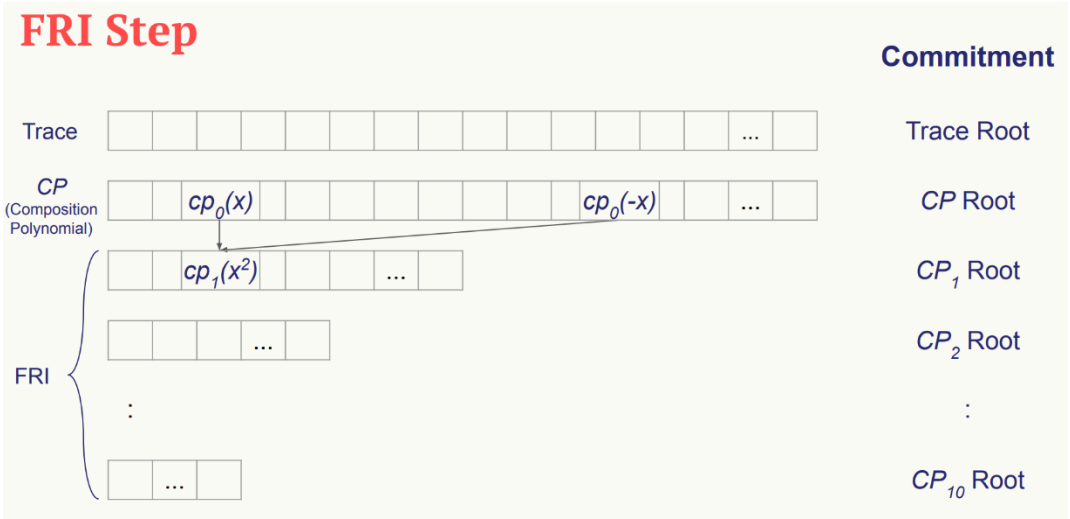


Figura 24 Poichè c'è relazione tra f(x) e f(-x) ad ogni step lo spazio S di dimezza (slide ufficiali Starkware).

Ritornando all'area di partenza in cui si considerano x e x^2 rispettivamente sulle ascisse e sulle ordinate, abbiamo che fissando il valore delle ordinate $y = x^2 = k$ la funzione lungo la quale ci si muove è quella di una retta:

$$g(x^2) + \beta x \cdot h(x^2) = k + \beta x \cdot k$$

mentre fissando il valore di x ci si muove su un polinomio di grado n/2. Quindi per un determinato layer i, si verificano i punti su una colonna **g** del polinomio scelta in modo randomico, e per ognuno dei punti scelti si eseguono le verifiche sulla riga dei punti con x^2 costante, analizzando i due punti sulla diagonale e verificando che i 3 punti stiano su una retta. Le colonna analizzata in questo layer diventa la diagonale del layer successivo, in cui si dovrà analizzare una colonna scelta a caso in uno spazio dimezzato di P, e con un grado del polinomio da verificare dimezzato a sua volta. Dopo un numero logaritmico di step, si arriverà al punto in cui la colonna da analizzare risulterà una costante (cosa che si verificherà analizzando il valore di 2 punti).

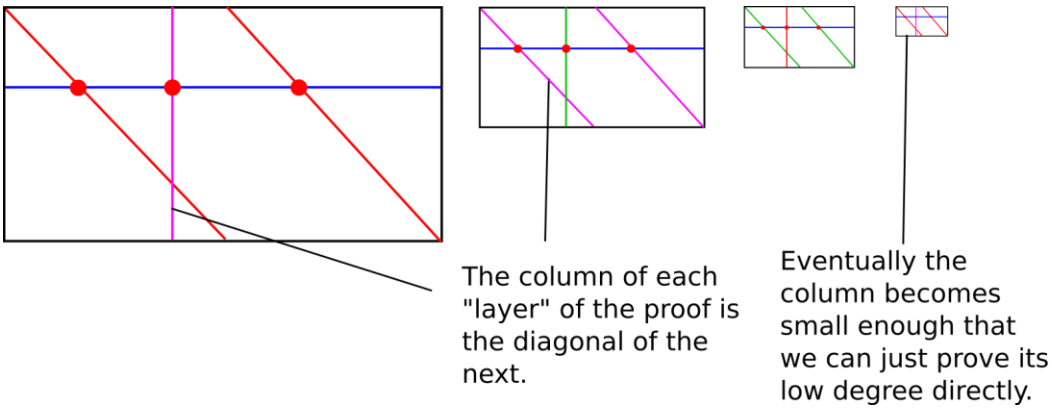


Figura 25 Relazione tra aree, punti e layer successivi.

