|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DIMENSIONI  (Byte) | **abi.json** | **out** | **out.ztf** | **programma .zok** | **proof.json** | **proving.key** | **Cost to store proving.key on Ethereum blockchain** | **verification.key** | **witness** |
| **1 argomento**  **1 assert**  **[916 vincoli]** | 114 | 146.295 | 72.909 | 56 | 1.003 | 290.648 | 181.655.000 gas | 1.386 | 5.536 |
| **2 argomenti**  **2 assert**  **[1832 vincoli – x2]** | 191  (x1.67) | 292.534  (x2) | 148.966  (x2) | 92  (x1.64) | 1.003 | 580.440  (x2) | 362.775.000 gas | 1.386 | 11.616  (x2) |
| **3 argomenti**  **3 assert**  **[2748 vincoli – x3]** | 268  (x2.35) | 438.773  (x3) | 225.023  (x3) | 128  (x2.28) | 1.003 | 935.768  (x3.21) | 584.855.000 gas | 1.386 | 17.696  (x3.2) |
| **8 argomenti**  **8 assert**  **[7328 vincoli – x8]** | 653  (x5.7) | 1.169.968  (x8) | 609.060  (x8.35) | 308  (x5.5) | 1.003 | 2.319.192  (x7.98) | 1.449.495.000 gas | 1.386 | 48.855  (x8.82) |
| **1 argomento**  **2 assert**  **[1579 vincoli – x1.72]** | 114 | 258.653  (x1.77) | 124.639  (x1.71) | 76  (x1.35) | 1.003 | 499.736  (x1.72) | 312.335.000 gas | 1.386 | 9.592  (x1.73) |
| **1 argomento**  **3 assert**  **[2242 vincoli – x2.44]** | 114 | 371.011  (x2.54) | 176.369  (2.41) | 95  (x1.70) | 1.003 | 774.360  (x2.66) | 483.975.000 gas | 1.386 | 13.648  (x2.46) |
| **1 argomento**  **8 assert**  **[5557 vincoli – x6.06]** | 114 | 932.801  (x6.38) | 437.503  (x6) | 190  (x3.39) | 1.003 | 1.754.264  (x6.06) | 1.096.415.000 gas | 1.386 | 34.435  (x6.22) |
| **2 argomenti**  **1 assert**  **[1169 vincoli]** | 191 | 180.176 | 95.438 | 70 | 1.003 | 436.888 | 273.055.000 gas | 1.386 | 7196 |



Immagine che contiene testo, aquilone

Descrizione generata automaticamente

GRAFICO:

* la linea blu rappresenta la crescita di dimensione della proving.key aumentando i parametri e imponendo su ognuno di essi un singolo assert
* la linea arancione invece rappresenta la crescita di dimensione della proving.key considerando un singolo parametro, ma aumentando di volta in volta il numero degli assert imposti su di esso
* I numeri sulla sinistra sono la dimensione della proving.key in byte
* I numeri in basso indicano quanti parametri e assert sono stati usati, e vengono letti nel seguente modo (esempio):
  + - 2-2 | 1-2 🡪 2-2 è il dato relativo alla linea blu, e ci dice che stiamo considerando un programma con 2 parametri e 2 assert(uno per parametro). 1-2 invece è il dato relativo alla linea arancione e ci dice che stiamo considerando un programma con 1 parametro e 2 assert su di esso.

Come si può vedere anche dal grafico, la dimensione della proving.key aumenta linearmente all’aumentare del numero di parametri e assert su questi parametri.

Il costo di memorizzazione deve essere pagato una volta per ogni smart contract relativo al verificatore di zokrates, perché per ogni verificatore abbiamo una singola proving.key creata nel momento del setup e non più modificata (a meno di cambi nel codice zokrates).

Il costo di retrieve della proving.key invece è nullo, in quanto questa è una semplice funzione che legge lo stato, non lo modifica, e quindi è eseguita solo sulla macchina che la chiama (è infatti una funzione view).