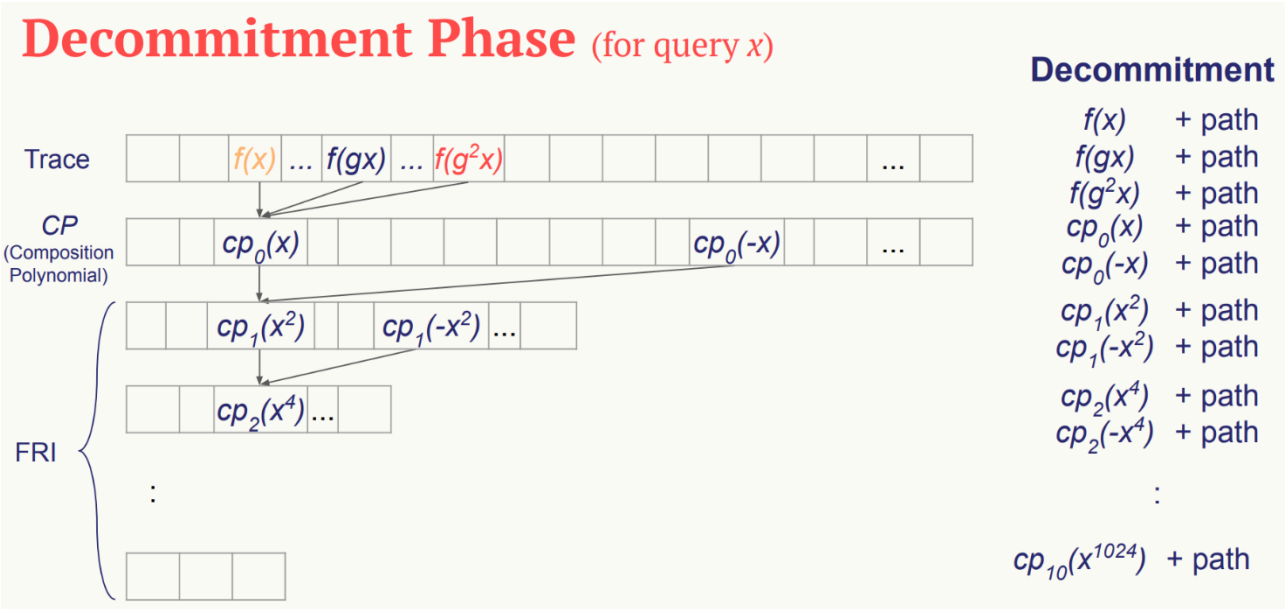


Figura 26 Relazioni tra polinomi ai vari step (slide ufficiali Starkware).

Ogni 'commitment' o 'fornitura di prova' comporta la trasmissione del merkle root hash della sequenza dei punti di quel polinomio per tutti i valori di x nello spazio di lavoro (che ricordiamo, anch'esso si dimezza nelle dimensioni ad ogni step grazie alla relazione univoca esistente tra f(x) e f(-x)).

Il protocollo viene quindi ripetuto per log N volte, ad ogni step considerando una componente randomica e non predicibile per rendere la vita difficile a dei dimostranti disonesti, in ogni caso le prove che ne risultano non sono enormi, la verifica di complessità diventa logaritmica al crescere del numero N e quindi estremamente efficiente.

Il protocollo FRI ha anche altre modifiche e ottimizzazioni: per esempio utilizza un campo finito di Galois, in cui il modulo è una potenza di 2 e i coefficienti sono binari. L'esponente utilizzato per le righe è tipicamente 4 e non 2, ossia una $f(x, x^4)$. Tali modifiche aumentano ulteriormente l'efficienza, ma non sono di per sè fondamentali a capire il funzionamento del protocollo. Dal sito di:

https://zeroknowledgedefi.com/

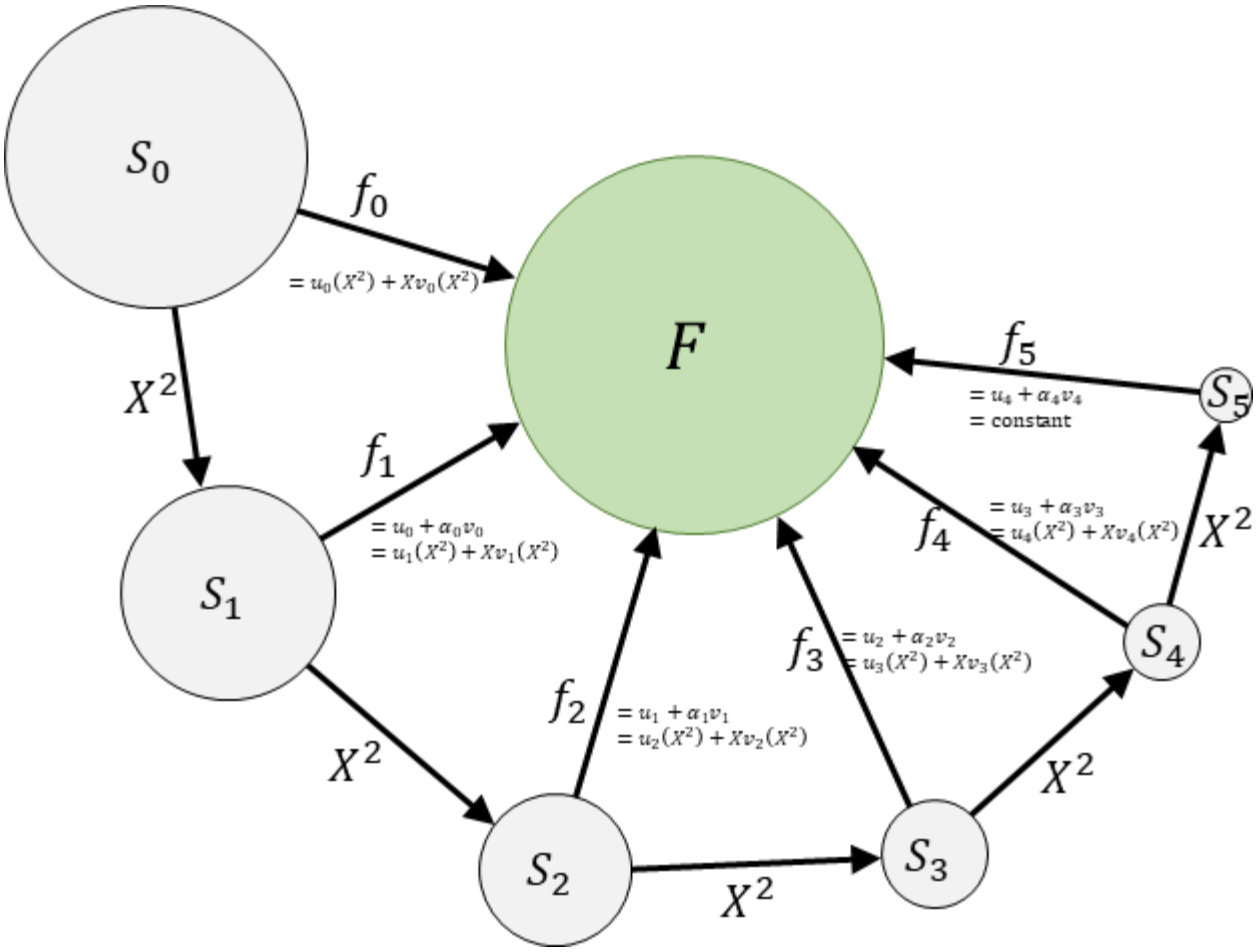


Figura 27 Altra rappresentazione del protocollo Stark.

1.3.5 Campi di Galois

Quanto scritto in chiusura del precedente paragrafo omette dei dettagli in effetti non fondamentali, ma utili ed importanti per chi vuole capire le cose fino in fondo. La cosa è approfondita (anche se a mio avviso manca un esempio chiarificatore) al link seguente:

https://zeroknowledgedefi.com/2021/12/26/starks-ii-is-that-a-polynomial/

al paragrafo "**Binary Fields**". In particolare ricordiamo che il controllore ad ogni step verifica che i nuovi punti scelti casualmente tramite il root hash della nuova funzione a grado dimezzato, obbediscano alla seguente relazione matematica:

$$P_{i+1}(x^2) = g_i(x^2) + \beta \cdot h_i(x^2) = \frac{P_i(x)+P_i(-x)}{2} + \beta \frac{P_i(x)-P_i(-x)}{2x}$$

La domanda è ... come facciamo a garantire che un punto sia divisibile per 2 nel campo in cui si opera ? normalmente finchè si opera in un campo Z_p con p numero primo, l'inverso di 2 esiste sempre ed è (p+1)/2 in quanto:

$\frac{p+1}{2} \cdot 2 = (p+1)_{mod\ p} = 1$

I campi di Galois sono campi finiti di ordine 2^k (Zk Stark opera su 2^{256}, per chi opera nel campo delle reti e sa qualcosa di IpV6 può solo immaginare quanto sia grande questo spazio ...) in cui i valori sono esclusivamente binari ossia 0 oppure 1, lo spazio di lavoro è quindi in modulo 2. Conseguentemente le operazioni di somma sono più semplici da gestire in quanto sono 'carrierless' o senza riporto, in altre parole 0+0=1+1=0. Tali campi si prestano a realizzazioni particolarmente efficienti per programmi ed algoritmi che richiedono tanti calcoli, in quanto le operazioni si riducono a AND o XOR logici effettuati su bit, e quindi sono anche ottimizzabili usando hardware dedicato. Tuttavia in questo mondo dei numeri modulo 2, NON ESISTE la divisione per 2, in quanto significherebbe dividere per 0: non esiste alcun valore $x\epsilon[0,1]$ tale per cui $x \cdot 2 = 1$. L'unico valore che ha l'inverso è 1, e il suo inverso è 1 stesso. Come si risolve quindi questo problema ? Traducendo dal sito riportato da qui in avanti ...