Gia con un singolo parametro su cui verifichiamo un singolo assert possiamo notare che la dimensione della proving.key è molto elevata, e ciò non ci consente di memorizzarla sulla blockchain, in quando il limite di gas utilizzabile è circa 12M, ma il gas che si andrebbe a spendere per la memorizzazione è circa 181M.

Un primo approccio che si potrebbe utilizzare sarebbe quindi quello di utilizzare un servizio esterno di memorizzazione tradizionale (non basato sulla blockchain) per memorizzare la proving.key.

TABELLA:

CONSIDERAZIONI

* L’ultima riga della tabella, in grigio, rappresenta un programma con due parametri di ingresso, e un solo assert, cioè una sola condizione che comprende entrambi questi parametri. Un esempio potrebbe essere guadagno1 > guadagno2, con entrambi i guadagni input privati del programma. Possiamo notare che la dimensione della proving.key è minore del programma con 2 parametri e 2 assert, ma maggiore di quello con 1 parametro e 1 assert, e questo ci conferma che la dimensione della proving.key dipende sia dal numero di parametri che dal numero di assert.
* Sulla blockchain di Ethereum gni dato occupa uno slot di memoria da 32 byte. Il SET di un dato (cioè mettere lo slot da “zero” a un certo valore) costa 20K gas circa.
* Con una prova su Remix su un semplice smart contract per memorizzare un dato, ho visto che il costo è di circa 21684 gas. Per farlo ho considerato una funzione molto semplice che preso in input una stringa la memorizza, e eseguendola sempre sull’IDE Remix ho potuto vedere l’execution cost, cioè il gas speso per l’esecuzione.
* Considero comunque per semplicità 20K gas come costo nella seguente tabella.
* Dato che ogni slot è da 32 byte, per sapere quanto gas viene usato per memorizzare una stringa o un qualsiasi dato uso il seguente ragionamento:
  + - #slot = (dimensione\_dato\_in\_byte / 32)
    - Gas\_usato = #slot \* costo\_gas = (dimensione\_dato\_in\_byte / 32) \* 20K

BREVE DESCRIZIONE DEL PROCEDIMENTO

Per quanto riguarda la creazione dei vari file e la loro valutazione il procedimento è il seguente: per prima cosa si scrive un programma in linguaggio ZoKrates e lo si salva con estensione “.zok”. A questo punto il procedimento si divide in due:

1)Verifier

Il verificatore è colui che vuole attestare la veridicita di una prova che gli viene sottoposta dall’esterno. Esso riceverà una prova e dovrà verificare che essa sia giusta.

Per prima cosa dunque il verificatore prende il programma .zok e lo compila tramite il comando

“zokrates compile -i nomeFile.zok”

La compilazione genererà i seguenti file:

abi.json 🡪 al suo interno, in formato json vi sono indicati gli input e gli output del programma

out e out.ztf 🡪 sono due file binari che serviranno per creare le proving.key e la verification.key

Dopo la compilazione il verificatore tramite il comando “zokrates setup” genererà la verification.key e la proving.key

La proving.key dovrà poi essere fornita all’entita che vuole creare una proof.

Infine il verificatore crea lo smart contract in solidity tramite il comando “zokrates export-verifier”

2) Prover

Il prover è l’entità che vuole provare di essere a conoscenza di un certo fatto senza rivelarlo, e per questo creerà una proof che fornirà al verificatore.

Come primo passo anche lui deve compilare il programma .zok, con lo stesso comando del verificatore.

A questo punto deve recuperare la proving.key, che gli verrà fornita in qualche modo dal verificatore.

Una volta fatto ciò il prover può creare un witness, cioè un file che servirà per creare la prova vera e propria.

Il witness viene creato tramite il comando “zokrates compute-witness -a a1 a2 a3 …” dove a1 a2 a3 … sono gli eventuali parametri da passare al programma. Se gli argomenti sono corretti, il witness viene generato.

Il prover infine genera la proof, tramite il comando “zokrates generate-proof”. Questa proof è quella da inviare al verificatore; è un file dalle dimensioni costanti che contiene 3 valori.