In aggiunta, il fatto che $-x$ sia uguale ad x ovunque significa che la mappa quadratica di x ossia le funzioni $f(x^2)$ sono funzioni "uno ad uno" invece che "due ad uno". Conseguentemente il dominio S_i allo step i-esimo ha le stesse dimensioni del dominio S dello step precedente, aumentando in generale la complessità dell'esecuzione algoritmica.

Il tutto può essere risolto per i campi binari utilizzando la mappa (o funzione, o traslazione) seguente:

$\vartheta_i(x) = x^2 + a_i x$

in altre parole non si esegue solamente il quadrato di x ma si aggiunge un elemento non nullo in modo che S_i sia 'chiuso' aggiungendo a_i. Sulla base della precedente espressione, il polinomio $f_i(x)$ può essere scomposto come segue:

$f_i(x) = g_i\big(\vartheta_i(x)\big) + x \cdot h_i(\vartheta_i(x))$

In cui g_i ed h_i sono polinomi di grado minore di M/2. Si può verificare che in un campo binario:

$\vartheta_i(x + a_i) = \vartheta_i(x)$

per tutti i punti del dominio, ossia ϑ_i è una funzione due ad uno. Alice può risolvere le solite equazioni allo step i-esimo per ogni x come segue:

$g_i\big(\vartheta_i(x)\big) = a_i^{-1} \cdot (\, f(x) + f(x + a_i)\,)$

$h_i\big(\vartheta_i(x)\big) = f_i(x) + x \cdot g\big(\vartheta_i(x)\big)$

Possiamo garantire che esiste un elemento a_i con le proprietà richieste scegliendo il dominio inziale S come un sottospazio additivo chiuso del campo finito. Se è cosi, questo sarà vero anche per ciascun sottodominio S_i, ed a_i può essere un qualunque elemento non nullo. L'algoritmo risulta quindi essere esattamente lo stesso descritto nella precedente sezione.

Nota del traduttore: non credo che quanto sopra abbia chiarito ogni cosa, anzi ... al di la del formalismo matematico probabilmente corretto, quello che manca a mio avviso sono sempre dei SEMPLICI esempi che facciano capire le cose. Cosi come per l'algoritmo Stark manca un esempio semplice che faccia vedere cosa accade nei sui 4-5 passi successivi di esecuzione.

1.3.6 Generalizzazione

Questa parte è stata presa dalla fonte seguente:

https://kitten-finance.medium.com/low-degree-testing-in-zk-stark-part-2-7c729118d10c

la spiegazione non è cosi semplice, ma pare utile per quanto appena detto, sull'uso dei campi di Galois e di un fattore 4 per le colonne allo scopo di aumentare l'efficienza (o almeno questo è come l'ha interpretato il traduttore).

Come ulteriore esempio si prenda il seguente polinomio di grado 200:

$$f(x) = x^{200} - 3x^{50} + 4x^6 - 2$$

Possiamo ridurre il grado di tale polinomio da 200 a 3, se introduciamo le variabili seguenti:

a= x, b= x^4, c= x^{16}, d= x^{64}

Il risultato è riassunto nella tabella seguente:

$f(x) =$	x^{200}	$-3x^{50}$	$+4x^6$	-2
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$\downarrow$	$(x^4)^2(x^{64})^3$	$-3x^2(x^{16})^3$	$+4x^2(x^4)$	-2
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$g(a,b,c,d) =$	b^2d^3	$-3a^2c^3$	$+4a^2b$	-2

Il termine x^n in $f(x)$ diventa a^i, b^j, c^k, d^l nel polinomio $g(a,b,c,d)$ con i,j,k,l valori nello spazio Z_4 ossia [0,1,2,3]. Pertanto il termine x^{200} in $f(x)$ diventa $d^3c^0b^2a^0d^3 = d^3b^2$ nel polinomio $g(a,b,c,d)$ poichè 200 in base 4 è proprio 3020. Quindi il polinomio di grado 200 diventa un polinomio di grado 3 nelle variabili a,b,c,d o anche:

$$f(x) = x^{200} - 3x^{50} + 4x^6 - 2 = g(a,b,c,d) = g(x, x^4, x^{16}, x^{64}) = b^2d^3 - 3a^2c^3 + 4a^2b - 2$$

Come altro esempio, un polinomio di grado 65535 può diventare un polinomio di grado inferiore a 3 con il cambio di variabili seguente:

$$f(x) = g(a,b,c,d,e,f,g,h) = g(x, x^4, x^{16}, x^{64}, x^{256}, x^{1024}, x^{4096}, x^{16384})$$

Si è quindi trasformata una funzione polinomio di x in una funzione multi-dimensionale ad N variabili, e si passa a dimostrare che ciascuno dei polinomi 'parziali' (ossia muovendoci lungo una sola dimensione e mantenendo costanti tutte le altre) è inferiore al grado N. Quindi nel caso precedente:

$$f(x) = g(a,b,c,d) = g(x, x^4, x^{16}, x^{64})$$

Supponiamo che il dimostrante Bob fornisca un merkle tree di tutta f(x), da questo prende un valore random α e fissa la prima dimensione del polinomio g:

$$g(\alpha, x^4, x^{16}, x^{64})$$

Bob dimostra ad Alice (il controllore) che tale polinomio è di grado inferiore a 4. Successivamente produce un merkle root di tutti i valori della funzione $g(\alpha, c^4, c^{16}, c^{64})$ per ogni c appartenente allo spazio considerato, estrae a caso un valore β dall'hash prodotto e considera:

$$g(\alpha, \beta, x^{16}, x^{64})$$

A questo punto Alice verifica che nuovamente il polinomio in questione ha grado minore di 4 per ogni c appartenente allo spazio considerato:

$g(\alpha, \beta, c^{16}, c^{64})$

Si procede in questo modo fino all'ultima dimensione, nel momento in cui i 4 parametri sono 'fissati' la funzione di output è una costante, in modo deterministico dopo un numero fisso di passi.

1.3.6.1 Aritmetica dei campi finiti

Grazie alla suddetta aritmetica dei campi finiti, per il dimostrante calcolare lungo una colonna tutti i valori del polinomio è relativamente semplice e più veloce in quanto le operazioni sono svolte in un campo di Galois con .

In particolare sia p un numero primo di grandi dimensioni, e si prenda una radice primitiva β. Il gruppo Z_p risulta ciclico e generato da β, in altre parole per ogni $x \epsilon Z_p$ esiste un valore n tale per cui $x = \beta^n{}_{mod\ p}$. Prendiamo per esempio Z_5, abbiamo β=2 in quanto:

$2^1 = 2 \qquad 2^2 = 4 \qquad 2^3 = 8_{mod5} = 3 \qquad 2^4 = 16_{mod5} = 1 \qquad 2^5 = 32_{mod5} = 2$

Conseguentemente ogni intero appartenente al gruppo Z_5 è scrivibile come $\beta^n{}_{mod\ p} = \beta^{n+p-1}{}_{mod\ p}$.

Quindi invece di lavorare con la funzione $f(x)$ si può lavorare invece con la funzione $f(\beta^n)$, e quindi:

$f(a, b, c, d)\ viene\ scritta\ come\ f(\beta^a, \beta^b, \beta^c, \beta^d)$

Per costruzione abbiamo che:

$f(x, x^4, x^{16}, x^{64}) = f(\beta^n, \beta^{4n}, \beta^{16n}, \beta^{64n}) = f\{\mathrm{n}, 4\mathrm{n}, 16\mathrm{n}, 64\mathrm{n}\}$

in altre parole come abbiamo visto ogni 4 volte il valore (p-1)/4 si ritorna al valore di partenza in un ciclo infinito. Quindi abbiamo anche che:

$f\left(\beta^{n+\frac{p-1}{4}}\right) = f\left(n + \frac{p-1}{4},\ 4n,\ 16n,\ 64n\right)$

e più in generale:

$$f(\beta^n) = \{n, \qquad 4n, 16n, 64n\}$$

$$f(\beta^{n+(p-1)/4}) = \{n + (p-1)/4, \quad 4n, 16n, 64n\}$$

$$f(\beta^{n+2(p-1)/4}) = \{n + 2(p-1)/4, \quad 4n, 16n, 64n\}$$

$$f(\beta^{n+3(p-1)/4}) = \{n + 3(p-1)/4, \quad 4n, 16n, 64n\}$$

Quindi il concetto è che ogni 4 valori di intervallo (p-1)/4, i valori del secondo, terzo e quarto parametro della funzione si ripetono e sono sempre gli stessi, a causa della ciclicità del campo finito e del particolare valore

scelto come parametro 'p'. Poichè il polinomio in questione è di grado inferiore a 4, possiamo interpolare questi 4 valori per calcolare velocemente:

$$\{\mathrm{a}, 4\mathrm{n}, 16\mathrm{n}, 64\mathrm{n}\},\ \forall\ \mathrm{a} \in Z_p$$

Chiaramente p può essere molto grande e (p-1)/4 è comunque un range elevato di valori, ma data la correlazione tra il valore di x sulle varie colonne ed il fatto che ogni valore cicla su se stesso se elevato esponenzialmente alla p-1, il risultato è una migliore efficienza.

1.3.6.2 Bob passo 1

(1) Bob calcola $f(a)$ per ogni a in Z_p

(2) Bob costruisce un Merkle Tree M1 per tutti i valori di $f(a)$. Bob può dimostrare che ogni valore che ha calcolato di $f(a)$ è reale e genuino sfruttando il Merke Tree prodotto. Alice si potrebbe comportare come un light client verifier.

(3) Bob sfrutta il root hash per scegliere un valore A pseudo randomico. Poichè l'output di un hash non è prevedibile (gli algoritmi di PoW si basano proprio su questo), Bob non può barare e sapere in anticipo quale valore di A sarà scelto.

(4) Bob a questo punto calcola:

$$\{A, 4b, 16b, 64b\}, \forall\ b \epsilon [0, \frac{p-1}{4} - 1]$$

(5) Bob calcola un altro Merkle Tree M2 per tutti questi valori

(6) Bob utilizza il root hash di M2 per scegliere 40 valori pseudo-casuali $[b_1, b_2, b_3, \dots, b_{39}, b_{40}]$ nel range $[0, \frac{p-1}{4} - 1]$ e come precedentemente spiegato non può barare perchè il modo in cui vengono scelti dall'hash è predeterminato, ma l'hash non è calcolabile a priori.

(7) Bob invia i 40 valori $[A, 4b_1, 16b_1, 64b_1], \dots, [A, 4b_{40}, 16b_{40}, 64b_{40}]$ e 40 Merkle proofs (????) ad Alice.

(8) Si noti che c'erano 160 valori di f in M utilizzati per calcolare questi 40 valori di g. Bob invia i 160 valori di f e 160 Merkle proofs ad Alice.

1.3.6.3 Alice passo 1

Alice verifica tutte le prove fornite: le 40 di G contenute in M2 e le 160 di f contenute in M1. Alice verifica anche che ogni valore di g viene dall'interpolazione dei corrispondenti 4 valori di f. Conseguentemente è molto probabile che il grado di a sia minore di 4, poichè Alice ha verificato 40 valori (34 in più di quanto strettamente necessario) che Bob non poteva sapere prima, e perchè il campo finito in cui si opera Z_pè enorme.

1.3.6.4 Bob passo 2

(1) Bob sfrutta il root hash M2 per scegliere un valore B pseudo randomico. Poichè l'output di un hash non è prevedibile (gli algoritmi di PoW si basano proprio su questo), Bob non può barare e sapere in anticipo quale valore di B sarà scelto.

(2) Bob a questo punto deve calcolare:

$$\{A, 4b, 16c, 64c\}, \forall\ c\ \epsilon [0, \frac{p-1}{16} - 1]$$

Tuttavia risulta che tali valori sono stati GIA' calcolati allo step precedente al punto 4:

$$\{A, 4b, 16b, 64b\}, \forall\ b \epsilon [0, \frac{p-1}{4} - 1]$$

In particolare:

$$\left\{A, 4(c+\tfrac{p-1}{16}), 16c, 64c\right\} = \left\{A, 4(c+\tfrac{p-1}{16}), 16(c+\tfrac{p-1}{16}), 64(c+\tfrac{p-1}{16})\right\} = \{A, 4c, 16c, 64c\}$$

$$\left\{A, 4(c+2\tfrac{p-1}{16}), 16c, 64c\right\} = \left\{A, 4(c+2\tfrac{p-1}{16}), 16(c+2\tfrac{p-1}{16}), 64(c+2\tfrac{p-1}{16})\right\} = \{A, 4c, 16c, 64c\}$$

$$\left\{A, 4(c+3\tfrac{p-1}{16}), 16c, 64c\right\} = \left\{A, 4(c+3\tfrac{p-1}{16}), 16(c+3\tfrac{p-1}{16}), 64(c+3\tfrac{p-1}{16})\right\} = \{A, 4c, 16c, 64c\}$$

Poichè il grado del polinomio in b è minore di 4, Bob può interpolare il risultato e calcolare velocemente qualunque *{A, 4b, 16c, 64c}*

(5) Bob calcola un altro Merkle Tree M3 per tutti questi valori

(6) Bob utilizza il root hash di M3 per scegliere 40 valori pseudo-casuali $[c_1, c_2, c_3, \dots, c_{39}, c_{40}]$ nel range $[0, \frac{p-1}{16} - 1]$ e come precedentemente spiegato non può barare perchè il modo in cui vengono scelti dall'hash è predeterminato (per esempio il primo byte ne individua uno), ma l'hash non è calcolabile a priori.

(7) Bob invia i 40 valori $[A, 4B, 16c_1, 64c_1], \dots, [A, 4B, 16c_{40}, 64c_{40}]$ e 40 Merkle proofs (????) ad Alice.

(8) Si noti che c'erano 160 valori di *{A, 4b, 16b, 64b}* in *M2* utilizzati per calcolare questi 40 valori di $[A, 4B, 16c, 64c]$. Bob invia i 160 valori di $[A, 4B, 16b, 64b]$ e le 160 'Merkle proofs' in M2 ad Alice.

1.3.6.5 Alice passo 2

Alice verifica tutte le Merkle proofs: 40 prove di $[A, 4B, 16c, 64c]$ e 160 prove di $[A, 4B, 16b, 64b]$ in M2. Inoltre verifica che ciascun valore di $[A, 4B, 16c, 64c]$ proviene dalla interpolazione dei corrispondenti 4 valori di $[A, 4B, 16b, 64b]$.

Di conseguenza è alquanto probabile che il grado del polinomio in b sia minore di 4, poichè Alice ha verificato la cosa sfruttando 40 diversi valori, Bob non ha potuto scegliere arbitrariamente i punti in cui effettuare la verifica, e lo spazio dei numeri in cui si opera è enorme.

NOTA del traduttore: il post in questione potrebbe avere senso, ma è scritto e spiegato male. E per spiegarlo meglio, bisognerebbe averlo capito e sapere con certezza dove sono gli errori. Purtroppo non sono ancora arrivato a questo punto.

1.4 FRI protocol

Dal sito ufficiale di Starkwave, in particolare dal seguente link:

https://medium.com/starkware/low-degree-testing-f7614f5172db

traduciamo la parte che descrive il protocollo, anche se indicativamente è stato già fatto precedentemente. Aggiungiamo anche qualcosa dal link seguente:

https://zeroknowledgedefi.com/2021/12/13/starks-i-polynomials-everywhere/

in particolare per descrivere la fase di setup iniziale che è comunque importante.

Prima di procedere, riassumerò brevemente le operazioni preliminari per affrontare il problema descritto nel presente post. Ci sono due parti, Bob (il dimostrante) e Alice (il controllore). Bob esegue dei calcoli che riguardano un numero enorme N di passi di calcolo. Ogni calcolo consiste di un numero fisso di variabili, i cui valori si trovano in un campo finito F. Ad ogni step dei calcoli, le variabili vengono aggiornate, fino ad arrivare ad uno step finale, con i valori finali delle variabili.

y0,0	y0,1	y0,2	y0,3
y1,0	y1,1	y1,2	y1,3
y2,0	y2,1	y2,2	y2,3
⋮	⋮	⋮	⋮
yN-2,0	yN-2,1	yN-2,2	yN-2,3
yN-1,0	yN-1,1	yN-1,2	yN-1,3

Figura 28 Traccia di esecuzione.

Bob invia la traccia di questa esecuzione dei calcoli ad Alice, che vuole verificarne la correttezza ma non ha nè il tempo nè le risorse di calcolo di Bob per ripetere i calcoli. Quindi Bob ha bisogno anche di inviare qualche una prova ad Alice che la convinca che i calcoli sono effettivamente stati eseguiti correttamente.

Alice deve poter eseguire il controllo in forma breve (succinctness), in un numero molto inferiore di passi rispetto ad N, chiedendo se serve delle informazioni 'accessorie', ma non troppe. Naturalmente Bob potrebbe anche mentire, per cui è necessario verificare che Bob non solo fornisca dei punti corretti, ma che gli stessi siano appartenenti ad un polinomio.

Il protocollo consiste di due fasi: una fase di 'commit' ed una fase di 'richieste' da parte del controllore.

1.4.1 Fase di 'commit'

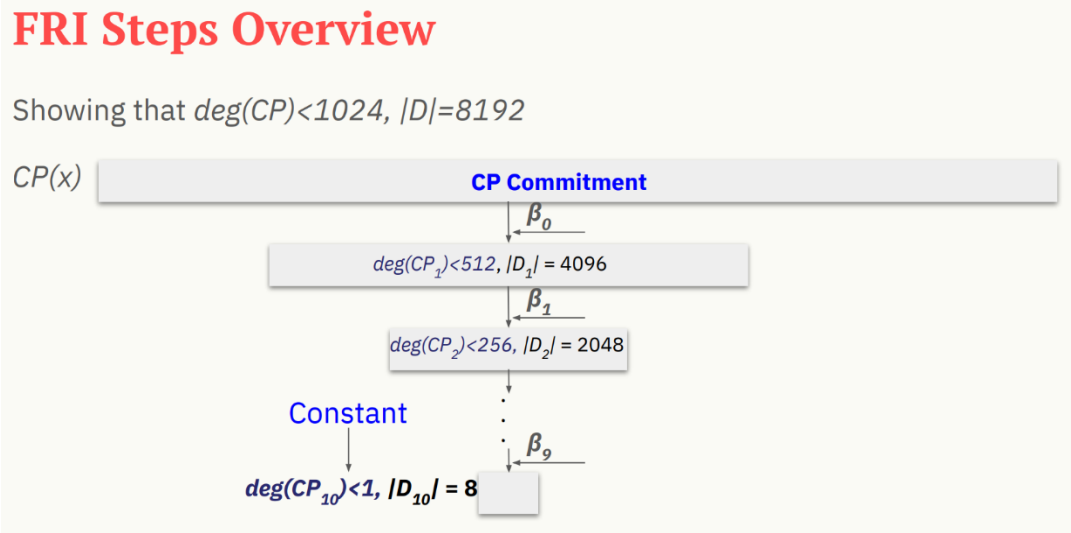
Il dimostrante suddivide il polinomio di origine $f_0(x) = f(x)$ di grado minore di d, in due polinomi di grado minore di d/2, $g_0(x)$ e $h_0(x)$ tali da soddisfare l'equazione seguente:

$$f_0(x) = g_0(x^2) + x \cdot h_0(x^2)$$

Il controllore estre casualmente un valore α_0 e lo invia al dimostrante, al quale chiede di fornire una dimostrazione di conoscenza del polinomio:

$$f_1(x) = g_0(x) + \alpha_0 \cdot h_0(x)$$

Si può in tal modo continuare ricorsivamente suddividendo a sua volta $f_1(x)$ in $g_1(x)$ e $h_1(x)$, scegliendo un valore casuale α_1, costruendo $f_2(x)$ e cosi via. Ad ogni passo il polinomio risulta di grado dimezzato, conseguentemente dopo log(d) passi, il risultato è un polinomio costante e il dimostrante può semplicemente inviare tale costante al controllore.



Una note importante riguardo i domini: perchè il protocollo descritto funzioni, è necessario che per ogni z nel dominio L, anche $-z$ sia ancora in L. Inoltre, la prova riguardante $f_1(x)$ non è su L ma su $L^2 = \{x^2 : x \epsilon L\}$. Poichè iterativamente si applicato lo step FRI, L^2 a sua volta dovrà soddisfare la regola descritta su z e $-z$.

Tali requisiti algebrici sono facilmente soddisfatti attraverso scelte naturali del dominio L (come per esempio un sottogruppo moltiplicativo la cui dimensione sia una potenza di 2), ed infatti coincide con quelli di cui abbiamo in ogni caso bisogno per beneficiare di algoritmi FFT (Fast Fourier Transform di calcolo dei polinomi) efficienti.

1.4.2 Fase di 'Query'

A questo punto dobbiamo verificare che il dimostrante non abbia mentito o in qualche modo imbrogliato. Il controllore estrae a caso uno z a caso nello spazio L e chiede di avere i valori $f_0(z)$ e $f_0(-z)$. Tali due valori sono sufficienti per determinare i valori di $g_0(z^2)$ e $h_0(z^2)$, come si può derivare dalle due equazioni seguenti già viste più volte precedentemente:

$$f_0(z) = g_0(z^2) + z\, h_0(z^2)$$
$$f_0(-z) = g_0(z^2) - z\, h_0(z^2)$$

Il controllore può risolvere questo sistema di equazioni e deduce i valori di $g_0(z^2)$ e $h_0(z^2)$, da questi poi calcola $f_1(z^2)$ che è una combinazione lineare delle due. Poi il controllore chiede il valore di $f_1(z^2)$ e verifica che lo stessa coincida con quello calcolato. Questo garantisce che il polinomio che si sta attualmente esaminando sia ricavato da quello dello step precedente secondo le equazioni evidenziato, e non sia generato dal nulla in modo fraudolento. Il controllore può continuare richiedendo $f_1(-z^2)$, si ricordi che $(-z^2)\epsilon L^2$e che i dati del polinomio erano stati forniti sempre su L^2, e dal valore di $f_1(-z^2)$ può dedurre $f_2(z^4)$. Il controllore continua in questo modo finchè dopo logD passi si arriva ad una costante. Tale valore

f_{log d}(z) è un polinomio costante il cui valore era stato già inviato dal dimostrante nella fase di 'commit' prima di scegliere z.

Per capire come sia difficile imbrogliare, si consideri il semplice problema per cui f_0 è zero sul 90% delle coppie della forma {z,-z}, i.e., $f_0(z) = f_0(-z) = 0$ e sia diverso da 0 nel rimanente 10%. Con probabilità del 10% il valore di z scelto casualmente cade nelle coppie 'cattive'. Non solo quel valore di α porterà ad $f_1(z^2)=0$, mentre il resto porterà ad $f_1(z^2)\neq 0$. Se il dimostrante bara sul valore di $f_1(z^2)$ sarà preso, per cui assumiamo

che non sia cosi per quello specifico step. Ad ogni modo, con alta probabilità ($f_1(z^2)$, $f_1(-z^2)$) sarà a sua volta una coppia 'cattiva' nel layer successivo (il valore di $f_1(-z^2)$ è poco importante dato che $f_1(z^2)\neq 0$).

D'altra parte, poichè siamo partiti da una funziona con il 90% di zeri, è improbabile che il dimostrante sarà in grado di andare vicino ad un polinomio di basso grado che sia diverso dal polinomio nullo. In sostanza, se il dimostrante bara, è probabile che finirà per dover inviare come valore costante del polinomio all'ultimo step.

Nel test che abbiamo appena descritto (e tenendo presente l'analisi di cui sopra) la probabilità che il controllore rilevi un dimostrante disonesto è solo del 10%, ossia la probabilità di errore è del 90%. Ciò si può esponenzialmente migliorare ripetendo la fase di richiesta per un certo numero di valori casuali di z. Per esempio scegliendo 850 valori, otteniamo una probabilità di errore di 2^{-128} che è sostanzialmente nulla.